Patodia Equation Solver

Advanced Mathematical & Scientific Computing Platform

A Practical University Guide

Expanded practical edition for Version 4.0

Panna Lal Patodia
Patodia Infotech Private Limited

11 November 2026

A practical guide for university professors, research scholars and students. Complete model inputs are paired with interpretation, independent mathematical checks, teaching activities and reproducible research workflows. Use this book as a working companion to the software rather than as a sequential mathematics textbook.

About this edition

This expanded edition retains the established modelling and numerical chapters, strengthens the practical routes for university users, and adds eighteen laboratory studies with complete inputs and mathematical checks. Chapters 33–39 cover differentiation, sampled data, interpolation, limits, special functions and exact integer utilities. Chapters 40–42 show how to choose a workflow, conduct a laboratory study and prepare a reproducible research record. Appendix D provides the complete updated Compute Functions reference, including a brief description, accepted domain and example for each function.

The 3.0.1 update adds multiple quadrature ranges, declared split points, and standalone singularity analysis. See Chapters 31 and 35 for worked examples, Chapter 32 for numerical diagnostics, and Chapter 2 for the Equation Generator and Fun with Mathematics.

The mathematical illustrations and historical Desktop screenshots are retained. Analytical reference plots illustrate mathematical behaviour and should not be interpreted as captures of a fresh software run. Examples distinguish mathematical reference values, historical test reports and checks that readers should perform in their installed application.

Version 4.0 adds programmable workflows. Chapter 3 introduces the language, the numerical chapters show service-specific workflows, and Chapter 43 combines several services in a complete study. Earlier build and validation records retain their original version numbers.

All mathematical functions, including every function available in Compute, are available to Version 4.0 workflows. Call shared core and scalar Compute functions directly in workflow expressions. Use a compute job for list-valued Compute output such as primefactors. Inside a service job, the service's own expression grammar applies. To use a Compute scalar in another service, evaluate it in the workflow and substitute the numeric result with @name.

How to use the book

Start with Chapters 1–4 for input and interpretation, or Chapter 40 for a route matched to your task. Select the application chapters relevant to your subject, then use Chapter 41 for complete laboratory studies. Chapter 42 explains research records and numerical comparisons. Consult Appendix D when choosing a Compute function; use the Word Navigation Pane or PDF bookmarks to move directly to a topic.

Commands and listings use US notation unless identified otherwise. Number-style conversion, project settings and licensing are discussed explicitly. This book is not a licence agreement and does not confer redistribution rights. References to supported modes are deliberately bounded; unsupported numerical structures require an appropriate specialist method or a separately validated extension.

Contents

Chapter 1 Introduction to Equation Solving
Chapter 2 Getting Started with Patodia Equation Solver
Chapter 3 Writing Equations and Constraints
Chapter 4 Understanding and Checking Solutions
Chapter 5 The Workflow Programming Language
Chapter 6 Classical Mathematics and Geometry
Chapter 7 Integer and Mixed Integer Systems
Chapter 8 Polynomial Equation Systems
Chapter 9 Transcendental Equation Systems
Chapter 10 Chemistry and Chemical Engineering
Chapter 11 Physics and Engineering Applications
Chapter 12 Economics and Other Applications
Chapter 13 Pell Equations and Very Large Integer Bounds
Chapter 14 General Quadratic Equations in Integers
Chapter 15 Weighted Sums of Squares
Chapter 16 Initial Value Ordinary Differential Equations
Chapter 17 Stiff Equations Events and Continuous Output
Chapter 18 Supported Singular Initial Value Problems
Chapter 19 Regular Two Point Boundary Value Problems
Chapter 20 Laplace and Poisson Equations on Rectangles
Chapter 21 Heat and Wave Equations in One Space Dimension
Chapter 22 Fredholm and Volterra Integral Equations
Chapter 23 Extended Precision and Numerical Reliability
Chapter 24 Automation and Integration
Chapter 25 Benchmarks Teaching and Further Exploration
Chapter 26 Mixed Boundary Conditions and Adaptive PDE Accuracy
Chapter 27 Sturm Liouville Eigenvalues and Eigenfunctions
Chapter 28 Linear Reaction Diffusion and Advection Diffusion
Chapter 29 Adaptive Integral Accuracy and Coupled Volterra Systems
Chapter 30 Parameter Studies and Numerical Visualisation
Chapter 31 Numerical Definite Integration on Finite and Infinite Intervals
Chapter 32 Convergence Verification and Reproducible Research
Chapter 33 Numerical Differentiation of Mathematical Expressions
Chapter 34 Differentiation and Interpolation of Tabular Data
Chapter 35 Finite One Sided and Infinite Limits
Chapter 36 Compute Functions and Special Functions
Chapter 37 Number Theory Functions and Exact Integers

Chapter 38 Linear Diophantine Families and Modular Results

Chapter 39 Integrated Research Challenges and Release Validation

Chapter 40 Choosing a Practical University Workflow

Chapter 41 Worked University Laboratory Studies

Chapter 42 From an Exploratory Run to a Reproducible Study

Chapter 43 A Complete Multi Service Workflow

Appendix A Desktop command and result reference

Appendix B Version 3 worked model catalogue

Appendix C Source basis and mathematical glossary

Appendix D Complete Compute Functions Reference

Complete listings use explicit multiplication and US number notation unless stated otherwise. Confirm CSV paths and saved settings before running an example. Mathematical references in the new laboratory studies are independent checks, not reports of a newly executed Windows test.

Chapter 1 Introduction to Equation Solving

An equation solver turns a stated mathematical relationship into candidate values or a numerical function. The most useful calculation begins before the Solve command: we must decide what is unknown, what data are fixed, which domain is intended and what kind of answer would resolve the question. A pair of integers, an equilibrium concentration, an oscillating trajectory and a temperature field are all mathematical solutions, but they require different inputs and different checks.

Patodia Equation Solver 4.0 combines several such tasks in one Windows Desktop. It retains simultaneous algebraic equations, inequalities, integer and binary domains, nonlinear search and extended precision. It adds exact methods for supported two-variable quadratic integer equations, initial-value ODEs, regular two-point boundary-value problems, selected singular IVP continuations, rectangular PDEs and second-kind integral equations. This book explains the mathematics and the Desktop workflow together. Its aim is to make a returned answer understandable and independently checkable.

1 1 Begin with the question rather than the software

Suppose two numbers have sum five and difference one. The unknowns are x and y; the two relationships are x+y=5 and x-y=1. Adding them gives 2x=6, hence x=3 and y=2. The computation is simple enough to do by hand, which makes it an excellent first software test. A correct input should return the same pair, and substitution should recover both original right-hand sides.

```
var x,y
x+y=5
x-y=1
```

The var declaration controls the order of the output columns. It does not supply a starting vector or change a real variable into an integer. The model is a set of simultaneous relationships, not a sequence of assignment statements. Reversing the order of the two equations does not change the intended mathematical solution.

Now change the second relationship to x*y=6. The pairs (2,3) and (3,2) both satisfy the model. A user who expects a single pair could wrongly dismiss one of them. If x and y refer to labelled objects, exchanging their values gives another ordered solution. If the objects are physically interchangeable, those two vectors can describe the same configuration. Numerical distinctness and physical distinctness must be discussed separately.

1 2 Distinguish unknown quantities fixed data and units

An equation connects quantities. Some quantities are to be found and others are supplied. In a spring model F=k*x+alpha*x^3, a prescribed force and measured coefficients leave displacement unknown. If all four symbols are left as free variables, the entered question is different and may have infinitely many solutions. Use constant declarations or numerical values for known data.

```
constant load=10
constant stiffness=2
constant cubic=1
stiffness*x+cubic*x^3=load
bound x=[0,5]
```

The trial displacement x=2 is rejected because 2*2+2^3=12, which does not equal ten; this quick substitution rejects that tempting guess. The actual positive solution lies between one and two: the residual is -7 at one and 2 at two. The derivative 2+3*x^2 is positive, so the residual is strictly increasing and there is exactly one real root. A rough bracket and a monotonicity argument can be more useful than an unsupported decimal guess.

Units are not automatically attached to variable names. If load is in newtons and x is in metres, the coefficient dimensions must be consistent. Converting displacement to millimetres without converting coefficients creates a different numerical model. Reporting more digits cannot repair that error. Define the basis of a calculation before entering it, especially for concentrations, temperatures, time and flow rates.

1 3 Use residuals as checks of the entered relationships

Write an equation L=R as a residual r=L-R. A numerical candidate is accepted when the relevant residual and feasibility tests meet the method's criteria. For x^2=2 at x=1.4142, the residual is -0.00003836. A more accurate candidate reduces that residual, but the size of the residual is not automatically the error in x.

At the repeated root of (x-1)^2=0, the candidate x=1.000001 has coordinate error 1e-6 and residual about 1e-12. The twelve small residual places do not mean twelve correct coordinate places. Near nearly dependent equations, small residuals can likewise accompany substantial changes in coordinates. Chapter 4 develops substitution, scaling, conditioning and independent verification in detail.

Exact integer routes provide a different form of evidence. A reported exact pair is checked by integer arithmetic against the original identity. A residual of zero then means an exact integer equality, not merely a tiny floating-point quantity printed as zero. Even so, verifying each pair is separate from proving that every pair has been enumerated.

1 4 Classify the mathematical problem

Question	Typical input	Intended output
Simultaneous algebraic equations	Equalities, domains and bounds	Isolated solution vectors or a search diagnostic
Integer quadratic equation	Exact coefficients and integer declarations	Exactly verified integer pairs within the applicable domain
Initial-value ODE	ode, derivatives, initial data and interval	A trajectory and possibly event records
Two-point ODE BVP	bvp, derivatives and endpoint conditions	A curve satisfying numerical collocation and boundary criteria
PDE	pde, supported operator, domain, boundary/initial data and grid	A sampled field on the requested grid
Integral equation	integral, kernel, forcing, interval and grid	A sampled function satisfying a discrete quadrature system
Regular eigenproblem	eigen with p q weight and homogeneous boundaries	Lowest eigenvalues and normalised modes
Known-function integral	quadrature with finite or infinite endpoints	One approximate definite integral and error diagnostics

A model for a function needs information beyond a single equality at one point. For an IVP that information is given at the initial independent coordinate. For a BVP it is distributed across endpoints. A PDE adds a spatial domain and boundary data. An integral equation specifies how values across an interval influence one another. The mode header tells the engine which interpretation to use.

1 5 Understand linearity and nonlinearity

An algebraic system is linear in its unknowns when they appear only to the first power without products between unknowns or nonlinear functions of them. Constants multiplying unknowns do not make a system nonlinear. A unique square linear system is usually a simpler task than a nonlinear system, but dependent or inconsistent linear equations still require mathematical interpretation.

Nonlinear models can have several isolated roots, repeated roots, curves of solutions or no real solution. The factorisation (x+2)*(x-1)*(x-3)=0 gives three real roots directly. The equation x^2+1=0 has no real root. The equation x+y=5 alone describes infinitely many real pairs. A finite list returned from a numerical search is not a complete representation of that line.

Linearity also applies to differential equations. The equation y''+0.5*y'+4*y=0 is linear in y and its derivatives, although the solution is an oscillating function. The equation y''=-sin(y) is nonlinear in the state but still affine in the highest derivative. Supported direct ODE input requires that highest derivative to enter affinely; it does not permit every implicit differential relation.

1 6 Make the domain part of the question

For sin(x)=0, unrestricted real solutions are x=k*pi for every integer k. In [-3,3] only zero occurs. In [0,2*pi] the endpoints and the intermediate root must be considered. Without a domain, asking for all roots can mean an infinite family. Use bounds because they express the problem, not just because a run seems slow.

For an integer count use int, pos int, nonneg int or binary as appropriate. A positive integer excludes zero. A bound such as [0,10] does not turn an unconstrained real variable into an integer. The intersection of declarations and bounds is the effective domain. In the Desktop, dialog bounds override model-text bounds, so a saved project should accompany results whenever those overrides matter.

An algebraic inequality such as x>=0 filters admissible points. It is not the same as an initial condition initial y=0. A PDE boundary such as boundary left u=0 prescribes a function on a domain side. Similar numbers occur in these statements, but their mathematical roles differ.

1 7 Connect geometry to equations

Two bamboos stand vertically, one 15 feet high and the other 10 feet high, on level ground. Two straight strings join each top to the foot of the other bamboo. Find the height at which the strings cross. Let d>0 be the horizontal separation, x the distance from the taller bamboo and h the crossing height.

The descending string has h=15*(1-x/d), while the ascending string has h=10*x/d. Adding the ratios gives h/15+h/10=1, hence h=6 feet. The separation cancels; it affects the crossing's horizontal coordinate, not its height. The same argument gives h=a*b/(a+b) for positive heights a and b.

```
h/15+h/10=1
bound h=[0,10]
```

The short model is preferable when only h is needed. Introducing x and d would be appropriate if the horizontal location or separation were also part of the question, but an unspecified positive d leaves a family of geometric arrangements. The elimination identifies exactly why the requested height is still determined.

1 8 Recognise exact integer structure

Pell's equation x^2-D*y^2=N, general integer quadratics and weighted sums of squares have arithmetic structure that ordinary real root search does not exploit fully. For x^2-2*y^2=1 with positive x and y bounded by x<=1000, the pairs are (3,2), (17,12), (99,70) and (577,408). The recurrence (x,y) to (3x+4y,2x+3y) generates further positive pairs.

The recognised routes use exact integer arithmetic and preserve decimal strings that can contain hundreds or thousands of digits. Version 2.0.1 also accepts supported decimal/scientific literal bounds beyond floating-point range, such as bound x=[1e499,1e500]. This does not make every expression or coefficient arbitrary precision, and unsupported numerical models reject those oversized bounds.

A row cap can stop an exact enumeration before the entire bounded set is returned. Read the appropriate integer completion flag and reason rather than inferring completeness from exact values alone. Chapters 13 to 15 explain the distinctions through Pell recurrences, general conics and weighted representations.

1 9 Solve an initial-value problem as a function

```
ode
y'=-y
initial t=0
initial y=1
interval t=[0,2]
method=RK45
output points=21
```

The reference function is y=exp(-t). At t=0 it equals one, and differentiation gives y'=-y. At t=2 its value is approximately 0.1353352832. This independent solution checks the complete trajectory rather than merely whether the first row matches the supplied initial data.

The initial condition selects one member of a family. Without y(0)=1, every C*exp(-t) satisfies the differential equation. Numerical integration advances a chosen state through an interval. Its adaptive local error estimates and output interpolation are not proofs of global accuracy. Compare tolerances and use physical invariants or analytic solutions when available.

For higher-order equations supply initial derivatives too. For y''=-y, both y(0) and y'(0) are required. A fixed RK4 step is different from an adaptive output step: one controls internal integration and the other normally controls reporting. Chapters 16 and 17 explain these controls before introducing stiffness and events.

1 10 Treat singular points through compatibility

The equation x*y''+y'+x*y=0 has a vanishing leading coefficient at zero. Dividing by x creates an undefined explicit derivative there, yet its regular solution with y(0)=1 and y'(0)=0 exists. Its power series begins 1-x^2/4+x^4/64, and y''(0)=-1/2.

For the supported scalar linear class, the engine can use a checked regular series without asking the user to invent nearby initial values. The original initial row remains. At x=1, the regular value is approximately 0.7651976865579666. This is a specific supported continuation method, not a rule that every zero denominator can be replaced by an epsilon.

For x*y'=y with y(0)=0, every C*x fits. Selecting a nearby point would silently choose a free slope. A genuine divergent branch cannot be made finite by the same trick. Interior crossings require compatibility with the branch approaching the singularity. Chapter 18 develops determined, ambiguous, incompatible and unsupported cases separately.

1 11 Distinguish boundary data from initial data

A boundary-value problem can prescribe y at both ends rather than y and y' at one end. For y''=-2 on [0,1], y(0)=y(1)=0 gives y=x*(1-x). The slope at the first endpoint is found as part of the solution; it is not supplied as an initial condition.

The regular BVP solver uses adaptive collocation and local Newton iteration. A guess is a starting curve for that iteration, not another boundary condition. Nonlinear BVPs can have several branches, and changing a guess can select a different one. A converged curve meets numerical defect and boundary residual criteria; it is not automatically the unique mathematical solution.

The distinction matters for singularity support too. Automatic regular-series continuation is an IVP feature for a bounded class. It does not mean the regular BVP solver supports every singular endpoint. Chapter 19 explains admissible endpoint conditions, guesses and failure diagnostics with two of the user's Robin/logarithmic examples.

1 12 Move from a curve to a field

Laplace's equation u_xx+u_yy=0 describes a function on a two-dimensional region. Boundary values on a square determine the intended physical problem. A grid replaces the continuum calculation by finitely many coupled unknowns; the returned field approximates the continuous solution when the formulation and discretisation are appropriate.

At an interior node of an equally spaced square grid, the five-point discrete Laplace equation says that the value equals the average of its four axial neighbours. Iterative updates propagate boundary influence through the domain, so one update sweep is not generally the converged field. A small discrete residual checks the finite-difference system, while a grid-refinement study investigates discretisation error.

Heat and wave examples add time to one spatial coordinate. The Desktop heatmap uses x and y for a steady rectangular field or x and t for these transient fields. The section slider selects a spatial level or a time level and displays u(x). Chapters 20 and 21 explain boundaries, grid counts and the different roles of stability and accuracy.

1 13 Understand dependence through an integral

A second-kind Fredholm equation combines a local value with a weighted contribution from an entire interval. A Volterra equation integrates only up to the current coordinate. Both are functions-to-functions problems, although the numerical method produces a finite table after quadrature.

For y(x)=1+integral from 0 to x of y(s) ds, differentiation gives y'=y and evaluating the original integral at zero gives y(0)=1. Thus y=exp(x). This provides an independent check of a Volterra calculation. Differentiation is useful here because the data are smooth and the initial relation is retained; arbitrary integral equations should not be differentiated without preserving their conditions.

The current modes use finite ordinary kernels and finite intervals. They do not implement first-kind inversion, principal values or singular kernels. Chapter 22 uses manufactured Fredholm solutions and a forward Volterra calculation to show what quadrature residuals and refinement comparisons can establish.

1 14 Follow a reproducible Desktop routine

Define quantities and units, classify the problem, write the supported model and predict simple reference checks. Choose F4 for syntax checking, then Ctrl+Enter or F5 for Solve. Read Solutions and Run report together. Examine plots only after identifying the coordinates and the returned success status.

Save a .pdeq project for model-specific settings and bound overrides. Export full JSON for numeric strings and diagnostics, CSV for tabular exchange and the run report for readable evidence. Editing the model does not change the previous completed-run snapshot. Force-Stop terminates the worker without retrieving unfinished partial results; a retained table can therefore belong to an earlier run.

Commercial entitlement permits double-precision Desktop and CLI use. Extended precision and DLL computation have separate entitlement requirements. An eligible automatic trial can operate without license.txt for 14 days. These rules explain operation; deleting a licence file is not a valid test of whether a trial-enabled application is protected.

1 15 Use precision and pictures responsibly

Binary128 gives more arithmetic precision than double, not arbitrary precision and not automatically more reliable physics. Exact integer digits, approximate root digits, local integration estimates and discrete PDE residuals represent different evidence. The book retains these distinctions even when every result appears in the same Solutions tab.

Plots use double display arithmetic and connect sampled values. A smooth curve can hide poorly resolved behaviour. A colourful heatmap can display a discrete field whose mesh is too coarse. Conversely a plotting range problem need not destroy the full numeric strings available in exports. Always return to the model, residuals and convergence checks.

The previously reported long-value viewer limitation remains open in this review edition. A visible fragment is not evidence that the stored value was truncated. Preserve full numerical strings through copying and JSON exports, and verify critical long values in an independent plain-text or exact-integer workflow.

1 16 Practice problems

1. Explain why x+y=5 alone does not determine two real unknowns, and give two different pairs.

2. Find both ordered positive pairs satisfying x+y=5 and x*y=6. Explain how labels affect the count.

3. At x=1.001, compare coordinate error and residual for (x-1)^2=0.

4. Find the crossed-string height for upright heights 12 and 8 on level ground.

5. Verify that (17,12) satisfies x^2-2*y^2=1 exactly.

6. Determine y(1) for y'=-y and y(0)=2.

7. Find the missing coefficient of x^2 in the regular Bessel series with y(0)=1.

8. Verify y=x*(1-x) in its equation and endpoint conditions.

9. Explain why halving a wave spatial step without changing the time step can break stability.

10. Distinguish an empty Solutions array in algebraic JSON from an empty algebraic array in a successful trajectory result.

1 17 Worked answers

1. One independent linear condition leaves a line: (0,5), (1,4) and every (a,5-a) are examples. More numerical precision cannot add missing information.

2. Eliminate y=5-x to obtain x^2-5x+6=0, hence x=2 or 3. The ordered pairs are (2,3) and (3,2). Interchangeable physical objects can make them one unlabelled configuration.

3. Coordinate error is 0.001=1e-3, while residual is 1e-6. They are related by a square, not equality. Reporting six small residual places does not establish six coordinate places.

4. The height is 12*8/(12+8)=4.8. Positive separation cancels from the ratio calculation, but the ideal straight-string and level-ground assumptions remain.

5. 17^2-2*12^2=289-288=1. This exact substitution verifies the pair but says nothing by itself about every possible pair.

6. y=2*exp(-t), so y(1)=2/e, approximately 0.7357588823. Both derivative and initial value should be checked.

7. Set y=1+a*x^2+... . Then x*y''+y'+x*y=(4a+1)*x+...; therefore a=-1/4. The second derivative is 2a=-1/2.

8. Differentiating gives y'=1-2x and y''=-2. Both endpoint values vanish. At the midpoint y=1/4, which is a useful sample check.

9. The Courant ratio c*dt/dx doubles. A previously admissible ratio of 0.8 becomes 1.6; reducing dt proportionally maintains it. Stability is separate from continuum accuracy.

10. A trajectory result stores samples in trajectory and can leave solutions empty by design. Inspect schema, problem type, method-specific success and status before deciding whether data are missing.

1 18 Plan the route through the book

Chapters 2 to 4 establish the Desktop workflow, language and checking habits. Chapters 6 to 12 develop geometry, discrete models, polynomial and transcendental systems and applied balances. Chapters 13 to 15 concentrate on exact integer structure. Chapters 16 to 19 treat IVPs, events, stiffness, supported singular continuation and BVPs. Chapters 20 to 22 address fields and integral equations. Chapters 23 to 25 return to arithmetic reliability, integration and reproducible teaching/research.

Every worked model belongs to its stated assumptions. Screenshot placeholders specify what evidence to capture, and existing screenshots retain their role as illustrations of actual earlier runs. When the current interface or time differs, use your own capture and retain the returned numerical evidence. The goal is an explanation that remains useful when an example is adapted, rather than a collection of output pictures without a mathematical argument.

Chapter 2 Getting Started with Patodia Equation Solver

Chapter 1 explained how to turn a question into equations and how to distinguish an algebraic answer from a meaningful one. This chapter takes that reasoning to the desktop. You will install or open Patodia Equation Solver, enter small models, inspect alternative solutions and save enough information to repeat a calculation.

The Version 4.0 Desktop runs standalone models and workflow scripts. The workspace includes trajectory, convergence, field, eigenpair and parameter-study views. See Chapter 3 for workflow syntax and Chapter 43 for a complete study.

The examples below have deliberately simple reference answers. This lets you learn the application while checking its mathematics independently. The answers are derived here; they are not claimed as newly measured solver runs. No timing, discovery order or root count is promised for a particular search allowance.

2 1 Prepare the application and your working folder

Use a Windows 10 or Windows 11 x64 computer. The desktop distribution does not require you to install a compiler, Python or a spreadsheet application. Two distribution methods are described in the guide: a setup executable and a ZIP package. Choose the method supplied to you rather than combining files from different releases.

For the current portable Windows distribution, extract the complete ZIP and start PatodiaEquationSolver.exe. Keep the adjacent equation_solver.exe, runtime DLLs, examples and documentation together. A separately supplied installer can have its own filename and installation layout; do not assume an old setup filename identifies this edition.

With the ZIP package, extract the entire archive into a writable folder before opening the program. Do not run it from inside the compressed archive. Open PatodiaEquationSolver.exe, the desktop application. The similarly named equation_solver.exe is its numerical worker. It is neither the installer nor the desktop window that you ordinarily open.

Use the complete matching Windows 4.0 package for the current edition. Keep its worker, Desktop, runtime DLLs, examples and guides together. Older installer filenames and layouts are historical packaging details and do not establish which engine is running now.

Create a separate working folder, for example Documents\PatodiaBook\Chapter 02. Keep your models, projects and exported results there. This prevents a package update from becoming confused with your own experiments. Save an unchanged copy of each example before modifying it.

2 2 Check trial or licence status

The supplied guide describes an automatic full-featured 14-day trial for the current Windows account. It starts on first use when the application checks its licence, rather than on the day the installer is downloaded. An internet connection, trial licence file or fingerprint request is not required for that automatic trial.

Read the banner after opening the desktop. Help > Licence Information shows the active edition, scope, expiry and related details. Reinstalling or moving the application does not restart the trial. A previously installed licence retains its terms; an invalid or expired licence should be addressed through activation or support.

For an individual Commercial or Professional licence, the documented procedure uses Help > Machine Fingerprint. Send the copied identifier to the publisher and activate the signed file supplied in response through Help > Activate Licence. A departmental or campus academic licence uses the institutional file supplied by the administrator, without a machine fingerprint. Follow the agreement governing eligible users and machines.

Normal precision is available under the Commercial entitlement described in this guide. Professional, Trial, Academic and OEM entitlements support extended precision. Availability in the interface does not by itself establish redistribution or integration rights; those depend on the applicable agreement.

For the first exercises, use normal precision. You do not need extended precision to understand an exact answer such as x=3 and y=2. If an extended-mode request is rejected, inspect the active entitlement before treating the message as a mathematical or numerical failure.

Keep the computer date and time correct. The documented offline checks can reject clock rollback. Do not alter a signed licence file. Installation, licensing and equation syntax are separate issues: resolve a launch or entitlement problem before investigating a model's root count.

2 3 Recognize the desktop workspace

The equation editor holds the current model. Enter one equation, declaration or condition per line. Line numbers and syntax highlighting help you locate mistakes, but they do not establish that the mathematics is correct. The editor is where you change the question; the result tabs show what happened in a completed calculation.

The File menu provides opening, saving, import, export and printing. Edit and Search support ordinary text work. Compute contains Check Syntax, Variables and Bounds, Solver Settings, Precision, Number Style, Solve and Stop. Help provides quick instructions, licence information and diagnostics. The Examples menu loads editable models. Tools includes Equation Generator. Section 2.26 describes it and the Fun with Mathematics activities.

The result areas include Solutions, Run report, Diagnostics, ODE / BVP / integral plot, ODE events, PDE fields / animation, Convergence / eigenpairs / sweeps and Parameter sweep. Read scalar quadrature results as an integral and error estimate rather than as a trajectory. Their contents depend on the captured problem type.

A useful keyboard sequence is F4 to check syntax, followed by F5 or Ctrl+Enter to solve. Ctrl+S saves. Esc stops an active solve. The distinction between checking and solving matters: F4 can accept a syntactically valid model that has no real solution.

For example, x^2+1=0 is meaningful input. It has no real root because a real square cannot be negative. A successful syntax check says that the application can read the statement. It does not promise a solution or prove consistency.

Results are associated with the model used for their completed run. If you change the editor afterwards, the displayed values do not automatically become answers to your new equations. Read the completed model in Run report and solve again whenever an edit changes the question.

2 4 Complete your first solve

Choose File > New. Select US notation through Compute > Number Style so that the examples in this chapter use decimal points and comma-separated bound endpoints. Enter this model exactly:

```
x+y=5
x-y=1
```

Press F4. Check that the recognized unknowns are x and y. No initial estimates are needed. The two statements are simultaneous equations, not assignments executed from top to bottom.

Open Compute > Solver Settings. Begin with a two-second time budget, normal double precision and the ordinary default tolerance. For this first model, keep the other search settings at their defaults. Press F5 and let the run finish.

The reference answer is x=3, y=2. Adding the equations gives 2*x=6, so x=3. Substituting into x+y=5 gives y=2. Check both original relationships: 3+2=5 and 3-2=1.

Read each value under its variable heading. A solution is a named vector, so the order in which columns appear should never replace the variable names. If the application returns a nearby decimal such as 2.9999999999999996, inspect the full value and residual rather than assuming a different mathematical answer.

This model also provides a basic installation check. If it cannot run, review Diagnostics, the executable arrangement and the selected number style. There is little benefit in immediately trying a difficult benchmark when a two-equation example fails to launch or parse.

Save the model as ch02_first_solve.txt. Then save a project copy as ch02_first_solve.pdeq if you want to retain its settings. In the next examples you will preserve this same routine: enter, check, solve, verify and save.

2 5 Discover two nonlinear solutions

Replace the second equation with a product relationship and add explicit bounds:

```
x+y=5
x*y=6
bound x=[0,5]
bound y=[0,5]
```

Check syntax and solve. The reference vectors are (x,y)=(2,3) and (3,2). These are different assignments to the named variables, even though their values form the same unordered pair.

To derive them, substitute y=5-x into x*y=6. Then x*(5-x)=6, or x^2-5*x+6=0. Factoring gives (x-2)*(x-3)=0. Thus x=2 or x=3, and the sum equation supplies the corresponding y.

Substitution checks both vectors. For (2,3), the sum is 5 and the product is 6. The same checks succeed for (3,2). The bounds admit both because every coordinate lies between zero and five.

If one run finds only one of these vectors, first inspect Maximum solutions. A cap of one can stop a search after the first accepted candidate. Then check the bounds and increase the time allowance if necessary. A missing row is not evidence that the second algebraic root has disappeared.

Discovery order may change. Compare named coordinates rather than solution row numbers. Save this example as ch02_sum_product.txt and retain a separate project if you change its settings.

Here completeness follows from the quadratic reduction: every solution of the original system must give x=2 or x=3, and both resulting pairs have been checked. The finite numerical run is useful evidence of discovery. The algebra supplies the reason there are no other real solutions to this model.

2 6 Use bounds and conditions deliberately

A bound describes a finite inclusive interval. In bound x=[0,5], both endpoints are allowed. Bounds can represent physical restrictions or the region selected for an experiment. Record which interpretation applies, because a deliberately narrow search box can exclude mathematically valid roots.

Return to the two-solution model and add this condition:

```
x+y=5
x*y=6
bound x=[0,5]
bound y=[0,5]
x<=2.5
```

Only (2,3) remains feasible. The other pair still satisfies the equalities but violates x<=2.5. Removing that condition restores the original question; it does not improve or damage the equation solver.

For a second illustration, use a new file:

```
x^2+y^2=25
x-y=1
bound x=[0,5]
bound y=[0,5]
```

The unrestricted pairs are (4,3) and (-3,-4). The present bounds retain only (4,3). Substitution gives 16+9=25 and 4-3=1. A positive search box excludes the negative pair by design.

Compute > Variables and Bounds provides another way to enter bound overrides. For a reproducible book exercise, keep the bounds in the model text and check that no unintended override changes them. The completed run report is the record of what was actually used.

Do not write 0<=x<=5 as a chained comparison. Use a bound or two separate conditions. Bounds also do not supply initial guesses. They restrict admissible values while the program manages its own search initialization.

Save the constrained version under a new filename. Keeping both versions makes it possible to explain a changed solution count by pointing to the changed mathematical requirement.

2 7 Learn the input conventions you need now

Write explicit multiplication with *. The expression x*y is a product; xy is a single variable name. Names are case-sensitive, so x and X denote different unknowns. Use ordinary ASCII characters in input, rather than copying a typographic multiplication sign, superscript or long minus from formatted prose.

Powers may use ^ or **. Parentheses make the intended grouping clear. Write (x+1)^2 for the square of the complete sum. Do not expect x+1^2 to express the same operation. Functions require parentheses: sin(x), exp(x), sqrt(x) and ln(x).

Comments can explain a model without adding equations. The following example constructs a unique reference root from an exponential and a linear relation:

```
// A teaching example with reference root x=1 y=2
exp(x-1)=y/2
x+y=3
bound x=[0,3]
bound y=[0,3]
```

At (1,2), exp(0)=1 equals y/2, and the sum is 3. Substituting y=3-x gives exp(x-1)=(3-x)/2. On this interval the left side strictly increases while the right side strictly decreases, so they can meet at most once. The checked pair is therefore the unique solution in the stated box.

Trigonometric inputs use radians. The explicit constants #pi and #e are available. Both log and ln mean natural logarithm; log10 means base ten. Respect real domains: ln(x) needs x>0 and a denominator cannot be zero.

Chapter 3 develops declarations, functions and constraints in detail. At this stage, the essential habit is to use readable expressions and check the recognized variables before searching. An unexpected name often reveals a spelling or multiplication error immediately.

2 8 Understand settings before changing them

Use the complete matching Windows 4.0 package for the current edition. Keep its worker, Desktop, runtime DLLs, examples and guides together. Older installer filenames and layouts are historical packaging details and do not establish which engine is running now.

The time budget allows search effort. It is cooperative rather than a strict wall-clock deadline: launch, parsing, refinement and reporting add overhead. A zero budget removes that time limit; other stopping limits still apply. Use a small finite allowance while learning so that a difficult experiment returns control predictably.

Maximum solutions limits how many distinct accepted candidates are requested. Maximum starts and iterations per start limit different parts of the search. A run can end before its time allowance if another limit is reached. Consequently, increasing time alone does not guarantee that more solutions will appear.

The random seed controls randomized choices. For a controlled comparison, keep it unchanged while changing one other setting. A different seed can explore different candidates, but it is not a mathematical change to the equations or a completeness certificate.

Normal precision has a default residual tolerance of 1e-9. Extended mode normally uses 1e-24. A custom tolerance can survive a precision change, so check the actual field. Do not use an extremely small tolerance merely to make an answer appear more impressive.

The initial automatic radius, range growth and range stages govern automatic exploration for otherwise unrestricted variables. Leave these and advanced search aids at their defaults for the exercises. Meaningful model bounds are easier to interpret than undocumented changes to search controls.

2 9 Read and check the completed results

Each solution row represents one accepted vector. Max residual summarizes the largest reported equation residual or constraint violation at the chosen precision. A small value supports numerical agreement with the model, but does not establish the same number of correct coordinate digits.

For the sum-and-product example, define r1=x+y-5 and r2=x*y-6. At an exact reference pair both are zero. With exported approximate coordinates, calculate both residuals using all available digits. Rounding a coordinate before substitution can make an accurate result look worse.

Found (s) records within-run acceptance time. It is not application startup time. If three rows have times 0.04, 0.01 and 0.07 seconds, the earliest listed acceptance occurred at 0.01 seconds and the latest at 0.07 seconds. Adding these values does not give a useful total runtime because they are timestamps measured from a common start.

The worker may search after accepting the final reported root. The latest listed acceptance, configured budget and measured process duration are therefore different quantities. Compare the same measurement when recording experiments.

Double-click a row for full solution details. For a wide system, scroll horizontally and read all variable headings. Select a rectangular block and use Ctrl+Shift+C to copy selected result cells. Use Run report to confirm that the equations, precision and bounds belong to the calculation you intend to inspect.

Never infer the proprietary algorithm from a fast timing or a low residual. Neither observation identifies symbolic elimination, interval methods, GPU use or any other specific implementation. This book discusses documented behavior and checked mathematics; it does not reconstruct internal methods from output alone.

2 10 Save models projects and results separately

A text model and a project serve different purposes. A .txt or .eq file stores equation text. A .pdeq project retains equations, solver settings, number style and bound overrides. The project does not retain the results table. Save and export are therefore complementary actions.

For an important run, first save a project. Let the calculation complete, then use File > Export solutions to save CSV or JSON. Use File > Export run report to retain the completed text report. These files answer different questions: what model and settings were prepared, which values were returned, and what the completed run recorded.

A simple filename pattern helps. Use ch02_sum_product.pdeq for the model configuration, ch02_sum_product_results.json for values and ch02_sum_product_report.txt for the completed report. If you add x<=2.5, use a new name such as ch02_sum_product_filtered so that the two questions remain distinguishable.

JSON preserves long numerical values as decimal strings and includes the desktop_run model and settings snapshot. CSV is convenient for tabular inspection, but a spreadsheet may automatically round long values when importing them as ordinary floating-point numbers. Preserve strings or import numeric columns as text when full exported digits matter.

File > Print lets you choose the completed report or current equations. If an installed PDF printer is available, it can produce a printed PDF. Check which content you selected: an edited model and an older completed report need not describe the same question.

Save regularly because the guide does not describe automatic crash recovery. Before updating the application, preserve your projects and completed exports. A reproducible calculation consists of the question, configuration and evidence, not merely a screenshot of the visible result table.

2 11 Run a small controlled experiment

Now construct a model with six known real solutions:

```
(x+2)*(x-1)*(x-4)=0
y^2=9
bound x=[-3,5]
bound y=[-4,4]
```

The first equation gives x=-2, 1 or 4. The second gives y=-3 or 3. Since the two equations are independent, every x choice combines with both y choices. The complete reference set is (-2,-3), (-2,3), (1,-3), (1,3), (4,-3) and (4,3).

Use normal precision and a maximum solution setting of at least six. Record the actual tolerance and seed. Run with a two-second budget and export the completed result. Repeat with five seconds, changing only the time budget. Compare the named vectors against the reference set, irrespective of their row order.

A longer run may find the same set. That is a useful observation, not a failure: once all six mathematical roots are present, there is no seventh distinct root to discover. If a root is absent, check limits and domains before drawing a performance conclusion.

Next add y>=0 and solve again. The feasible reference set now has three members: (-2,3), (1,3) and (4,3). Explain the reduction through the new condition rather than through elapsed time. Save this model separately so the comparison remains auditable.

A record should include filename, application version, precision, tolerance, seed, bounds, time allowance, maximum solutions, returned vectors and reported residuals. For timing comparisons also record the computer and the definition of elapsed time. These details turn an informal demonstration into an experiment that someone else can meaningfully repeat.

2 12 Preview integer and transcendental models

An integer declaration changes which values are allowed. Consider the following model:

```
int n
n+y=4.5
bound n=[0,2]
bound y=[0,5]
```

Here n may be 0, 1 or 2, while y remains real. The reference vectors are (n,y)=(0,4.5), (1,3.5) and (2,2.5). Without the integer declaration, the sum equation describes a continuous family in the box. Do not replace integer solving by rounding arbitrary continuous answers; rounding can destroy the equality.

For a transcendental example, enter:

```
sin(t)=0
bound t=[-4,4]
```

The reference roots are -pi, 0 and pi. In solver notation, #pi represents pi when used in expressions. The domain contains these three integer multiples of pi and no others, since 2*pi is greater than four. Without finite bounds, the equation has infinitely many real roots and cannot be exhausted by a finite list.

For example, expm1(x) evaluates exp(x)-1 and log1p(x) evaluates ln(1+x), with numerical advantages near zero. Their purpose is more careful evaluation of small changes, not arbitrary precision or a guarantee that every model using them is well conditioned. Chapter 23 returns to this distinction.

These small exercises introduce different model types without making the first desktop session depend on difficult numerical reference data. Check the mathematics first, then use the program to explore it.

2 13 Recover from common problems

If the application or worker does not start, verify that the complete package was installed or extracted. Keep companion files from the same release together. If a trial or licence message blocks calculation, inspect Help > Licence Information. Neither changing equations nor increasing search time repairs missing runtime files or an expired entitlement.

For a syntax error, examine the reported line, variable spelling, parentheses, explicit multiplication and number style. Do not paste labels such as f1: or explanatory prose from a conversation into the editor unless the documented grammar supports them. Paste only equations, declarations and valid comments.

US style uses decimal points and commas between function arguments or bound endpoints. EU style uses decimal commas and semicolons in those positions. The chapter examples use US style. Do not write thousands grouping: use 1000 or 1e3. A text file does not record its intended style; a project does.

If no roots appear, check the mathematical domain and every constraint. A logarithm may be undefined where the search is attempted. A bound may exclude a known root. A solution cap or other effort limit may stop discovery early. First retest a simple model that has a known answer, then isolate the issue in the harder one.

Stop or Esc terminates the numerical worker. In the documented version, partial roots from the interrupted run are not returned because final results arrive only at completion. Earlier completed results can remain visible and are labelled as previous. Prefer a finite budget and allow completion when you need the new run's results.

Before requesting support, preserve the model and settings, read Diagnostics and review Help > Copy Diagnostics for private details. Include the version, precision, style and relevant bounds. A precise reproducible example is more informative than saying only that the solver did not work.

2 14 Practice exercises

Exercise 1. Enter x+y=9 and x-y=3. State the reference solution, check both equations and describe the desktop actions needed to obtain it.

Exercise 2. Change the sum-and-product model to x+y=7 and x*y=12, with each variable bounded between zero and seven. Derive all real solutions before running it.

Exercise 3. Add a condition to Exercise 2 so that only the vector with the smaller x remains. Save the filtered question separately.

Exercise 4. Use (x+1)*(x-2)*(x-5)=0 and y^2=4, with bounds that admit every root. List the complete reference vectors. Explain why a maximum solution setting of four is inadequate for enumerating the reference set.

Exercise 5. A model passes F4 but contains sqrt(x)=-2. Explain why syntax success does not imply a real solution.

Exercise 6. A completed table shows x=3 and y=2. You edit the model to x+y=9 and x-y=3. Explain what must happen before you use the table as evidence for the edited equations.

Exercise 7. You reopen a saved .pdeq file and the previous result table is absent. Explain what the project stores and which exports should have been made.

Exercise 8. Three accepted solutions have Found times 0.02, 0.09 and 0.05 seconds. Identify the earliest and latest acceptance times and explain why their sum is not process duration.

Exercise 9. Form a mixed-integer model with integer n, real y, n+y=5.25 and 1<=n<=3. Choose a suitable bound for y and list its feasible vectors.

Exercise 10. Restrict sin(t)=0 to [-7,7]. List the mathematical roots and state whether a longer search alone proves that all roots have been returned.

Exercise 11. Explain why a residual of 1e-24 does not automatically establish 24 correct decimal digits in every coordinate.

Exercise 12. Plan a two-run comparison that changes the time budget alone. State which records must remain unchanged and which files you will preserve.

2 15 Answers and discussion

Answer 1. Adding the equations gives 2*x=12, hence x=6 and y=3. Both 6+3=9 and 6-3=3 hold. Enter the model in a new editor, check syntax with F4, verify settings, solve with F5 and inspect the completed values and residual.

Answer 2. Substitute y=7-x to obtain x^2-7*x+12=0, which factors as (x-3)*(x-4)=0. The vectors are (3,4) and (4,3). Both satisfy the bounds and both original equations.

Answer 3. Add x<=3.5. Only (3,4) is feasible. The other pair remains an algebraic solution but violates this condition.

Answer 4. The vectors are (-1,-2), (-1,2), (2,-2), (2,2), (5,-2) and (5,2). There are six independent combinations. A cap of four permits stopping before all six have been returned.

Answer 5. The real square root is nonnegative, so it cannot equal -2. F4 checks the input grammar and recognized model, not mathematical existence.

Answer 6. Solve again. The visible table belongs to its completed run until a new calculation finishes. The new reference vector is (6,3).

Answer 7. The project stores equations, settings, style and bound overrides, rather than the results table. Export CSV or JSON for solutions and a run report for the completed record.

Answer 8. The earliest listed acceptance is 0.02 seconds and the latest is 0.09 seconds. They share a common time origin. Process duration also includes work outside those acceptance events.

Answer 9. Use int n, n+y=5.25, bound n=[1,3] and bound y=[0,5], on separate lines. The vectors are (1,4.25), (2,3.25) and (3,2.25).

Answer 10. The roots are -2*pi, -pi, 0, pi and 2*pi. The next multiples lie outside the interval. Completeness comes from the known zeros of sine and the domain, rather than from the length of an unsuccessful additional search.

Answer 11. Residual measures agreement with equations. Coordinate error also depends on conditioning and multiplicity. Arithmetic precision and input-data accuracy impose further limits.

Answer 12. Keep equations, bounds, overrides, precision, tolerance, seed and effort limits unchanged except for time. Preserve each project configuration, completed solution export and run report, together with version and timing definitions.

2 16 From a first run to a reusable method

You now have a practical desktop procedure. Prepare an explicit model, confirm the syntax and recognized variables, inspect settings, let a finite-budget run complete, check every returned vector and preserve the completed record. Each step answers a separate question: what was asked, what was searched, what was found and why the answer is credible.

Use the supplied examples as editable starting points. File > Open replaces the current model after handling unsaved edits; File > Import equations inserts text at the cursor. Choose deliberately, because importing a complete second model can unintentionally combine its equations with the first.

Keep initial experiments small enough to verify independently. Confidence comes from a correct formulation and a checked result, not from the number of digits or the speed of a single run. A simple model with a known complete answer is particularly useful for learning an unfamiliar setting.

Chapter 3 develops the language of equations and constraints. It explains how to express constants, integer decisions, functions and admissible domains clearly. The desktop habits established here remain the same as the mathematical models become larger.

Use the complete matching Windows 4.0 package for the current edition. Keep its worker, Desktop, runtime DLLs, examples and guides together. Older installer filenames and layouts are historical packaging details and do not establish which engine is running now.

2 17 Current edition workflow and scope

Understanding the Desktop window

2 18 Editor and adjustable panes

The upper pane is the equations editor. It shows line numbers, syntax colouring, the current line and the caret position. A horizontal divider separates it from the lower results pane. Drag the divider to allocate more room to the model or results. The editor does not visually wrap long lines; scroll horizontally to inspect them.

Above each pane are independent font controls. The licence banner is at the top. The status bar reports the current action, editor line and column, number style and selected precision. An asterisk in the window title indicates unsaved model or project changes.

2 19 Result tabs and problem types

Tab	What it contains	When to use it
Solutions	Root rows or numerical samples	Read values and select cells for copying
Run report	Model used, method, status and diagnostics of the completed calculation	Check the stopping reason and numerical evidence
Diagnostics	Worker task, exit code, stderr and raw output	Investigate errors or prepare a support report
ODE / BVP / integral plot	A curve with horizontal and vertical selectors	Inspect a trajectory, integral curve or phase portrait
ODE events	Located event names, coordinates, states and terminal flags	Review event stopping or crossings

Tab	What it contains	When to use it
PDE fields / animation	Numerical, exact or error maps; contours, sections and playback	Inspect steady or transient fields
Convergence / eigenpairs / sweeps	Internal grids, errors and mode data	Assess convergence and references
Parameter sweep	Per-case summaries and overlays	Compare repeated model solves

Tabs remain present even when a model does not produce data for them. An algebraic calculation does not create a trajectory plot. A PDE uses the PDE tab rather than the ordinary trajectory plot.

2 20 Completed results and current edits

Results belong to the last returned run snapshot. Editing equations, changing settings or converting number style does not recalculate the displayed result. Check the completion time and MODEL USED in Run report. Solve again when you want the table to reflect your current editor.

The application retains the prior completed result when a worker is force-stopped. Its message identifies that the table is from the PREVIOUS completed run. Never attribute those rows to the interrupted model merely because the editor now shows it.

Number style precision and fonts

2 21 US and EU number styles

Choose Compute > Number Style > US - decimal point or EU - decimal comma. Changing style converts the current model notation. It does not recompute existing rows or reformat the completed result snapshot into a new run style.

Construct	US style	EU style
Decimal value	1.5	1,5
Two-argument function	atan2(y,x)	atan2(y;x)
Bound interval	bound x=[1.0,2.0]	bound x=[1,0;2,0]
Comment	// example	// example

Do not use thousands separators. For clarity, especially in EU style, use one integer variable declaration per line. Imported text is not automatically interpreted as a different number style. Check it after importing. JSON exports always use decimal points; EU CSV uses semicolons as field separators.

2 22 Arithmetic precision

Compute > Precision selects Double precision or Binary128 extended precision. The same extended option is available in Solver Settings. Double is the normal starting choice. Extended precision can reduce roundoff and improve refinement on suitable problems, but needs an eligible entitlement and does not prove accuracy or completeness.

For algebraic searches, binary128 normally refines candidates discovered using double arithmetic. ODE, BVP and field modes use the selected arithmetic for their numerical work. Plots use double precision, even when tables and

exports retain longer decimal strings. Exact recognised integer routes use exact integer arithmetic; large integer output does not require extended floating-point entitlement.

Switching precision adjusts the standard algebraic residual tolerance between 1e-9 and 1e-24 when it still has one of those standard values. A custom tolerance is retained. Differential-model rtol, atol and mesh directives remain in the editor and must be chosen separately.

2 23 Independent font controls

Use Equation font and Size (pt) above the editor to change model text. Use Solution font and its Size (pt) above the lower pane to change solution presentation. Sizes from 8 to 48 points are supported. Choose a readable monospaced font for dense numeric text. Changes apply immediately and preferences are saved.

View > Reset equation and solution fonts restores the application defaults. The editor's default is a monospaced 11-point font. Increasing the solution font can widen columns and increase row heights. Long-value width caps remain; a larger font is not a remedy for the reported wrapped-viewer defect.

Running checking and stopping

2 24 Check Syntax and Solve

Press F4 to check syntax. Read an error message and Diagnostics if checking fails; the editor can highlight reported error lines where available. Fix the model before solving. For numerical models a check validates required syntax, names and settings, not values sampled during integration or mesh solution.

Press Ctrl+Enter or F5 to solve. The Desktop starts a separate hidden worker and reports elapsed time. While a worker task is active, actions such as starting another solve, changing solver settings, opening a different model or activating a licence are disabled. Some editing remains possible; the active worker still uses the model captured at its start.

Read the returned stopping reason before assessing success. Algebraic rows are candidates accepted for the entered equations. Numerical trajectories have mode-specific success checks. A BVP or PDE failure is not turned into a successful curve merely because an intermediate numerical calculation existed.

2 25 Stop and time limits

Compute > Stop or Esc force-terminates an active solve worker. A force-stopped run does not return its partial roots or trajectory. Previous completed results can remain, clearly labelled. If stopping fails, wait for the worker allowance or close carefully when prompted.

Use a finite time budget or method-specific step/mesh limit when you want a graceful engine return. An IVP that reaches an internal numerical limit may return earlier finite samples; a forced Desktop Stop cannot retrieve them. Failed BVP and field requests return no converged trajectory. These are different outcomes.

Compute > Clear Results clears displayed result state without deleting your equations. Compute > View Run Log selects Diagnostics. These commands do not rerun the model. On exit during an active task, the Desktop asks whether to stop the task and close.

2 26 Equation Generator and Fun with Mathematics

Generate a reproducible nonlinear system

Open Tools → Equation Generator. Choose 1–20 variables, select the function families and complexity, and add bounds if needed. Save the seed and settings with the generated model so that you can recreate the exercise. Reveal the constructed solution on request.

The generator simplifies its equations using exact arithmetic. It combines like terms and powers, factors out common terms when that shortens the expression, and removes redundant parentheses.

Teaching exercise. Give students the generated model and any bounds. Ask them to solve it and check the residuals, then reveal the constructed solution and substitute it into the equations. Compare it with the returned roots and discuss why one constructed solution does not establish uniqueness.

Choose a mathematical activity

Find Your Number in Pi searches π for a chosen sequence of digits. The King's Puzzle asks for the smallest starting number that satisfies successive divisibility conditions. Number Puzzle asks you to arrange number tiles so that they satisfy the displayed arithmetic rules.

Chapter 3 Writing Equations and Constraints

The desktop procedure in Chapter 2 begins with a model in the editor. This chapter explains how to write that model so that it represents the intended mathematical question. You will distinguish unknowns from fixed data, group arithmetic correctly, use supported functions, define admissible domains and express integer decisions.

The central principle is simple: every accepted solution must satisfy the equations and the stated conditions together. Clear syntax makes a model readable, while sound mathematical formulation makes it meaningful. A successful syntax check confirms that the program can read the input; it does not confirm that the input describes your intention.

The syntax and desktop behavior described here follow the supplied Desktop User Guide with the October 2026 modulo update. All runnable examples use US number style unless explicitly identified as EU style. The reference answers are established by derivation or substitution, rather than presented as newly measured software runs. Detailed assessment of numerical accuracy follows in Chapter 4.

3 1 Begin with the question and define the unknowns

Before entering equations, write a short description of what must be found. Identify the unknown quantities, fixed data, units and admissibility conditions. This preparation is especially useful when a problem is stated in ordinary language: a solver cannot decide whether a quantity represents a coordinate, a nonnegative length or a count of indivisible objects.

Suppose a rectangular panel has perimeter 28 metres and area 48 square metres. Let width and height denote its side lengths in metres. The perimeter gives 2*(width+height)=28 and the area gives width*height=48. Both sides must be positive. If orientation matters, state which side is called width; otherwise two assignments may describe the same shape after rotation.

A complete input is:

```
// Rectangle dimensions in metres
2*(width+height)=28
width*height=48
bound width=[1,14]
bound height=[1,14]
```

The mathematical solutions are (width,height)=(6,8) and (8,6). Dividing the perimeter equation by two gives width+height=14. Substituting height=14-width into the area relationship gives width^2-14*width+48=0, or (width-6)*(width-8)=0.

The lower bound of one is a stated restriction for this exercise. It admits both solutions, but it is not a general translation of positivity: a different panel might have a side between zero and one metre. Never copy a convenient numerical lower bound into another application without checking its meaning.

Write comments that preserve the assumptions, not just the variable names. A useful note says that all lengths use metres and that the panel is rectangular. This gives a later reader the information needed to judge whether the equations remain appropriate when the data change.

3 2 Treat equations as simultaneous relationships

In an equation file, the equal sign expresses a requirement. It is not an assignment operator that updates memory in a sequence of instructions. The equation x+y=7 has the same role whether it appears above or below x*y=10. Both must hold for the same values of x and y.

For example:

```
x+y=7
x*y=10
bound x=[0,7]
bound y=[0,7]
```

The reference pairs are (2,5) and (5,2). Reversing the two equation lines preserves the mathematical solution set. Numerical discovery order and timing can still differ between runs; mathematical equivalence does not require identical search histories.

The guide also permits a bare expression, which is interpreted as equal to zero. Thus x^2-9 is shorthand for x^2-9=0. For teaching and review, an explicit equality is usually clearer. It avoids making readers wonder whether a line is incomplete.

One equation per line makes errors easier to locate. Top-level semicolons can separate statements, but the line-based form is preferable when learning and when sharing examples between number styles. Use blank lines and comments to separate conceptual groups without changing their simultaneous meaning.

An equation does not automatically introduce a new independent piece of information. The pair x+y=7 and 2*x+2*y=14 is dependent and describes an entire line. Replacing the second right-hand side by 15 makes the pair inconsistent. Counting lines is therefore not enough to establish that a system has a unique solution.

When building a larger model, ask what requirement each equation adds. A material balance, geometric relation and constitutive relationship may each be essential. Repeating a relationship in a different algebraic form does not replace a missing physical law.

3 3 Choose variable names and declarations carefully

Variable names are case-sensitive. Use ASCII letters, digits and underscores, beginning with a letter or underscore. The names pressure, pressure_in and x1 are readable examples. The names x and X refer to different unknowns, so an accidental capital letter can change the model.

Explicit multiplication prevents another common ambiguity. The text xy denotes one variable name; it does not mean x multiplied by y. Likewise, flowrate is a single name. Write x*y and flow*rate when two unknowns are intended.

The book's variable-order declaration can make a report easier to read:

```
var width,height
width+height=14
width*height=48
bound width=[1,14]
bound height=[1,14]
```

Here var width,height specifies the declared variable order; it is not a list of initial guesses. The equations still determine the values. Most examples in this chapter rely on recognized variable names directly, so they can be read without a separate ordering declaration.

Use a declaration only when it contributes meaning or clarity. Integer and binary declarations change admissible values. A fixed constant declaration supplies known data. A variable-order declaration organizes names. These are different purposes and should not be confused.

Declared integer or binary variables and named bounds must refer to variables used in the model. A bound alone does not introduce an independent free search variable. If you intend to solve for an unknown, connect it to the actual relationships rather than merely listing it with an interval.

After pressing F4, review the recognized variable names. If you expected x and y but see xy, inspect multiplication. If a stray name appears, inspect spelling and case. This quick check often finds a modelling mistake before time is spent searching an unintended system.

3 4 Use arithmetic and parentheses explicitly

The supported arithmetic operators include +, -, *, /, %, ^ and **. Both power operators express exponentiation. Although implicit multiplication is supported in some forms, explicit * is the clearest choice for a book example, a classroom exercise or an exchanged model.

Parentheses determine which quantities belong together. Compare (a+b)/2 with a+b/2. The first is the average of a and b; the second adds a to half of b. At a=6 and b=2, they evaluate to four and seven respectively. Both are valid expressions, so syntax checking cannot detect this difference in intention.

Similarly, 1/(x+1) differs from 1/x+1. At x=1 the first is one half and the second is two. When transcribing a fraction, put parentheses around its complete numerator and denominator whenever grouping could be misunderstood.

Powers associate to the right: x^2^3 means x^(2^3). If you intend the cube of x squared, write (x^2)^3. Those expressions are x^8 and x^6 respectively. Avoid relying on a reader's assumptions about repeated exponent notation.

Make the negative sign explicit too. Use (-x)^2 for the square of a negative quantity and -(x^2) for the negative of a square. At x=3 these are nine and minus nine. Parentheses are particularly valuable when copying expressions between software systems with different conventions.

Retain simple fractions such as 8/3 rather than shortening them prematurely to 2.6667. The fraction is evaluated numerically in the chosen arithmetic; it does not turn the solver into an exact symbolic rational system. It does, however, avoid inserting a deliberately rounded coefficient into the model.

Use scientific notation for small or large values, such as 1e-6 or 2.5e4. Do not insert thousands separators. In US style, write 1000 rather than 1,000, because punctuation also has grammatical meaning in the input language.

3 5 Use the percent operator for congruences

A congruence specifies an integer remainder. Write x % 5 = 1 for x congruent to 1 modulo 5. The operator returns a value from zero through one less than the positive modulus. In particular, (-4) % 3 = 2. It does not calculate a percentage and does not use the negative remainder convention of C++.

Declare every variable in a model containing % as integer. The modulus must be a constant positive integer between 1 and 1000000000. The numerator may use integer constants, integer variables, addition, subtraction, multiplication and nonnegative integer powers up to 64. Division and transcendental functions are not allowed inside the numerator. Constant integer expressions may be used for the modulus.

Multiplication, division and % have equal precedence and associate from left to right. Write parentheses around a combined numerator: (x+y) % 9 = 5. Without those parentheses, x+y % 9 means x+(y % 9). Power operators bind more tightly: x^2 % 7 takes the remainder after squaring x.

```
pos int x, y
x % 5 = 1
y % 6 = 2
(y - x) % 3 = 2
(x + y) % 9 = 5
(x*y) % 7 = 6
```

The vector x=6, y=8 satisfies all five equations: the checked remainders are 1, 2, 2, 5 and 6. The difference y-x is 2; the sum 14 has remainder 5 modulo 9; and the product 48 has remainder 6 modulo 7.

For a pure system of polynomial congruences, the solver can enumerate one fundamental period when explicit bounds are absent. This example has periods 315 for x and 126 for y and returns 18 representatives in the positive

period rectangle. Each pair represents further solutions obtained by adding multiples of the reported periods. A solution limit or time limit can stop this enumeration early; inspect the run report.

To request individual solutions within a chosen range, add explicit bounds for every variable. General expressions containing %, such as 2*(x % 5)=2, also require explicit bounds unless they form a supported pure congruence system. Automatic period and candidate-domain limits apply; if the engine asks for narrower bounds, provide them rather than treating the message as mathematical infeasibility.

A quotient-variable translation is still valid, but each congruence needs its own auxiliary integer: x-5*a=1, y-6*b=2, y-x-3*c=2, x+y-9*d=5 and x*y-7*e=6. These auxiliary variables must allow zero and negative values. At x=6, y=8 their values are a=1, b=1, c=0, d=1 and e=6.

3 6 Separate fixed constants from unknown quantities

A fixed parameter belongs in a constant declaration when its value is known and is not being solved for. This distinction makes a model easier to update and reduces uncertainty about the role of each symbol.

Consider a rectangular design written with named data:

```
constant perimeter=28
constant area=48
2*(width+height)=perimeter
width*height=area
bound width=[1,14]
bound height=[1,14]
```

The unknowns are width and height. Perimeter and area are fixed inputs. The two reference pairs remain (6,8) and (8,6). Changing a parameter changes the data for the model; it does not add a new degree of freedom.

By contrast, perimeter=28 without the word constant introduces an equation involving an unknown named perimeter. The equality fixes that unknown numerically, but it is still part of the equation system. This can be mathematically valid while obscuring the intended distinction between data and quantities to discover.

Constant expressions may refer to other declared constants. For example, constant diameter=10 followed by constant radius=diameter/2 records a known radius of five. Circular definitions are invalid, and a declared constant cannot depend on an unknown whose value is to be found.

Do not use a constant declaration for a relationship that is genuinely part of the solution. If radius is unknown and must satisfy an area equation, it belongs in that equation. Declaring it constant would replace the question with fixed data.

For physical calculations, attach units in a comment or accompanying prose. A constant declaration does not tell the software that 28 represents metres. Unit consistency remains your responsibility. Naming a parameter pressure_kPa can help readers, but the name does not trigger automatic unit conversion.

3 7 Use elementary functions with the correct meaning

Functions require parentheses around their arguments. The guide supports trigonometric functions sin, cos and tan; inverse functions asin, acos, atan and atan2; hyperbolic sinh, cosh and tanh; exponential and logarithmic functions; and several root, magnitude and rounding functions.

Trigonometric angles are in radians. A known angle of 30 degrees can be written as 30*#pi/180. For example, sin(30*#pi/180) equals one half. Typing sin(30) instead asks for the sine of 30 radians.

Both log(x) and ln(x) mean the natural logarithm. Use log10(x) for base ten. The explicit built-in constants #pi and #e represent their mathematical values. Use these forms instead of assuming a plain name pi or e will be interpreted as a built-in constant.

A small model demonstrates the difference between logarithm bases:

```
ln(x)=2
log10(y)=2
bound x=[1,10]
bound y=[1,200]
```

The unique reference vector is x=exp(2), approximately 7.389056099, and y=100. Both coordinates are positive and lie in the stated box. Replacing ln(x) with log10(x) would change x to 100 and would make the current bound for x exclude that answer.

The functions sqrt(x) and cbrt(x) represent square and cube roots. abs(x) is magnitude. hypot(a,b) represents a two-component Euclidean magnitude. The two-argument forms min(a,b), max(a,b) and pow(a,b) are also supported in US style.

In physical models, arguments of logarithms, exponentials and trigonometric functions should be dimensionless. Write ln(pressure/reference_pressure) when that ratio is the intended dimensionless quantity. Taking ln of a pressure number without a reference makes the answer depend on an unstated choice of unit.

3 8 Build valid domains into the formulation

A real expression is defined only on its real domain. sqrt(x) requires x>=0. ln(x) requires x>0. A denominator must not vanish. Inverse sine and inverse cosine require arguments between minus one and one. Tangent has poles and is not defined at every real input.

The input language supports inclusive <= and >= conditions, but not strict <, strict >, != or logical combinations. This makes it important to distinguish a mathematical domain statement in prose from a supported line of input. Do not type x!=0 and expect it to exclude a denominator singularity.

Consider:

```
ln(x)=0
bound x=[0.1,10]
```

The reference root is x=1. The positive lower bound keeps the entire search interval inside the logarithm's domain. For this particular example, one can independently establish that the only root is one, so the interval does not lose another root.

However, x>=1e-6 is not a universal encoding of x>0. It excludes every positive value below 1e-6 and can remove a valid root from another problem. Use a lower bound justified by the model or explicitly describe the restricted domain you are exploring.

A substitution can enforce a domain structurally. Writing x=exp(t) makes x positive for every finite real t, but changes the numerical formulation. A finite interval for t also limits the possible x values. Document that transformation and translate the final answer back to the original quantities.

For a reciprocal model, a physically justified positive interval such as bound x=[1,10] avoids zero directly. If positive and negative branches are both relevant, solve appropriate separate boxes and record the omitted neighbourhood. No unsupported logical operator is needed to investigate the branches separately.

3 9 Distinguish hard bounds from inequality conditions

A model-text bound has the form bound x=[lower,upper]. Both endpoints are finite and inclusive, and the lower endpoint must not exceed the upper. Equal endpoints fix a variable, as in bound x=[2,2]. Avoid duplicate bounds for the same variable in the model text.

An inequality condition expresses a relationship to be checked. It may involve several unknowns, such as x+y<=6, rather than only one variable. Conditions are checked using the residual tolerance; explicit bounds define the hard

search interval. Near a condition boundary, inspect numerical violations rather than assuming exact symbolic enforcement.

The following model gives a simple filtering experiment:

```
(x+2)*(x-1)*(x-4)=0
bound x=[-3,5]
x>=0
x<=3
```

The polynomial roots in the box are -2, 1 and 4. Only x=1 satisfies both additional conditions. Removing either condition changes which roots are admissible. Removing both restores all three reference roots.

Do not type a chained comparison such as 0<=x<=3. Write x>=0 and x<=3 on separate lines, or use bound x=[0,3] when the interval itself is the intended restriction. The bound form is often simpler for a single-variable domain.

Without explicit bounds, the documented automatic search explores increasing scales governed by radius, growth and stage settings. This is not a statement that the entire real line has been exhausted. A finite range or search allowance always has an interpretation that should be retained with the results.

Distinguish a known physical limit from a convenient demonstration box. A nonnegative concentration and an experimental interval selected to shorten runtime may both be useful, but they supply different kinds of information about the original problem.

3 10 Check desktop overrides before interpreting a model

Use the complete matching Windows 4.0 package for the current edition. Keep its worker, Desktop, runtime DLLs, examples and guides together. Older installer filenames and layouts are historical packaging details and do not establish which engine is running now.

Suppose the editor contains:

```
x^2=4
bound x=[-3,3]
```

The model-text interval admits x=-2 and x=2. If the desktop override sets x between zero and three, only the positive root is admissible for that run. Looking only at the editor would miss the reason for the changed question.

To restore the model interval, clear both override entries for that variable. Leaving both fields blank uses the text bound or automatic search, as applicable. When entering an override, supply both lower and upper values. Integer types appear in this dialog but are declared in the editor.

A .pdeq project saves bound overrides along with equations and settings. A plain-text equation export does not preserve that separate table. Consequently, two people can exchange identical text and still run different domains if one has additional overrides active.

For the worked examples in this chapter, use the displayed model-text bounds and clear unintended overrides. Read the completed run report to confirm the actual configuration. If a result count seems surprising, check the effective domain before increasing the search allowance.

This is a modelling issue as much as an interface detail. A solution is always a solution to a particular set of requirements. Keeping all active requirements visible or recorded is necessary for explaining why one run accepts a point and another rejects it.

3 11 Express coupled inequalities without inventing logic

Some admissibility conditions couple several variables. A budget, capacity or safety relation can restrict a sum or product while leaving the original equalities unchanged. Use supported <= or >= statements directly rather than introducing a separate optimization objective.

For example:

```
x^2=4
y^2=9
bound x=[-3,3]
bound y=[-4,4]
x+y<=0
```

Before filtering, the four reference pairs are (-2,-3), (-2,3), (2,-3) and (2,3). Their sums are -5, 1, -1 and 5. The condition retains (-2,-3) and (2,-3). Checking every complete vector makes the filtering transparent.

A requirement such as either x<=-1 or x>=1 needs special treatment because logical combinations are not supported. One practical approach is to run the two restricted subproblems separately and combine their checked results, accounting for any overlap. Do not silently replace the disjunction with both conditions: demanding x<=-1 and x>=1 simultaneously would make it impossible.

Similarly, strict positivity cannot be expressed by typing x>0. If a justified positive minimum is available, use it. Otherwise consider a documented domain transformation or a stated restricted search. The formulation should not claim exact equivalence when a numerical cutoff has changed the admissible set.

The solver is an equation and feasibility solver. It does not accept minimize or maximize objectives. If several feasible vectors remain, report them and interpret them. Selecting the cheapest design or greatest yield is a separate optimization question, for which Patodia Optimizer is a different product.

3 12 Declare integer and binary decisions

Variables are real-valued unless their type is restricted. Use int n or integer n for an integer unknown. Write each integer declaration on a separate line, especially when sharing a model between number styles. The signed forms pos int, neg int, nonneg int and nonpos int restrict integer domains further.

For a purchase example, suppose two package types cost four and seven units of currency, and the total paid is 39. Let small and large count the packages. Require nonnegative integer counts:

```
int small
int large
4*small+7*large=39
bound small=[0,9]
bound large=[0,5]
```

The feasible vectors are (small,large)=(8,1) and (1,5). To verify completeness, reduce the equation modulo four. Since 7*large has remainder 3*large and 39 has remainder three, large must be congruent to one modulo four. In the allowed interval it is one or five. The first gives small=8; the second gives small=1. Both satisfy the cost equation. This direct enumeration shows why checking a single obvious answer is insufficient.

A binary declaration restricts a variable to zero or one:

```
binary selected
selected+reserve=2
bound reserve=[0,2]
```

The reference vectors are (selected,reserve)=(0,2) and (1,1). Reserve remains real. A binary decision can represent a choice, but the declaration alone does not add business meaning; explain what zero and one signify.

Integer solutions must satisfy the original relationships. Solving a continuous relaxation and rounding its coordinates is not generally equivalent. A rounded count may break a total, capacity or nonlinear equality. Chapter 7 develops these models in more depth.

3 13 Introduce auxiliary variables when they clarify the model

An auxiliary unknown represents an intermediate quantity that helps express relationships. It can make a long model easier to audit, especially when the same nonlinear expression is used several times. It must be connected by equations rather than treated as a procedural assignment.

For a right triangle with fixed legs three and four, write:

```
constant leg_a=3
constant leg_b=4
hypotenuse^2=leg_a^2+leg_b^2
bound hypotenuse=[0,10]
```

The nonnegative root is five. If a squared intermediate quantity is useful elsewhere, an equivalent formulation is:

```
constant leg_a=3
constant leg_b=4
squared_length=leg_a^2+leg_b^2
hypotenuse^2=squared_length
bound squared_length=[0,100]
bound hypotenuse=[0,10]
```

The reference vector is squared_length=25 and hypotenuse=5. The extra unknown does not create another physical degree of freedom because its defining equation fixes it. Both equalities must hold in the same solution.

Auxiliaries can also expose a missing condition. If q=x^2, then q is nonnegative mathematically. If instead x=q^2, x is nonnegative and q may have two signs. Reversing the roles changes the model even though both lines contain a square.

Do not multiply variables and equations without a purpose. A direct expression may be clearer than a chain of named intermediates for a small problem. Choose the formulation that makes assumptions, domains and units easiest to inspect. Equivalent mathematics can still have different scaling or search behavior, so preserve the formulation used when comparing results.

3 14 Preserve domains when transforming equations

Algebra can simplify a model, but a transformation may introduce or remove candidates. The final check belongs to the original relationships and their domains. This matters even when the transformed equations are easier to search.

Consider:

```
sqrt(x)=x-2
bound x=[0,10]
```

Squaring gives x=(x-2)^2, or (x-1)*(x-4)=0. The candidates are one and four. At x=1 the original sides are one and minus one, so that candidate fails. At x=4 both sides are two. Only four is valid. The transformation must preserve the nonnegativity of the original right-hand side.

Multiplying away a denominator needs similar care. For (x^2-1)/(x-1)=0, x=1 is undefined. The valid root is x=-1. Replacing the quotient equation with x^2-1=0 while forgetting the exclusion introduces the invalid candidate one. Simply cancelling the common factor is valid only on the original domain.

Squaring a norm can also lose a sign requirement. The equality sqrt(x^2+y^2)=r requires r>=0. The polynomial equation x^2+y^2=r^2 alone allows both signs of r. If r represents a radius, restore its nonnegative meaning explicitly.

Taking logarithms requires positive arguments. Dividing by an expression requires it to be nonzero. These are mathematical requirements, not implementation details. Record them before applying the transformation.

A reliable habit is to keep a copy of the original model, document the transformation and substitute every returned vector into the original expressions. Detailed residual checking belongs to Chapter 4, but the choice of which expressions to check is decided during formulation.

3 15 Use stable functions and recognize nonsmooth models

The supplied interaction introduces expm1 and log1p, which are useful when small differences would otherwise lose accuracy. expm1(x) evaluates exp(x)-1, and log1p(x) evaluates ln(1+x). Near zero, these forms are designed to avoid the loss of significant digits associated with subtracting nearly equal values or rounding 1+x too early.

A reference example is:

```
expm1(x)=0.01
bound x=[0,0.1]
```

The exact root is ln(1.01), approximately 0.009950330853. The exponential function is strictly increasing, so this root is unique in the interval. This example demonstrates meaning; it is not a claim that ordinary exp(x)-1 fails at this particular input size.

The real domain of log1p(x) is x>-1. A numerical function's stable evaluation does not remove its domain restriction. Nor does using a specialized function turn normal arithmetic into arbitrary precision. Conditioning, coefficients and the rest of the model remain relevant.

abs, min and max are Shared Core functions that can introduce corners. For example, abs(x)=2 has roots -2 and 2, but abs(x) is not differentiable at zero. floor, ceil and round are Compute-only functions in this version; they are not function calls accepted by the ordinary equation parser.

Mathematically, floor(x)=1 describes an interval rather than isolated roots, but this is not an accepted ordinary-equation input in the current parser. When a variable represents a count, use an integer declaration. A function working in Compute does not make it available in an algebraic model.

Use each function because it matches the intended mathematics. Do not insert abs around a logarithm argument merely to avoid domain errors: ln(abs(x)) admits negative nonzero x and therefore defines a different question from ln(x).

3 16 Keep units notation and comments consistent

The program does not automatically check units or supply physical property databases. Before solving, choose consistent units and convert fixed data accordingly. A numerical answer can satisfy every typed equation while describing an incorrectly converted physical problem.

If distance is in metres and time in seconds, speed should be in metres per second in distance=speed*time. Entering a speed supplied in kilometres per hour without conversion changes the model by a factor of 3.6. Extra arithmetic precision cannot repair that input error.

Comments preserve these choices:

```
/*
Uniform motion teaching model
Distance in metres and time in seconds
Speed is a fixed value in metres per second
*/
constant speed=12.5
distance=speed*time
distance=100
bound time=[0,20]
bound distance=[0,200]
```

The reference values are time=8 and distance=100. Distance is deliberately retained as an unknown fixed by an equation to show the relationships; a shorter model could substitute 100 directly.

US style uses decimal points and comma-separated function arguments and bound endpoints. EU style uses decimal commas and semicolons inside those constructs. For example, the US bound x=[0.5,2.5] corresponds to the EU bound x=[0,5;2,5]. Select the intended number style before entering or importing such text.

Write separate integer declarations rather than a comma-separated list when preparing text for conversion. Plain text does not store the intended style, while a project does. Comments can state the style to help another reader, but they do not automatically select it.

Use // for an end-of-line comment and /* ... */ for a block comment. Keep mathematical typography in explanatory prose and ASCII syntax in runnable expressions. The visual signs ×, ≥ and superscript powers should not replace *, >= and ^ in a copyable model.

3 17 A complete formulation with a feasibility condition

Consider a rectangle whose side lengths have a sum of 13 metres and area 40 square metres. A design convention requires width to be at least height. The task is to find feasible dimensions, not to optimize an objective.

First define the unknowns and data. Width and height are real lengths in metres. Both are positive. The given sum and area provide two equalities. The convention supplies the coupled inequality width>=height.

```
// Rectangle lengths in metres
constant side_sum=13
constant area=40
width+height=side_sum
width*height=area
width>=height
bound width=[1,12]
bound height=[1,12]
```

Eliminate height using height=13-width. The product equation becomes width^2-13*width+40=0, or (width-5)*(width-8)=0. The equality solutions are (5,8) and (8,5). Only (8,5) satisfies width>=height, and both coordinates lie in the stated bounds.

Substitute into every original relationship: 8+5=13, 8*5=40 and 8>=5. The length bounds also hold. This verifies both the equations and the condition rather than checking only the final quadratic.

The intervals [1,12] are valid restrictions for the present data: the two positive equality roots are five and eight. For a reusable model with changing parameters, reconsider the intervals. A bound chosen for one example should not silently exclude solutions after a data change.

Save the model and clear unintended desktop bound overrides. Check syntax and recognized variables, run with a finite budget and export the completed report. If you remove width>=height, expect two mathematical vectors. Their physical equivalence under rotation is an interpretation; the named variable assignments remain distinct.

3 18 Practice exercises

Exercise 1. Write a model for two real numbers with sum 11 and product 24. Use bounds that contain both positive solutions and derive the reference pairs.

Exercise 2. Evaluate (a+b)/2 and a+b/2 at a=8 and b=4. Explain why a syntax check would not identify either expression as the wrong average.

Exercise 3. Rewrite a model with known radius five and unknown circle area using a constant declaration and #pi. State the area and its units if radius is in metres.

Exercise 4. Enter separate equations ln(x)=1 and log10(y)=1. Supply valid finite bounds and give the reference vector.

Exercise 5. Explain why x>=0 does not make ln(x) defined throughout its permitted values. Describe what changes when an arbitrary positive lower bound is imposed.

Exercise 6. Filter the roots of (x+4)*(x-2)*(x-6)=0 so that only x=2 remains. Use supported syntax and a box that initially contains all three roots.

Exercise 7. A text file bounds x between -5 and 5, but the desktop override bounds it between zero and five. Which domain is used? What must be saved to preserve the override?

Exercise 8. Form an integer model for 3*a+5*b=19 with nonnegative counts. Use finite bounds and enumerate all feasible vectors.

Exercise 9. Model a binary decision b and a real remainder r with b+r=3. List both feasible vectors with r bounded between zero and three.

Exercise 10. Square sqrt(x)=x-6, find the polynomial candidates and check them in the original equation. Use a finite interval that contains all candidates.

Exercise 11. Explain why multiplying (x^2-4)/(x-2)=0 by x-2 requires a domain check. State the valid root.

Exercise 12. List the feasible vectors for x^2=1, y^2=4 and x+y>=0, with bounds containing every equality root.

Exercise 13. Write a model for expm1(t)=0.02 in a justified small positive box. Give an exact expression for its root.

Exercise 14. Explain the distinction between ln(abs(x))=0 and ln(x)=0. List the real solutions of each.

Exercise 15. Convert bound x=[0.5,3.5] and min(x,2.5) to EU notation. State why the application style must also be selected.

Exercise 16. A model contains width*height=40 and width+height=13. Add a supported condition selecting the vector with smaller width. Explain why this is feasibility rather than an optimization instruction.

3 19 Answers and discussion

Answer 1. Use x+y=11, x*y=24, bound x=[0,11] and bound y=[0,11]. Substitution gives (x-3)*(x-8)=0. The pairs are (3,8) and (8,3).

Answer 2. The values are six and ten respectively. Both expressions have valid grammar. Only the first represents the intended average of the two numbers.

Answer 3. Use constant radius=5, area=#pi*radius^2 and bound area=[0,100]. The reference area is 25*pi square metres. The application does not infer or check those units automatically.

Answer 4. Bounds x=[1,4] and y=[1,20] admit the roots. The vector is x=#e, y=10. ln and log10 use different bases.

Answer 5. Zero remains permitted by x>=0, while ln(0) is undefined. A positive cutoff makes a restricted domain and excludes smaller positive values; it needs justification if completeness for the original problem matters.

Answer 6. Use bound x=[-5,7], x>=0 and x<=3 with the polynomial equality. The three equality roots are -4, 2 and 6; only x=2 satisfies both conditions.

Answer 7. The desktop override takes precedence, so the effective interval is [0,5]. Save a .pdeq project to retain it. A plain-text export does not preserve the separate override table.

Answer 8. Declare int a and int b separately. Use bound a=[0,6], bound b=[0,3] and 3*a+5*b=19. Checking b=0,1,2,3 leaves only b=2, which gives a=3. The reference vector is (3,2).

Answer 9. Use binary b, b+r=3 and bound r=[0,3]. The vectors are (b,r)=(0,3) and (1,2). The declaration changes b alone; r remains real.

Answer 10. Squaring gives x=(x-6)^2, hence x^2-13*x+36=0 and (x-4)*(x-9)=0. At four the original right side is negative, so it fails. At nine both sides equal three. Only nine is valid; [0,12] contains both polynomial candidates.

Answer 11. The original denominator excludes x=2. The numerator equation alone admits -2 and 2. Checking the original expression retains only x=-2.

Answer 12. The equality pairs are (-1,-2), (-1,2), (1,-2) and (1,2). The sums are -3, 1, -1 and 3. The feasible pairs are (-1,2) and (1,2).

Answer 13. Use expm1(t)=0.02 and bound t=[0,0.1]. The unique root is ln(1.02), approximately 0.01980262730. Strict monotonicity establishes uniqueness in the box.

Answer 14. ln(abs(x))=0 requires abs(x)=1 and admits x=-1 and x=1. ln(x)=0 requires x>0 and admits only x=1. Adding abs changes the domain and solution set.

Answer 15. Write bound x=[0,5;3,5] and min(x;2,5) in EU style. Selecting EU notation tells the parser how to interpret the punctuation. Text alone does not store that selection.

Answer 16. Add width<=height. Of (5,8) and (8,5), only (5,8) remains. The condition tests admissibility; it does not request the minimum of an objective over a feasible region.

3 20 Prepare the model for solution checking

A clear model identifies the unknowns, fixed data and domains without requiring the reader to guess. It uses explicit arithmetic, suitable function arguments and supported declarations. It records assumptions and units, and it preserves domain restrictions when equations are transformed.

Before solving, read every line as a mathematical requirement. Confirm that constants are fixed data, products contain *, all intended conditions are present and no desktop bound override changes the question unexpectedly. Then use F4 to inspect the recognized model and F5 to search.

The next chapter examines the returned evidence. It explains residuals, coordinate accuracy, distinctness and the limits of a finite search. Good formulation makes that checking possible: you can verify an answer only when you have kept a clear record of the original question.

Source note. Product syntax and desktop behavior follow the supplied Patodia Equation Solver Desktop User Guide with the October 2026 modulo update, dated 27 October 2026, especially sections 7 through 12. The supplied book establishes the chapter sequence. Worked examples are independently derived teaching models, not measured application benchmarks.

3 21 Current edition workflow and scope

3 22 Mode headers and data contracts

Algebraic declarations are not initial or boundary conditions. Start an IVP with ode, a regular two-point BVP with bvp, a field with pde, a second-kind integral equation with integral, a regular eigenproblem with eigen, or known-

function definite integration with quadrature. Coordinates and controls are mode-specific. Chapters 16 to 22 cover the original function solvers; Chapters 26 to 32 develop the Version 3 extensions.

Solver settings and variable bounds

3 23 Settings for algebraic searches

Open Compute > Solver Settings. The following are fresh Desktop defaults; saved preferences and projects can restore other values. They are not necessarily the command-line engine defaults.

Setting	Fresh Desktop default	Purpose
Time budget	2 seconds	Approximate numerical/search allowance; 0 removes the time deadline
Maximum solutions	100	Returned algebraic row cap; allowed 1 to 10000
Maximum starts	30000	Multistart search budget
Iterations per start	300	Local iteration ceiling
Residual tolerance	1e-9	Algebraic acceptance threshold
Random seed	42	Helps reproduce random search choices
Initial automatic search radius	10	Initial automatic range scale
Automatic range growth factor	10	Expansion between automatic stages
Automatic range stages	4	Number of automatic range stages
Integer neighbours	2	Additional nearby integer assignments
Use deflation	Selected	Helps search for additional roots
Infer search structure	Selected	Enables eligible structure-based processing
Use search slices	Selected	Enables additional search slices

The time allowance excludes some startup, parsing and final reporting overhead and can be exceeded within a numerical operation. Zero does not remove row, iteration, mesh, factorisation or storage limits. A larger random budget can help a heuristic search but cannot establish that every root was found.

Explicit bounds stay fixed; automatic radius expansion does not enlarge them. Use sensible finite bounds for engineering variables. If a feasible point is known, substitute it independently before assuming the search should find it. An invalid or excessively tight tolerance can increase failures without improving useful accuracy.

3 24 Variables and Bounds dialog

1. Enter an algebraic model and choose Compute > Variables and Bounds.

2. The application parses the model and lists Variable, Type, Lower and Upper.

3. Enter both endpoints for each override you want. Leave both blank to use model bounds or the automatic range.

4. Choose OK and solve again. Save a .pdeq project to preserve the overrides.

Variable and Type are read-only here. Change integer/binary declarations in the editor. Dialog overrides take precedence over model-text bounds and are not shown by rewriting the editor. An overlooked override can therefore explain an apparently missing root. Clear it by leaving both endpoint cells blank.

The Variables and Bounds dialog is for algebraic unknowns. Differential and field intervals, meshes and initial/boundary data belong in the model text. Do not add algebraic bounds to an ODE merely to set its integration interval.

3 25 Which controls apply to which problem

Problem	Controls in Solver Settings	Main controls in model text
Algebraic roots	Time, starts, iterations, solutions, tolerance, range and search options	Types, equations, conditions and hard bounds
Exact integer routes	Time and solution cap where applicable	Exact equation, integer declarations and bounds
ODE IVP	Time and precision	Method, step controls, rtol, atol, output and events
BVP	Time and precision	Boundary conditions, guesses, defect tolerances and mesh limits
PDE or integral	Time and precision	Grid, refinement, data and relevant linear/boundary controls
Eigenproblem	Time and precision	Modes, p, q, weight, boundaries and accuracy targets
Quadrature	Time and precision entitlement	Integrand, variable, interval, tolerances and quadrature budgets

3 26 Workflow language

Start the file with workflow. In Windows Desktop, open the file, choose Compute → Check Syntax if you want a preliminary check, then choose Compute → Solve. Read Run Report for each job's full output. The workflow table shows reports and job statuses, including the loop inputs associated with each row.

Write one workflow statement per line, start with workflow, and put each closing end on its own line. Indent the body for readability. Use // for line comments. Names start with a letter or underscore and may contain digits after the first character. Use lowercase keywords and descriptive names such as root_count. Choose names distinct from service keywords and constants.

Conditions accept Booleans or a comparison using ==, !=, <, <=, > or >=. Compare textual statuses with quoted strings, such as if area.status == "integration_complete". Use explicit multiplication in workflow expressions, such as 2*x. Loop iterations and branches have local scope. Initialize accumulators outside the loop so updates persist.

A condition can be a Boolean field such as area.accepted or a single comparison. For compound tests, nest if blocks. An assignment without let updates the nearest existing binding. Use let for a fresh local binding, and use set when a missing binding should raise an error. Variables declared inside a loop or branch disappear when that block ends. Define accumulators before entering the loop.

Define a numerical function at the top level with function name(parameters) = expression. It uses its explicit numerical parameters and can call other workflow functions, including functions defined later in the script. Each invocation sees only its own parameters. Pass values from the surrounding workflow as arguments. The engine allows up to 32 active calls.

A while loop rechecks its condition at the start of each iteration. It ends when the condition is false, a break exits it, or the workflow stops at a resource limit. Older scripts with an explicit iteration cap continue to honour it.

```
workflow
function improve(x) = (x + 2/x)/2
x = 1
iterations = 0
while abs(x*x - 2) > 1e-12
    x = improve(x)
    iterations += 1
end
report x
report iterations
```

Expected results are 1.414213562373095 and 5.

3 27 Reusable models and service jobs

Write within <seconds> for a per-job budget. The word in belongs to a loop header such as for i in 1..10. A service block starts with its keyword and a unique result name in the current block. Inside it, use the standalone service's directives without repeating its keyword. Close the block with end.

Define a model at the top level with model name(parameters) as service and end. Use @parameter in the service body to insert a numeric value. Call it with run result = name(arguments). A supported service can also take run result = name(arguments) within seconds. Pass every model parameter explicitly. For a direct job, @name or @root.x reads a numeric value from the enclosing workflow. Declare solver bounds inside the solve body.

Use [] or a list literal to create a collection. append name with expression adds a value. US syntax also accepts append name, expression. count(name) counts entries, and at(name, index) uses indices starting at 1. A collection can contain numbers, records, or other collections. Store results in a collection initialized outside the loop to retain them across iterations.

```
workflow
sum = 0
for i in 1..10
    sum += 1/i
end
report sum
```

Expected result is approximately 2.928968253968254.

The reports are 9, 7, 5 and 3. For nested loops, break and continue act on the innermost loop. The enclosing for values appear in each report's case fields.

Use break and continue inside loops. See Chapter 31 for a reusable quadrature model and Chapter 43 for a script that passes values between services.

3 28 Mathematical functions in workflows

All mathematical functions, including every function available in Compute, are available to Version 4.0 workflows. Call shared core and scalar Compute functions directly in workflow expressions. Use a compute job for list-valued Compute output such as primefactors. Inside a service job, the service’s own expression grammar applies. To use a Compute scalar in another service, evaluate it in the workflow and substitute the numeric result with @name.

Appendix D gives function signatures, domains and branch conventions. Its Compute-only classification refers to standalone service grammars.

```
workflow
a = gamma(3)
quad area
    expr = @a*x
    var = x
    range = [0,1]
end
if area.accepted
    report area.value
else
    report area.status
end
```

Gamma(3) = 2, so the expected accepted integral is 1.

Chapter 4 Understanding and Checking Solutions

A solver result is the beginning of an assessment, not the end of the mathematical work. The previous chapters showed how to formulate a model and run it. This chapter explains how to decide what the returned values establish, whether they meet the original requirements and how confidently they can be reported.

Four questions guide the discussion. Do the values satisfy the equations numerically? Do they satisfy the domains and admissibility conditions? Are the reported solutions distinct? Is there evidence that the relevant solution set is complete? A calculation can answer one of these questions without answering the others.

Use the complete matching Windows 4.0 package for the current edition. Keep its worker, Desktop, runtime DLLs, examples and guides together. Older installer filenames and layouts are historical packaging details and do not establish which engine is running now.

4 1 Read a solution as a named vector

For a single equation in one unknown, a solution may be one number. For simultaneous equations, it is a complete assignment of values to all unknowns. Read every value under its variable name. A row without its headings can be misunderstood when columns are reordered or when variable names do not follow numerical order.

Consider the model:

```
x+y=7
x*y=10
bound x=[0,7]
bound y=[0,7]
```

The reference solutions are (x,y)=(2,5) and (5,2). The value x=2 by itself is incomplete information. It belongs with y=5 in the first vector. Combining x from one row with y from another would create (2,2), which satisfies neither original relationship.

The ordering of solution rows is also different from the ordering of variables. One run may list (2,5) first and another may list (5,2) first. Compare vectors by their named coordinates rather than by row number. A change in row order does not indicate a change in the underlying mathematics.

Some applications identify physically interchangeable configurations. Rotating a rectangular panel may exchange width and height without producing a new physical object. The numerical vectors remain distinct assignments to named unknowns. State which interpretation is being counted when reporting a solution total.

In the desktop, double-click a result to inspect its full details. Scroll horizontally for wide systems. Use the completed Run report to identify the model and settings that produced the table. If the editor has since been changed, its current text may describe a different problem. Solve again before treating the old rows as answers to that edited model.

4 2 Turn each equality into a residual

For an equality L=R, define the residual r=L-R. At an exact solution, r=0. At an approximate solution, the sign and magnitude describe the remaining disagreement between the two sides. Either subtraction order is acceptable if it is used consistently; the absolute residual is unchanged.

Take x^2=2 and the candidate x=1.4142. The residual is:

```
r = 1.4142^2-2
r = -0.00003836
abs(r) = 0.00003836
```

The negative sign says that the candidate's square is smaller than two. It does not mean the candidate is physically negative or numerically invalid. The magnitude measures equation error in the scale of this particular expression.

For a system, calculate a residual for each equality. If the model is x+y=7 and x*y=10, use r1=x+y-7 and r2=x*y-10. At the exact reference pair (2,5), both vanish. Satisfying only the sum is insufficient: many pairs have sum seven without product ten.

Residuals should be evaluated from the original relationships where practical. A transformed polynomial may include candidates that the original square root or denominator excludes. First establish that the original expressions are defined; then calculate their residuals.

Do not demand a printed literal zero as the only evidence of a good numerical result. Floating-point arithmetic and finite decimal coordinates often leave a small remainder. Conversely, a printed zero can result from rounding or cancellation. Its meaning depends on the arithmetic, expression and retained digits.

The useful question is whether the disagreement is acceptably small for the stated model and purpose. That requires more context than counting zeros in a formatted column.

4 3 Check a complete approximate vector

The following example shows why every relationship should be checked. Use the circle-and-line model from Chapter 1:

```
x^2+y^2=25
x-y=1
bound x=[0,5]
bound y=[0,5]
```

The exact feasible vector is (4,3). Suppose a result has been shortened for a hand calculation to x=4.0001 and y=3.0001. These are illustrative rounded candidate values, not an actual application output.

The line residual is zero because 4.0001-3.0001-1=0. The circle residual is different:

```
r_circle = 4.0001^2+3.0001^2-25
r_circle = 0.00140002
r_line = 4.0001-3.0001-1
r_line = 0
```

Checking the line alone would make this pair appear exact. The circle exposes the remaining error. The maximum absolute equality residual is 0.00140002 for these candidate coordinates.

Now consider x=4.0001 and y=3. The line residual becomes 0.0001, while the circle residual is 0.00080001. Both are nonzero. A single maximum tells you the larger magnitude, but the individual values explain which relationships fail and by how much.

The bounds must be checked too. Both illustrative pairs lie in the permitted box, so the issue is equality accuracy rather than domain exclusion. A pair outside the box would fail even if its equation residuals were zero.

For an important model, prepare one residual expression per equation and evaluate all of them at the full exported vector. This systematic approach is more reliable than selecting one visually convenient relationship or checking only the equation that seems most difficult.

4 4 Interpret the maximum residual carefully

The desktop's Max residual field summarizes the largest reported residual or violation for an accepted candidate at the selected precision. It is useful for screening results, but a maximum hides the pattern of smaller entries and does not explain the units or scaling of the equations.

For equality residuals r1 through rm, a common mathematical summary is max(abs(r1),...,abs(rm)). This is the largest absolute component. It does not count how many relationships contribute errors of nearly the same size. Nor does it tell you whether the largest component comes from a balance, a geometry equation or another condition.

A hypothetical vector with residuals 1e-10, 2e-10 and 8e-10 has a maximum of 8e-10. Another vector with all three residuals equal to 8e-10 has the same maximum. Their error distributions differ even though the summary agrees.

Read the reported field in the context of the installed application and completed settings. If you independently calculate a maximum from equality residuals alone, label it as your equality-residual check. Do not assume it reproduces every internal constraint treatment or reporting detail.

The guide gives a normal default acceptance tolerance of 1e-9 and an extended default of 1e-24. Saved projects or custom settings can change the active tolerance. Acceptance means that the program's numerical criteria were met for that run. It is not a universal guarantee about coordinate digits, physical safety or completeness.

A good report retains the application value and supplements it with independent checks where needed. If the two appear inconsistent, compare the full model, effective bounds, arithmetic precision and exact decimal coordinates before concluding that the solver or checker is wrong.

4 5 Distinguish equation error from coordinate error

Residual error and coordinate error measure different things. For a known exact root r and candidate x, the coordinate error is x-r. The residual is F(x), where F defines the equation F(x)=0. Their relationship depends on the shape and scale of F near the root.

For F(x)=x-2, the two errors are equal. At x=2.001, the coordinate error and residual are both 0.001. This convenient agreement comes from the unit slope of the expression; it does not generalize to every equation.

For G(x)=1000*(x-2), the same candidate has coordinate error 0.001 but residual one. The exact root has not changed. Only the scale used to express the equation has changed.

A repeated root shows a more striking difference. Consider:

```
(x-1)^2=0
```

At x=1.000001, the coordinate error is 1e-6 while the residual is 1e-12 in exact decimal arithmetic. The residual is the square of the coordinate error. A very small remainder therefore does not justify claiming twelve correct decimal places in x.

For x^2-2=0 near its positive root, the identity x^2-2=(x-sqrt(2))*(x+sqrt(2)) relates the two errors. Dividing the residual by x+sqrt(2) gives the coordinate error exactly in real arithmetic. This relation is specific to this equation.

When the exact root is unknown, coordinate accuracy generally cannot be read directly from a residual column. Additional analysis, sensitivity estimates or certification may be needed. State that distinction explicitly when reporting many digits from an important calculation.

4 6 Recognize conditioning and nearly dependent equations

Conditioning describes how sensitive the mathematical answer is to small changes in the problem. It differs from arithmetic precision and from the success of a particular search. Even a correctly computed solution can change substantially when uncertain input data are perturbed.

For a scalar equation near a simple root, elementary calculus gives F(x) approximately equal to F'(r)*(x-r). When the derivative is not small and the approximation is appropriate, coordinate error is roughly residual divided by the local slope. A small slope can amplify the coordinate uncertainty associated with a given residual.

Systems have a related difficulty when equations are nearly dependent. Consider these two linear relationships:

```
x+y=2
x+1.000001*y=2.000001
```

Subtracting the first from the second gives 0.000001*y=0.000001, so x=y=1. Now change only the second right-hand side to 2.000002. The difference becomes 0.000001*y=0.000002, giving y=2 and x=0.

A change of one millionth in one datum has moved each coordinate by one. The computation may satisfy the perturbed equations exactly and still give a very different answer. This is sensitivity in the mathematical model, not evidence that the program ignored precision.

In multivariable calculus, the Jacobian matrix collects first derivatives of the residuals. A nearly singular Jacobian can indicate sensitive or nearly dependent relationships. Formal estimates need appropriate scaling and a local analysis; this chapter does not require you to calculate a condition number for every example.

The practical lesson is to look for relationships that almost repeat each other, extreme scale differences and multiple roots. If the data are measured rather than exact, test plausible perturbations and report how the answer changes. More digits in the arithmetic cannot restore information absent from the measurements.

4 7 Interpret residuals in the scale of the equations

Multiplying an equality by a nonzero constant leaves its exact solution set unchanged but changes its raw residual. The equations x-2=0 and 1000000*(x-2)=0 have the same root. At x=2.000001, their residuals are 1e-6 and one respectively.

This matters in applications where equations use different units or characteristic magnitudes. A money balance in paise has a residual one hundred times its value in rupees. The accounting discrepancy has not changed; its numerical representation has.

Choose scales that help interpretation. If an equation represents a total flow of about 1000 units, a residual of 0.001 units has a different relative meaning from the same residual in a relationship whose entire typical magnitude is 0.01 units.

For an independent engineering check, you may divide each residual by a documented characteristic scale for that relationship. For example, compare a mass-balance residual with a representative flow and a temperature-equation residual with an appropriate temperature scale. Report both the scale and the resulting dimensionless measure.

Such normalization is a checking choice, not an undocumented description of the solver's internal algorithm. Avoid dividing blindly by a right-hand side that may be zero. A zero reference value requires another meaningful scale, rather than an undefined relative error.

Scaling can also change how a fixed numerical tolerance influences acceptance. Do not rescale an equation merely to make a residual appear smaller. Preserve the original formulation and explain any transformed version used for numerical work.

For comparisons between products or runs, keep equations, units, scales and tolerance definitions aligned. A residual of 1e-12 from one formulation cannot automatically be ranked above 1e-9 from another. First establish that the numbers measure comparable errors.

4 8 Check domains before accepting a residual

A residual is meaningful only when the original expression is defined. Undefined logarithms, zero denominators and trigonometric poles cannot be repaired by a small residual in a different transformed equation.

For sqrt(x)=x-2, squaring produces candidates one and four. At x=1 the square root is defined, but the original sides have opposite signs: one and minus one. Only four is valid. The polynomial residual alone is insufficient because the transformation introduced an extra candidate.

For (x^2-1)/(x-1)=0, the original expression is undefined at x=1. Multiplying through by x-1 gives a numerator equation with roots minus one and one. Only minus one survives the original-domain check.

A logarithm provides another warning. ln(x^2) is defined for every nonzero real x, including negative x. It is not equivalent to 2*ln(x) over that full real domain, because the latter requires positive x. An algebraic identity can need domain qualifications even when it looks familiar.

Near a pole of tan(t), a tiny change in t may produce a large value change. A candidate close to a pole deserves careful direct evaluation with sufficient retained digits. Proximity to a pole is not itself a root, and a sign change across a discontinuity does not establish that a continuous crossing occurred.

Reject nonfinite coordinates and undefined evaluations. Do not interpret NaN or infinity as unusually accurate or very large roots. Preserve the original equations and domains in the checking record so that an independent calculation tests the same mathematical question.

4 9 Check bounds inequalities and discrete values

Equalities are only part of feasibility. A vector may satisfy them and still violate a bound, an inequality, integrality or the physical interpretation. Check these requirements individually instead of treating a low equality residual as complete approval.

For x^2=4 with bound x=[0,3], the value minus two has zero equality residual but is inadmissible. For x+y=5 and x*y=6 with x<=2.5, the pair (3,2) satisfies both equalities but violates the condition. Only (2,3) is feasible.

An inequality can be assessed with a nonnegative violation measure. For g(x)<=0, use max(g(x),0). For h(x)>=0, use max(-h(x),0). These mathematical definitions help an independent checker distinguish a satisfied condition from an actual excess.

For example, at x=3 the condition x<=2.5 has violation 0.5. At x=2 it has violation zero; the slack is 0.5. Slack and violation are different: a point comfortably inside an inequality need not make its two sides equal.

The guide states that inequality conditions use the residual tolerance while explicit bounds provide the hard search interval. Inspect near-boundary values with their full digits and completed settings. Do not turn a small accepted violation into a claim of exact symbolic equality or a guaranteed physical margin.

For an integer unknown, inspect proximity to an allowed integer and verify the intended integer vector. If a display shows 2.9999999999999996, it may be a numerical representation near three. If you report it as three, substitute that intended value back into the original relationships. Never round an arbitrary continuous solution and assume the balances remain satisfied.

Finally, verify practical conditions that were not encoded. A geometric model may give a mathematically valid configuration that a mechanism cannot reach without collision. Record that as an additional application check rather than claiming the solver tested an absent requirement.

4 10 Separate distinct roots from repeated approximations

A solution count is useful only if its members are valid and distinct. Two slightly different decimal rows can approximate one root. Conversely, two close mathematical roots can be hard to distinguish at a chosen precision or numerical separation threshold.

The polynomial (x-1)^3=0 has one distinct real root, x=1, with multiplicity three. Multiplicity describes algebraic structure. It does not require a report of three identical numerical rows.

Now consider (x-1)*(x-1.0001)=0. There are two distinct roots separated by 0.0001. Candidates 1.0000000000001 and 0.9999999999999 are both near the first root; their small numerical difference does not establish two separate roots.

A particularly instructive candidate is the midpoint x=1.00005. Its two factors are 0.00005 and -0.00005, so the residual is -2.5e-9. Multiplying the whole equation by 0.0001 reduces that residual to -2.5e-13 without moving the midpoint closer to either root.

This example combines close roots, scaling and small residuals. A small remainder alone does not identify which root has been approximated or prove that two nearby roots have been resolved separately. Use fuller coordinates, refinement and an independently justified separation analysis when the distinction matters.

For several variables, compare entire named vectors and account for differing coordinate scales. A fixed absolute distance of 1e-6 may be negligible for a million-unit quantity but substantial for a micro-scale one. State the comparison rule used in an independent root-matching exercise.

Do not infer the solver's proprietary distinctness thresholds from the displayed decimal places. Report observed candidate separations and the evidence supporting their interpretation. More displayed digits can help inspection, but do not automatically certify distinct roots.

4 11 Distinguish numerical alternatives from physical alternatives

Symmetry and periodicity can make the same physical state appear in several numerical representations. This does not invalidate the vectors, but it changes how an application should describe the answer set.

For a rectangle, (width,height)=(5,8) and (8,5) are different named assignments. If orientation is irrelevant, they describe one shape up to rotation. If width refers to a fixed horizontal direction in a constrained opening, the two may represent different feasible arrangements.

Angles give another example. A rotation of theta and theta+2*pi has the same orientation in many models. If angles are allowed over several periods, a solver can return multiple coordinate vectors representing repeated orientations. The count of numerical roots then differs from the count of physical configurations modulo a full turn.

Choose a representation suited to the question. A principal angle interval may avoid repeated representations, but its endpoints require attention because both can describe the same orientation. Record the convention rather than silently removing rows that seem repetitive.

For mechanisms, permutations of identical components can introduce additional symmetry. For a chemical model, several nonnegative steady states can instead represent physically different operating points. The correct interpretation depends on the definitions and governing relationships, not merely on whether the coordinates look similar.

Separate three counts when necessary: accepted numerical vectors, independently distinct mathematical roots in the stated domain and physically inequivalent states under the model's symmetry. A report need not always calculate all three, but it should say which one its headline number represents.

This distinction is especially useful when presenting a many-root demonstration. A larger numerical count can reveal alternatives, but the practical significance comes from explaining what those alternatives mean.

4 12 Verify full exported values independently

Independent substitution provides a second calculation of whether the supplied coordinates satisfy the original relationships. Independence is strongest when the checker uses a separate implementation or higher-precision arithmetic and when the residual expressions have been reviewed against the original model.

Begin with the full exported coordinate strings. The desktop JSON export preserves long numerical values as decimal strings. Keep those strings intact until they are read by the checker's chosen numerical type. Converting first to ordinary double precision and only then to a higher-precision type cannot recover digits already lost.

Ordinary spreadsheet arithmetic can be useful for normal-precision checks of modest models. It is generally unsuitable for confirming residuals at 1e-24. A high-precision calculator or an appropriate arbitrary-precision library is better for that purpose. Keep the working precision and the exact strings with the checking record.

Use the original equations, including named constants and domains. If the solver model used a rounded coefficient, the checker must use that same coefficient when checking the model's numerical answer. A separate comparison with a more accurate physical constant addresses a different question.

A reported residual may differ slightly from independent substitution because the exported coordinates are finite decimals, arithmetic differs or evaluation orders differ. Investigate the magnitude and cause. Do not expect exact bit-for-bit agreement between unrelated implementations.

Substitution confirms agreement at the supplied point. By itself, it does not prove that an exact root lies nearby, that the root is unique in a neighbourhood or that every root in the domain has been found. Those stronger claims need additional mathematical evidence.

For a classroom exercise, a hand derivation and substitution may be sufficient. For a published benchmark or consequential engineering calculation, retain a reproducible independent checker and describe precisely what it verified.

4 13 Preserve digits without confusing them with accuracy

Printed digits, arithmetic precision and correct digits are separate concepts. A program can print more decimal characters than the arithmetic meaningfully supports. A number may also be accurately computed for a model whose coefficients are only rough approximations to physical data.

Normal double arithmetic provides about 15 to 16 significant decimal digits of precision. The supplied guide describes binary128 extended arithmetic as providing approximately 33 significant decimal digits. These are arithmetic capabilities, not promises that every coordinate in every model has that many correct digits.

The desktop uses double candidate discovery followed by binary128 refinement in extended mode. Refinement can improve an already discovered candidate and its residual. It does not guarantee that a root missed during discovery will appear merely because the output mode has more digits.

A simple rounding example shows the value of retaining coordinates. For x^2=2, the candidate 1.4142135623730951 is much closer to the positive root than 1.4142. If a table is shortened to four decimal places before checking, the residual reflects that shortening rather than the quality of the original value.

Use full values for verification and sensible rounded values for presentation. Label a shortened number as approximate. For measured physical quantities, report units and an appropriate precision based on the data and sensitivity, rather than filling a report with every available digit.

A displayed residual smaller than a familiar machine epsilon is not automatically impossible. Machine epsilon describes relative spacing near one in a particular arithmetic; it is not a universal minimum for every representable residual. Nor does a residual equal to epsilon prove maximum possible coordinate accuracy.

When precision changes, confirm the active tolerance. A custom value can remain in force. Record both fields so that another reader can understand whether a changed residual came with changed arithmetic, changed acceptance criteria or both.

4 14 Understand root discovery and completeness

Finding one verified vector gives numerical evidence for one solution. Finding several verified, distinct vectors gives evidence for several solutions. Claiming that these are all solutions is a different assertion because it also excludes every unreported point in the stated domain.

Elementary algebra can sometimes establish completeness. For x+y=7 and x*y=10, substitution gives x^2-7*x+10=0. Its only roots are two and five, and both produce valid pairs. This proves that the two checked vectors exhaust the original system.

For sin(t)=0 on [-4,4], the zeros are integer multiples of pi. Only -pi, zero and pi lie in that interval. Without a finite domain, the same equation has infinitely many real roots. No finite numerical list can enumerate them all.

A complicated nonlinear system may not admit such a short proof. Optional interval certification is a separate advanced engine workflow, not a desktop menu feature in the supplied guide. When used appropriately, a certification result must be interpreted within its documented hypotheses, bounds and conclusions.

Certification can have several possible scopes. An isolated box may establish a unique root locally, while a domain-wide argument may establish that no other boxes remain. Unresolved regions prevent a complete exclusion claim. Do not replace a qualified certification outcome with a blanket statement that every root was proved.

If no completeness argument is available, use precise wording: the run found a stated number of distinct accepted candidates within a specified domain and budget. This is still useful information. Clear limits make the observation reproducible and prevent a numerical search result from being mistaken for an exhaustive mathematical theorem.

4 15 Check congruences and residue representatives

For a model containing %, first check that every returned coordinate is an integer and that every modulus is positive. Recompute each remainder using the mathematical convention: the result lies between zero and modulus minus one. For negative numerators, Python uses this convention with positive moduli; C++ requires normalizing a negative remainder.

For x=6 and y=8, the expressions x % 5, y % 6, (y-x) % 3, (x+y) % 9 and (x*y) % 7 return 1, 2, 2, 5 and 6. A displayed zero residual confirms those equations at the returned point. It does not convert a representative into a complete list of an infinite solution set.

When the report says one fundamental period was enumerated, record its periods with the result. The five-congruence example has 18 representatives with periods x=315 and y=126. For any one representative (x0,y0), adding 315*m and 126*n preserves the equations; choose integer shifts that also preserve declaration signs and any explicit bounds. Time and solution limits can leave the representative list incomplete.

With explicit finite bounds for every variable, the modulo search enumerates points in that box rather than a single representative period. Preserve those bounds in the exported model. The JSON search_complete field still belongs to real-domain certification; use the modulo stopping reason to identify bounded enumeration, representative enumeration, a time limit or a solution limit.

4 16 Interpret missing roots and unsuccessful searches

No root found is an observation about a run, not automatically a proof that the model has no solution. The search may have insufficient allowance, an inappropriate domain, a restrictive cap or a difficult numerical structure. The input may also be inconsistent. Diagnose these possibilities separately.

First compare the completed model with the intended one. Check signs, constants, function names, number style and active bound overrides. A missing negative root can be explained immediately by a nonnegative search interval. More time will not restore a root deliberately excluded by the domain.

Next inspect the stopping settings. A maximum solution count of one prevents a claim that a two-root problem was exhaustively searched. Starts, iterations and other limits can end useful exploration even when the time allowance is larger. Read Diagnostics and the completed report.

Then review mathematical consistency and domains. For x^2+1=0, the absence of real roots follows from nonnegativity of x^2. That proof remains valid regardless of the search outcome. For a model that should have a known reference root, directly substitute it to check that the typed coefficients match the reference formulation.

Only after these checks should you conduct controlled search changes. Increase time or try another seed while preserving the model and recording what changed. Additional roots demonstrate additional discovery. Repeated failure to find a root does not, by itself, demonstrate nonexistence.

An interrupted desktop run also needs careful interpretation. The documented Stop action terminates the numerical worker before final results are returned; it does not provide partial roots from that interrupted calculation. Earlier completed results may remain visible. Confirm which run a visible table belongs to before reporting success or failure.

4 17 Report timings and claims at their proper scope

Found (s) is a within-run acceptance timestamp for a candidate. It is not the entire application runtime. If several rows show 0.03, 0.08 and 0.05 seconds, their earliest listed acceptance is at 0.03 seconds and their latest at 0.08 seconds. Adding them does not measure search duration.

The worker can continue searching after the latest reported candidate. A configured time allowance, last acceptance time and externally measured process wall time answer different questions. Launch, parsing, refinement and reporting can also contribute overhead beyond the cooperative search budget.

For benchmarks, identify the quantity being measured. Record the exact model, bounds, precision, tolerance, seed, stopping settings, application version and hardware. Compare like with like. A different domain or a lower acceptance threshold can change both the returned set and the work required.

Avoid claims about a proprietary numerical method based on speed or output alone. A short time does not establish symbolic elimination, interval arithmetic or GPU acceleration. A small residual does not identify the internal search architecture.

A defensible result statement might read: the completed run returned two candidate vectors; both satisfy the original equations under an independent residual check and meet the stated bounds. Algebraic elimination establishes that these are the only real solutions of this example.

For a harder model, the last sentence may instead say that completeness was not established. For an application, add what physical admissibility checks were performed. A steady-state root, for example, does not by itself establish dynamic stability. Match each claim to the evidence that actually supports it.

4 18 Practice exercises

Exercise 1. For x^2=3 and candidate x=1.732, calculate the signed residual and its absolute value. Distinguish both from coordinate error.

Exercise 2. For x+y=7 and x*y=10, check the illustrative candidate x=2.001, y=4.999. Calculate both residuals and their maximum absolute value.

Exercise 3. At x=1.0001 in (x-1)^2=0, compare coordinate error and residual. Explain why four powers of ten separate their magnitudes.

Exercise 4. Compare the residuals of x-3=0 and 1000*(x-3)=0 at x=3.0002. Do the equations have different exact roots?

Exercise 5. The equations x+y=2 and x+1.000001*y=2.000001 have root (1,1). Change the second right-hand side to 2.000003 and find the new root. What does the change illustrate?

Exercise 6. Check the candidates x=4 and x=9 in sqrt(x)=x-6. Explain why a zero residual in the squared equation does not validate both.

Exercise 7. A candidate x=-2 satisfies x^2=4. Is it feasible under bound x=[0,3]? Identify the failed requirement.

Exercise 8. For the condition x+y<=5, calculate slack and violation at (2,2) and at (3,3).

Exercise 9. Does (x-2)^4=0 have four distinct roots? How should algebraic multiplicity be distinguished from a numerical row count?

Exercise 10. The vectors (width,height)=(4,7) and (7,4) both satisfy a rectangle model. Describe one situation where they are physically interchangeable and another where they are not.

Exercise 11. Explain why parsing long JSON coordinate strings into double before passing them to an arbitrary-precision checker can weaken verification.

Exercise 12. List the roots of sin(t)=0 on [-7,7]. Give a completeness argument independent of a numerical search.

Exercise 13. Three rows have Found times 0.01, 0.12 and 0.04 seconds. What are the earliest and latest listed acceptance times? Can you determine process wall time from them?

Exercise 14. A five-second run and a ten-second run return the same seven roots of an unfamiliar system. State what can be concluded and what further evidence is needed to claim completeness.

Exercise 15. A normal-precision result has residual 1e-12. An extended result has residual 1e-25. Explain why this does not alone establish either a particular number of correct coordinate digits or a complete solution set.

Exercise 16. Draft a short report for a two-root example that distinguishes equation verification, admissibility, distinctness and completeness. State which files should accompany it.

4 19 Answers and discussion

Answer 1. The square is 2.999824, so the residual is -0.000176 and its absolute value is 0.000176. Coordinate error is 1.732-sqrt(3), a different quantity.

Answer 2. The sum residual is zero. The product is 10.002999, so its residual and the maximum absolute equality residual are 0.002999. Exact agreement in the sum does not validate the product.

Answer 3. Coordinate error is 1e-4 and residual is 1e-8 in exact decimal arithmetic. Squaring the coordinate error doubles its exponent. A residual of 1e-8 does not establish eight correct decimal places in x.

Answer 4. The residuals are 0.0002 and 0.2. Both equations have the same exact root, three. The factor changes the scale of equation error.

Answer 5. Subtraction gives 0.000001*y=0.000003, hence y=3 and x=-1. A two-millionths change in one datum moves both coordinates by two, illustrating sensitivity from nearly dependent equations.

Answer 6. At four, the sides are two and minus two, so the original residual is four and the candidate fails. At nine, both sides are three. Squaring removed the sign restriction on x-6.

Answer 7. No. The equality holds but the explicit lower bound excludes negative x. Equality accuracy and feasibility are different checks.

Answer 8. At (2,2), the sum is four, slack is one and violation is zero. At (3,3), the sum is six, violation is one and there is no nonnegative slack within the permitted side. The signed margin 5-x-y is minus one.

Answer 9. It has one distinct real root, x=2, of multiplicity four. Four nearly equal numerical rows would not establish four distinct roots.

Answer 10. If orientation is irrelevant, rotating the rectangle exchanges the dimensions without changing its shape. If width must fit a particular horizontal opening, the assignments can have different physical consequences.

Answer 11. The conversion rounds the values to double precision first. Increasing the checker's arithmetic precision afterwards cannot reconstruct the discarded digits. Read the original decimal strings directly into the chosen type.

Answer 12. The roots are -2*pi, -pi, zero, pi and 2*pi. Every sine zero is an integer multiple of pi, and the next positive and negative multiples lie outside the stated interval.

Answer 13. The earliest listed acceptance is 0.01 seconds and the latest is 0.12 seconds. The worker may continue after that, and process overhead is not included, so process wall time cannot be determined from those timestamps alone.

Answer 14. Both runs returned the same seven candidates, and the longer run found no additional reported ones. Completeness needs a domain-specific mathematical argument or appropriate certification that excludes other solutions.

Answer 15. Smaller residual supports closer numerical agreement with the equations as evaluated. Coordinate accuracy also depends on conditioning, scaling and multiplicity. Neither residual excludes unreported roots elsewhere in the domain.

Answer 16. For x+y=7 and x*y=10 in [0,7]^2, report (2,5) and (5,2), verify both equalities and bounds, note the distinct named vectors and use quadratic elimination to establish completeness. Retain the project, full solution export, completed run report and any independent checking record.

4 20 Build a repeatable checking habit

Use the same sequence for each important calculation. Identify the completed model and effective domain, preserve the full vectors, evaluate the original equations, check every admissibility requirement and compare candidates for distinctness. Then decide whether a separate completeness argument is available.

Scale the effort to the purpose. A classroom quadratic can be checked by hand. A sensitive engineering model may need higher-precision substitution, data perturbations and application-specific review. The numerical table is useful in both cases, but the strength of the conclusion depends on the additional evidence.

Keep the language of the result precise. Say which roots were found, how they were checked and whether completeness was established. State physical interpretations separately from numerical agreement. This makes the result easier to trust, reproduce and extend.

The next chapter applies these habits to classical mathematics and geometry. Those problems often permit an exact derivation alongside a numerical model, making them particularly useful for learning how formulation, discovery and verification support one another.

Source note. Desktop behavior follows the supplied Patodia Equation Solver Desktop User Guide with the October 2026 modulo update, dated 27 October 2026, especially sections 9 through 13 and 18. The supplied book provides the chapter sequence and checking themes. Worked candidates and calculations here are independently constructed or derived teaching examples. No application execution time, proprietary algorithm or exhaustive benchmark result is inferred from the supplied conversational material.

4 21 Current edition workflow and scope

4 22 Current result families

Algebraic roots live in solutions. IVP/BVP/field data use trajectory; field results additionally include eigenproblem data where applicable. Sweeps contain nested per-case results. Standalone quadrature has a scalar value and quadrature_successful flag. Read problem_type, status, reason and the applicable success/accuracy flags before using data. A null quadrature value with candidate_value is not acceptance; an accuracy_not_met field can retain inspectable but unaccepted samples.

Reading and copying results

4 23 Algebraic rows

Solutions shows Solution, one column per variable, Max residual and Found (s). A residual measures the entered equations and conditions at the returned point. Small residual is not a guaranteed coordinate-error bound. Found (s) records the point's original acceptance time; sorted row order need not be chronological.

Double-click an ordinary short algebraic cell to open solution details. The table is read-only; modify the model rather than editing returned values. The summary shows returned count, arithmetic/precision and completion information. Run report provides the stopping reason and model used.

No solution found means no accepted point was returned by that run. It is not generally a proof of infeasibility. Exact routes can prove completeness for an eligible integer domain, but only when the applicable completion flag and stopping reason establish it. A returned row cap can make an otherwise exact enumeration incomplete.

4 24 Numerical samples

For IVP, BVP, PDE and integral models, Solutions contains sampled coordinates and states rather than independent algebraic roots. A derivative column such as y' is part of a converted differential state. A PDE row identifies coordinates and u. Read the method and status in the summary, then inspect Run report for residual, refinement or stopping details.

A single initial IVP row is initial data, not proof of successful integration. A terminal event can end a successful IVP before the requested endpoint. A failed BVP or field calculation has no accepted solution trajectory. Plot availability follows the returned data, not merely successful syntax checking.

4 25 Copy selected cells

Click a result cell or drag a rectangular block. Use Ctrl+C while the Solutions grid has focus, or Edit > Copy selected result cells / Ctrl+Shift+C. A block is copied with tabs between cells and line breaks between rows. With no block, the current cell is copied. For several separated selection blocks, the current implementation copies the first block; use one rectangle for predictable output.

Copy in Run report or Diagnostics copies the selected text in that control. The main Edit > Select All command targets the equation editor; do not use it to assume the whole Solutions table has been selected. Use File > Export results for the entire returned dataset.

4 26 Very long values and the known viewer issue

Version 2.0.1 stores entire returned values. Columns containing more than 120 characters use a bounded preview width capped at 520 device-independent pixels. Ordinary cells retain existing sizing. The grid requests a horizontal scrollbar, but display width alone does not determine whether every digit is accessible.

Double-clicking a long cell opens Full value, which is intended to provide wrapping, a digit/character count and Copy full value. A Windows report shows that only part of the solution is visible there. Treat visual wrapping as unreliable in this release; this guide is not a viewer fix. Copy full value is designed to use the original raw text, and normal grid copying retains full cell data. Verify a critical large value through JSON or a plain-text editor rather than relying on the visible fragment.

Export JSON is the preferred record for 500- or 1000-digit integers. Avoid opening a numeric CSV directly in a spreadsheet that may convert long integers to floating point. Import such columns explicitly as text if a spreadsheet is needed. Do not add line breaks or commas to the original stored integer to improve appearance.

4 Workflow results from an algebraic solve

```
workflow
solve linear_pair within 1
    x+y = 5
    x-y = 1
end
for root in linear_pair.roots
    report root
end
```

Expected x = 3, y = 2.

Inspect job.accepted and job.status before relying on roots. A returned root and a complete search establish different claims.

Chapter 5 The Workflow Programming Language

Use workflow as a programming language for mathematical calculations, iteration and reporting. Every listing in this chapter is a complete standalone script. Paste one listing into Desktop and run it to see its reports.

5 1 Script structure and expressions

Write one workflow statement per line, start with workflow, and put each closing end on its own line. Indent the body for readability. Use // for line comments. Names start with a letter or underscore and may contain digits after the first character. Use lowercase keywords and descriptive names such as root_count. Choose names distinct from service keywords and constants.

The listings use US number style. Match Desktop's number style before running them. A report evaluates an expression and records its value. Reports inside a for loop also record the enclosing loop case values.

Element	Syntax and meaning
Numeric values	-2.5, 1e-12, #pi, #e. Use explicit constants in portable scripts.
Arithmetic	+, -, *, /, %, ^ or **. Parentheses group expressions.
Text and Booleans	"finite_limit", true and false.
Fields	For a record value named item, item.value selects the value field.
Collections	[], [1, 2, 3], or [[1,2], [3,4]]. Use at(items, 1) to read the first item.
Number style	US uses at([1.5, 2.5], 2). EU uses at([1,5; 2,5]; 2). Set the style before loading or running examples.

5 2 Variables assignment and scope

A condition can be a Boolean field such as area.accepted or a single comparison. For compound tests, nest if blocks. An assignment without let updates the nearest existing binding. Use let for a fresh local binding, and use set when a missing binding should raise an error. Variables declared inside a loop or branch disappear when that block ends. Define accumulators before entering the loop.

The following example contrasts updating an existing binding with declaring a fresh local binding. It reports 5 inside the branch and 12 after it.

```
workflow
x = 10
set x = x + 2
if x > 0
    let x = 5
    report x
end
report x
```

5 3 Conditions and branches

Use a Boolean value or compare two expressions with ==, !=, <, <=, > or >=. The else branch is optional. Nest if blocks to test several conditions. Use == for equality and = for assignment.

```
workflow
n = 17
if n > 1
    if isprime(n) == 1
        report "prime"
    else
        report "composite"
    end
else
    report "below two"
end
```

Expected report is "prime".

5 4 For loops and loop control

A range includes both integer endpoints. It counts up if the first is smaller and down if it is larger. A for loop can also iterate over a snapshot of a collection or over a list literal. Define an accumulator outside the loop when you need to report it afterward.

```
workflow
sum = 0
for i in 1..10
    sum += 1/i
end
report sum
```

Expected result is approximately 2.928968253968254.

Use continue to skip the rest of the current iteration and break to leave the loop.

```
workflow
for n in 9..1
    if n % 2 == 0
        continue
    end
    report n
    if n < 5
        break
    end
end
```

The reports are 9, 7, 5 and 3. For nested loops, break and continue act on the innermost loop. The enclosing for values appear in each report's case fields.

5 5 While loops and stopping conditions

A while loop rechecks its condition at the start of each iteration. It ends when the condition is false, a break exits it, or the workflow stops at a resource limit. Older scripts with an explicit iteration cap continue to honour it.

The condition is tested before the body, so a while loop may run zero times. Update a quantity used by the condition or arrange for a reachable break.

Assume independent birthdays distributed uniformly over 365 days. Keep multiplying the probability that all birthdays are distinct until the complementary probability reaches one half.

```
workflow
people = 0
distinct = 1
while 1 - distinct < 0.5
    distinct = distinct * (365 - people)/365
    people += 1
end
report people
report 1 - distinct
```

Expected results are 23 and approximately 0.5072972343.

5 6 Collections and indexing

Use [] or a list literal to create a collection. append name with expression adds a value. US syntax also accepts append name, expression. count(name) counts entries, and at(name, index) uses indices starting at 1. A collection can contain numbers, records, or other collections. Store results in a collection initialized outside the loop to retain them across iterations.

Use a dot to select a field from a record value. Use at to read an item from a list. You can nest lists and use at again to select an item from an inner list.

```
workflow
counts = []
for n in [90, 80, 70, 60]
    append counts with primepi(n)
end
report counts
report count(counts)
report at(counts, 2)
```

Expected reports are [24, 22, 19, 17], 4 and 22.

5 7 User defined functions

Define a numerical function at the top level with function name(parameters) = expression. It uses its explicit numerical parameters and can call other workflow functions, including functions defined later in the script. Each invocation sees only its own parameters. Pass values from the surrounding workflow as arguments. The engine allows up to 32 active calls.

The function body is a single expression. Give the function any outside numeric value it needs as an explicit argument. This example calls a function on each pass through a while loop.

```
workflow
function improve(x) = (x + 2/x)/2
x = 1
iterations = 0
while abs(x*x - 2) > 1e-12
    x = improve(x)
    iterations += 1
end
report x
report iterations
```

Expected results are 1.414213562373095 and 5.

5 8 Built in mathematical functions

Workflow expressions call shared core functions and scalar Compute functions directly. The full mathematical catalogue is available to workflows, including Compute functions. This chapter uses scalar calls directly. The list-

valued primefactors result uses Compute output. To create a list using the workflow language alone, append factors returned individually by primefactor(n, i).

Workflow arithmetic uses the selected engine precision. Scalar Compute calls use the high-precision Compute backend. Trigonometric arguments are in radians. The tables give signatures and domains for the workflow expression functions listed in the Workflow Guide.

Shared core functions

Syntax	Meaning and domain	Example
sin(x)	Sine, radians. Domain All real x	sin(#pi/6)
cos(x)	Cosine, radians. Domain All real x	cos(#pi)
tan(x)	Tangent. Domain cos(x) ≠ 0	tan(#pi/4)
cot(x)	Cotangent. Domain sin(x) ≠ 0	cot(#pi/4)
sec(x)	Secant. Domain cos(x) ≠ 0	sec(0)
csc(x)	Cosecant. Domain sin(x) ≠ 0	csc(#pi/2)
asin(x)	Principal inverse sine. Domain −1 ≤ x ≤ 1	asin(0.5)
acos(x)	Principal inverse cosine. Domain −1 ≤ x ≤ 1	acos(0)
atan(x)	Principal inverse tangent. Domain All real x	atan(1)
acot(x)	Principal inverse cotangent, π/2 − atan(x). Domain All real x	acot(0)
asec(x)	acos(1/x). Domain |x| ≥ 1	asec(2)
acsc(x)	asin(1/x). Domain |x| ≥ 1	acsc(2)
atan2(y,x)	Quadrant-aware angle. Domain (x,y) ≠ (0,0); shared evaluator rejects the origin; do not rely on backend-specific values there	atan2(1,1)
sinh(x)	Hyperbolic sine. Domain All real x	sinh(1)
cosh(x)	Hyperbolic cosine. Domain All real x	cosh(1)
tanh(x)	Hyperbolic tangent. Domain All real x	tanh(1)
coth(x)	Hyperbolic cotangent. Domain x ≠ 0	coth(1)
sech(x)	Hyperbolic secant. Domain All real x	sech(0)
csch(x)	Hyperbolic cosecant. Domain x ≠ 0	csch(1)
asinh(x)	Inverse hyperbolic sine. Domain All real x	asinh(2)
acosh(x)	Inverse hyperbolic cosine. Domain x ≥ 1	acosh(2)
atanh(x)	Inverse hyperbolic tangent. Domain −1 < x < 1	atanh(0.5)
acoth(x)	Inverse hyperbolic cotangent. Domain |x| > 1	acoth(2)
asech(x)	Inverse hyperbolic secant. Domain 0 < x ≤ 1	asech(0.5)
acsch(x)	Inverse hyperbolic cosecant. Domain x ≠ 0	acsch(2)
exp(x)	Exponential e^x. Domain All real x; finite output only	exp(1)

Syntax	Meaning and domain	Example
expm1(x)	Numerically stable exp(x) − 1. Domain All real x; finite output only	expm1(1e-20)
log(x), ln(x)	Natural logarithm. Domain x > 0	log(#e)
log10(x)	Base-10 logarithm. Domain x > 0	log10(100)
log1p(x)	Numerically stable log(1+x). Domain x > −1	log1p(1e-20)
sqrt(x)	Principal square root. Domain x ≥ 0	sqrt(2)
cbrt(x)	Real cube root. Domain All real x	cbrt(-8)
pow(x,y)	Real power x^y. Domain x>0 for arbitrary real y; negative x requires an integer exponent in the real power evaluator; use cbrt or Compute-only root for supported real roots	pow(2,10)
abs(x)	Absolute value. Domain All real x	abs(-3.2)
hypot(x,y)	Euclidean norm √(x²+y²). Domain All real x,y; finite output only	hypot(3,4)
min(a,b)	Minimum of two values. Domain All real a,b	min(3,5)
max(a,b)	Maximum of two values. Domain All real a,b	max(3,5)

Scalar Compute functions

Syntax	Meaning and domain	Example
logbase(x,b), logb(x,b)	Logarithm in base b. Domain x > 0, b > 0, b ≠ 1	logbase(8,2)
root(x,n)	Real n-th root. Domain n ≠ 0; negative x requires odd integer n	root(-27,3)
sign(x)	Sign: −1, 0 or +1. Domain All real x	sign(-4)
floor(x)	Greatest integer ≤ x. Domain All real x	floor(-1.2)
ceil(x)	Least integer ≥ x. Domain All real x	ceil(-1.2)
round(x)	Nearest integer; half values away from zero. Domain All real x	round(-2.5)
trunc(x)	Truncate fractional part toward zero. Domain All real x	trunc(-3.8)
frac(x)	Fractional part x − floor(x). Domain All real x; result in [0,1)	frac(-1.25)
factorial(n), fact(n)	Product 1×2×…×n (0! = 1). Domain Integer 0 ≤ n ≤ 2000	factorial(6)
gamma(x)	Euler gamma Γ(x). Domain Real x except nonpositive integers; finite result	gamma(0.5)
lgamma(x)	Logarithm of |Γ(x)| under Boost.Math convention. Domain Real x except nonpositive integers	lgamma(5)

Syntax	**Meaning and domain**	**Example**
beta(a,b)	Euler beta B(a,b). Domain Common regular case a>0,b>0; other real values depend on gamma poles and Boost implementation	beta(2,3)
erf(x)	Gaussian error function. Domain All real x	erf(1)
erfc(x)	Complementary error function 1−erf(x). Domain All real x	erfc(1)
digamma(x)	Logarithmic derivative Γ′(x)/Γ(x). Domain Real x except nonpositive integers	digamma(1)
polygamma(n,x)	n-th derivative of digamma. Domain Integer 0 ≤ n ≤ 8; x>0 recommended (required for n>0 per documented service contract)	polygamma(1,1)
besselj(v,x)	Bessel function J of order v. Domain x≥0 for noninteger v; real-valued result required	besselj(0,2)
bessely(v,x)	Bessel function Y of order v. Domain x>0 (real branch)	bessely(0,2)
besseli(v,x)	Modified Bessel I of order v. Domain x≥0 for general real v; real result required	besseli(0,2)
besselk(v,x)	Modified Bessel K of order v. Domain x>0	besselk(0,2)
airy_ai(x)	Airy Ai solution. Domain All real x	airy_ai(0)
airy_bi(x)	Airy Bi solution. Domain All real x	airy_bi(0)
legendre(n,x)	Legendre polynomial $P_n(x)$. Domain Integer n≥0 (≤10000); real x	legendre(3,0.5)
assoc_legendre(n,m,x)	Associated Legendre polynomial. Domain Integers n,m≥0; m≤n; typically |x|≤1	assoc_legendre(3,1,0.5)
hermite(n,x)	Physicists' Hermite polynomial. Domain Integer n≥0 (≤10000); real x	hermite(3,2)
laguerre(n,x)	Laguerre polynomial. Domain Integer n≥0 (≤10000); real x	laguerre(3,2)
assoc_laguerre(n,α,x)	Generalized Laguerre polynomial (implemented integer α). Domain Integers n,α≥0 (≤10000); real x	assoc_laguerre(3,2,1)
chebyshev_t(n,x)	Chebyshev polynomial $T_n(x)$	chebyshev_t(4,0.5)
chebyshev_u(n,x)	Chebyshev polynomial $U_n(x)$. Domain Integer n≥0 (≤10000); real x	chebyshev_u(4,0.5)
zeta(s)	Riemann zeta ζ(s). Domain Real s ≠ 1; finite real result	zeta(2)
lambertw(x)	Principal real branch $W_0(x)$, W e^W = x. Domain x ≥ −1/e	lambertw(5)
lambertw(x,branch)	Real branch W_0 or W_{-1}. Domain branch integer 0 or −1; W_{-1} on −1/e ≤ x < 0	lambertw(-0.1,-1)
ei(x)	Exponential integral Ei(x). Domain Real x ≠ 0 (principal real value)	ei(1)
si(x)	Sine integral $\int_0^x \sin(t)/t\,dt$. Domain All real x; series/quadrature/asymptotic approximations	si(1)

Syntax	Meaning and domain	Example
ci(x)	Cosine integral $\gamma+\ln(x)+\int_0^x(\cos(t)-1)/t\,dt$. Domain $x > 0$	ci(1)
elliptic_k(m)	Complete elliptic integral K with parameter $m=k^2$. Domain $0\le m<1$ (real finite principal branch)	elliptic_k(0.5)
elliptic_e(m)	Complete elliptic integral E with parameter $m=k^2$. Domain $0\le m\le 1$	elliptic_e(0.5)
elliptic_f(phi,m)	Incomplete elliptic integral of first kind. Domain $0\le m\le 1$; avoid singular integration paths	elliptic_f(1,0.5)
elliptic_einc(phi,m)	Incomplete elliptic integral of second kind. Domain $0\le m\le 1$ for real modulus	elliptic_einc(1,0.5)
elliptic_pi(n,m)	Complete elliptic integral of third kind. Domain $0\le m<1$; $n<1$ for regular real integral	elliptic_pi(0.2,0.5)
elliptic_pi(phi,n,m)	Incomplete elliptic integral of third kind. Domain $0\le m\le 1$; $1-n\sin^2(t)$ must not vanish on integration path	elliptic_pi(1,0.2,0.5)
totient(n)	Euler's totient $\phi(n)$: number of integers from 1 to n relatively prime to n. $1 \le n \le 10^{12}$.	totient(5929) = 4620
gcd(a,b)	Greatest common divisor of nonnegative integers. $0 \le a,b \le 10^{12}$; gcd(0,0)=0.	gcd(42,77) = 7
lcm(a,b)	Least common multiple of nonnegative integers. $0 \le a,b \le 10^{12}$; lcm(0,b)=0.	lcm(14,22) = 154
isprime(n)	Primality predicate: numeric 1 if n is prime, otherwise 0. $0 \le n \le 10^{12}$.	isprime(97) = 1
primepi(n)	Exact number of primes $p \le n$. $0 \le n \le 100{,}000{,}000$ (inclusive).	primepi(1000000) = 78498
mobius(n)	Möbius $\mu(n)$: 0 for repeated prime factors, otherwise (-1) to the count of distinct factors. $1 \le n \le 10^{12}$.	mobius(30) = -1
divisorcount(n)	Number $\tau(n)$ of positive divisors of n. $1 \le n \le 10^{12}$.	divisorcount(360) = 24
divisorsum(n)	Sum $\sigma(n)$ of positive divisors of n. $1 \le n \le 10^{12}$.	divisorsum(28) = 56
primefactor(n,k)	Select 1-based kth prime factor including multiplicity. $2 \le n \le 10^{12}$; $1 \le k \le 64$ and k must exist.	primefactor(5929,3) = 11

5 9 Worked study Sum the first 10000 primes

The script seeds the result with 2, then checks only odd candidates. It uses the scalar isprime function directly in workflow expressions.

```
workflow
// Sum the first 10,000 primes. Expected: 10000, 104729, 496165411.
total = 2
prime_count = 1
last = 2
candidate = 1

while prime_count < 10000
    candidate += 2
    if isprime(candidate) == 0
        continue
    end
    total += candidate
    prime_count += 1
    last = candidate
end

report prime_count
report last
report total
```

Expected count, last prime and sum are 10000, 104729 and 496165411.

5 10 Execution limits and diagnostics

The 10,000,000-step limit applies across the entire run, including loop checks. When step_limit is reported, inspect the reports recorded before the stop. The configured elapsed-time budget can also stop the script. Check loop progress before raising the budget.

Source text is limited to 4 MiB, nesting to 32 blocks and function depth to 32 active calls. A range or collection can contain up to 100,000 items. Range endpoints must lie within ±1 billion. Recorded results are limited to 8 MiB and tracked collection copying and appending to 32 MiB.

Source for the language reference and worked examples: Patodia Equation Solver 4.0 Workflow Guide.

Chapter 6 Classical Mathematics and Geometry

Classical problems make the connection between a question and its equations unusually visible. A purchase must satisfy both a budget and a count. A triangle must satisfy its side relationships as well as its area or perimeter. In this chapter we formulate five problems, solve them with Patodia Equation Solver and verify the answers by independent mathematics.

The examples also develop a distinction that matters throughout this book: two reported solution vectors may describe two genuinely different objects, or merely two ways to name the same object. The fruit problem has sixteen different purchases. The first triangle problem has two different third sides. In the right-triangle examples, exchanging the perpendicular sides gives another ordered vector but the same triangle shape.

The problems are presented in the classical mathematical tradition associated with Bhaskaracharya, also known as Bhaskara II, who was born in India in 1114 CE [1]. The fruit story is expressed in modern currency, and the variable names and software notation are contemporary. Our calculations depend on the stated assumptions and can be checked directly.

Historical reference [1]. MacTutor History of Mathematics Archive, University of St Andrews, Bhaskara II (1114–1185). https://mathshistory.st-andrews.ac.uk/Biographies/Bhaskara_II/

6 1 Define the quantities before writing the equations

Begin by stating what each variable measures. A count of bundles is different from a count of individual fruits. A height is a nonnegative length, whereas a Cartesian coordinate can be negative. A side length is normally a real quantity even when a particular answer happens to be an integer.

Use a separate model for each example. Repeated names such as a, b and c are local to the problem being solved; combining the five input blocks into one model would impose unrelated conditions on the same variables. The listings use US number style. The var declaration sets a readable output order and does not supply initial guesses.

The solution values below are mathematical reference answers, established by substitution or derivation. The marked screenshot locations are intended for completed solver runs. Execution times and displayed residuals should come from those runs, rather than being inferred from the exact reference values.

6 2 One hundred fruits for one hundred dollars

A person spends exactly $100 to buy exactly 100 fruits, purchasing at least one bundle of each variety. Bananas are sold in bundles of five for $3, oranges in bundles of seven for $5, apples in bundles of nine for $7, and mangoes individually for $3 each. Find every possible purchase.

Let x, y, z and w count purchases of the four offers. Each variable is a positive integer because fractional bundles and the omission of a variety are excluded.

Variable	Offer purchased	Fruits per purchase	Cost per purchase
x	Banana bundle	5	$3
y	Orange bundle	7	$5
z	Apple bundle	9	$7
w	Single mango	1	$3

The money equation uses prices as coefficients, while the fruit-count equation uses bundle sizes:

$$3x + 5y + 7z + 3w = 100$$

$$5x + 7y + 9z + w = 100$$

Both equations are linear. The difficulty is to find all admissible integer vectors rather than a single point on the continuous solution set. The positive-integer declarations are part of the mathematical model, not a request to round a continuous answer.

Solver input

```
var x,y,z,w
pos int x,y,z,w

3*x + 5*y + 7*z + 3*w = 100
5*x + 7*y + 9*z + w = 100

// Safe bounds derived from the equations and positivity
bound x = [1,13]
bound y = [1,10]
bound z = [1,8]
bound w = [3,15]
```

The bound declarations are consequences of the equations and positivity, as shown next. They make the finite search range explicit without removing any valid purchase. The original two equations and the positive-integer declarations express the problem itself.

6 3 Reduce the fruit model to a finite enumeration

Subtract the fruit-count equation from the money equation. Dividing the result by two gives:

$$-x - y - z + w = 0, \quad w = x + y + z$$

Substituting this expression for w into either original equation reduces the problem to:

$$3x + 4y + 5z = 50$$

Since y and z are at least one, 3x cannot exceed 41, so x is at most 13. Similarly, 4y cannot exceed 42, giving y at most 10; and 5z cannot exceed 43, giving z at most 8. These are safe inclusive integer bounds.

The relation between the bundle counts also bounds w. Subtract three times x+y+z from the reduced equation:

$$3w = 50 - y - 2z$$

Therefore w is at least three and at most floor(47/3)=15. The actual solutions occupy a smaller part of this permitted interval, but the bound itself follows without knowing the answers.

We can now establish completeness by checking every y from 1 through 10 and every z from 1 through 8. For each pair, calculate:

$$x = \frac{50 - 4y - 5z}{3}, \quad w = x + y + z$$

Retain the result only when x is a positive integer, and verify both original equations. This finite check gives exactly sixteen vectors. It supplies an independent completeness argument for the reference set, even if a particular numerical run stops before finding every one of them.

For example, the vector (x,y,z,w)=(1,3,7,11) buys 5 bananas, 21 oranges, 63 apples and 11 mangoes. Its total count is 5+21+63+11=100, and its cost is 3+15+49+33=100 dollars.

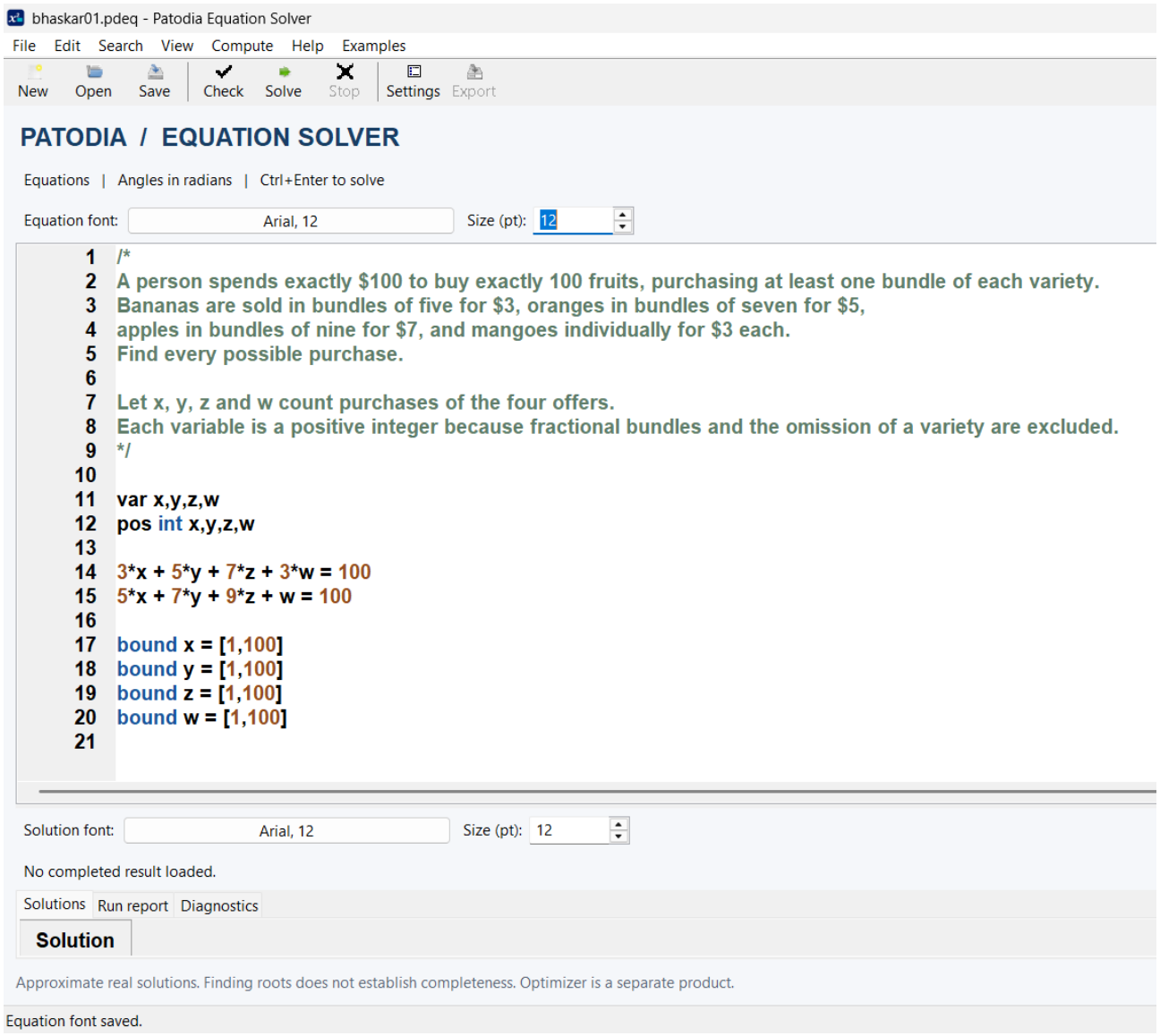
6 4 Interpret all sixteen fruit solutions

Table 5.1 lists the complete reference set. The first four numerical columns are the solver variables; the last four convert those variables into the actual quantities of fruit. The final mango count equals w because mangoes are sold individually.

Table 5.1 Bundle counts and corresponding fruit quantities

No.	x	y	z	w	Bananas 5x	Oranges 7y	Apples 9z	Mangoes w
1	1	3	7	11	5	21	63	11
2	1	8	3	12	5	56	27	12
3	2	1	8	11	10	7	72	11
4	2	6	4	12	10	42	36	12
5	3	4	5	12	15	28	45	12
6	3	9	1	13	15	63	9	13
7	4	2	6	12	20	14	54	12
8	4	7	2	13	20	49	18	13
9	5	5	3	13	25	35	27	13
10	6	3	4	13	30	21	36	13
11	7	1	5	13	35	7	45	13
12	7	6	1	14	35	42	9	14
13	8	4	2	14	40	28	18	14
14	9	2	3	14	45	14	27	14
15	11	3	1	15	55	21	9	15
16	12	1	2	15	60	7	18	15

A row order is not part of a purchase. A different run may display the same sixteen vectors in another order. Compare the variable values, not the row numbers. When converting a screenshot or export into a table for readers, retain the distinction between bundle counts and fruit quantities.


bhaskar01.pdeq - Patodia Equation Solver
File Edit Search View Compute Help Examples
New Open Save Check Solve Stop Settings Export
PATODIA / EQUATION SOLVER
Equations | Angles in radians | Ctrl+Enter to solve
Equation font: Arial, 12 Size (pt): 12
1 /*
2 A person spends exactly $100 to buy exactly 100 fruits, purchasing at least one bundle of each variety.
3 Bananas are sold in bundles of five for $3, oranges in bundles of seven for $5,
4 apples in bundles of nine for $7, and mangoes individually for $3 each.
5 Find every possible purchase.
6
7 Let x, y, z and w count purchases of the four offers.
8 Each variable is a positive integer because fractional bundles and the omission of a variety are excluded.
9 */
10
11 var x,y,z,w
12 pos int x,y,z,w
13
14 3*x + 5*y + 7*z + 3*w = 100
15 5*x + 7*y + 9*z + w = 100
16
17 bound x = [1,100]
18 bound y = [1,100]
19 bound z = [1,100]
20 bound w = [1,100]
21
Solution font: Arial, 12 Size (pt): 12
No completed result loaded.
Solutions Run report Diagnostics
Solution
Approximate real solutions. Finding roots does not establish completeness. Optimizer is a separate product.
Equation font saved.

	Solution	x	y	z	w	Maximum residual	Found seconds
▶	1	1	3	7	11	0	0.0013201
	2	1	8	3	12	0	0.0024381
	3	2	1	8	11	0	0.0032355
	4	2	6	4	12	0	0.0046464
	5	3	4	5	12	0	0.0068064
	6	3	9	1	13	0	0.0076931
	7	4	2	6	12	0	0.0084687
	8	4	7	2	13	0	0.009526
	9	5	5	3	13	0	0.0111068
	10	6	3	4	13	0	0.0124176
	11	7	1	5	13	0	0.01346
	12	7	6	1	14	0	0.0143768
	13	8	4	2	14	0	0.0153377
	14	9	2	3	14	0	0.0161332
	15	11	3	1	15	0	0.0180175
	16	12	1	2	15	0	0.0183955
*							

The requirement to buy at least one bundle of each variety is essential. Allowing zero would define a different purchase problem and change the reference set. Likewise, interpreting x as the number of bananas would make the stated coefficients incorrect.

6 5 Find a third side from two sides and an area

Two sides of a triangle have lengths sqrt(13) and sqrt(5). Its area is 4 square units. Find every possible length of the third side.

Let a and b denote the known sides, c the unknown side, and s the semiperimeter. These are real-valued geometric quantities. In particular, c must not be restricted to integers: one of the valid answers is irrational.

$$a = \sqrt{13}, \quad b = \sqrt{5}, \quad s = \frac{a + b + c}{2}$$

Heron’s formula expresses the square of the area A as:

$$A^2 = s(s - a)(s - b)(s - c)$$

Because A=4, its square is 16. The triangle inequalities ensure that the three numbers describe a triangle. The prescribed positive area excludes degenerate cases in which one side equals the sum of the other two.

Solver input

var a,b,c,s

a = sqrt(13)
b = sqrt(5)

s = (a+b+c)/2

s*(s-a)*(s-b)*(s-c) = 16

a+b >= c
a+c >= b
b+c >= a

All four equations hold simultaneously. The definitions of a, b and s are included in the model; their line order is not a procedural calculation sequence. The var declaration makes the output easier to read in the order a,b,c,s.

6 6 Explain why both third sides are valid

The reference answers for c are 4 and sqrt(20), approximately 4.47213595499958. They correspond to different triangles, not to an exchange of variable names.

Third side c	Semiperimeter s	Geometric interpretation
4	4.92080962648189	One valid triangle
sqrt(20)	5.15687760398168	A second valid triangle

An algebraic form of Heron's identity provides a complete check of the candidate side lengths:

$$16A^2 = 2a^2b^2 + 2b^2c^2 + 2c^2a^2 - a^4 - b^4 - c^4$$

Substitute a squared equal to 13, b squared equal to 5, and A squared equal to 16. After rearranging:

$$c^4 - 36c^2 + 320 = 0$$

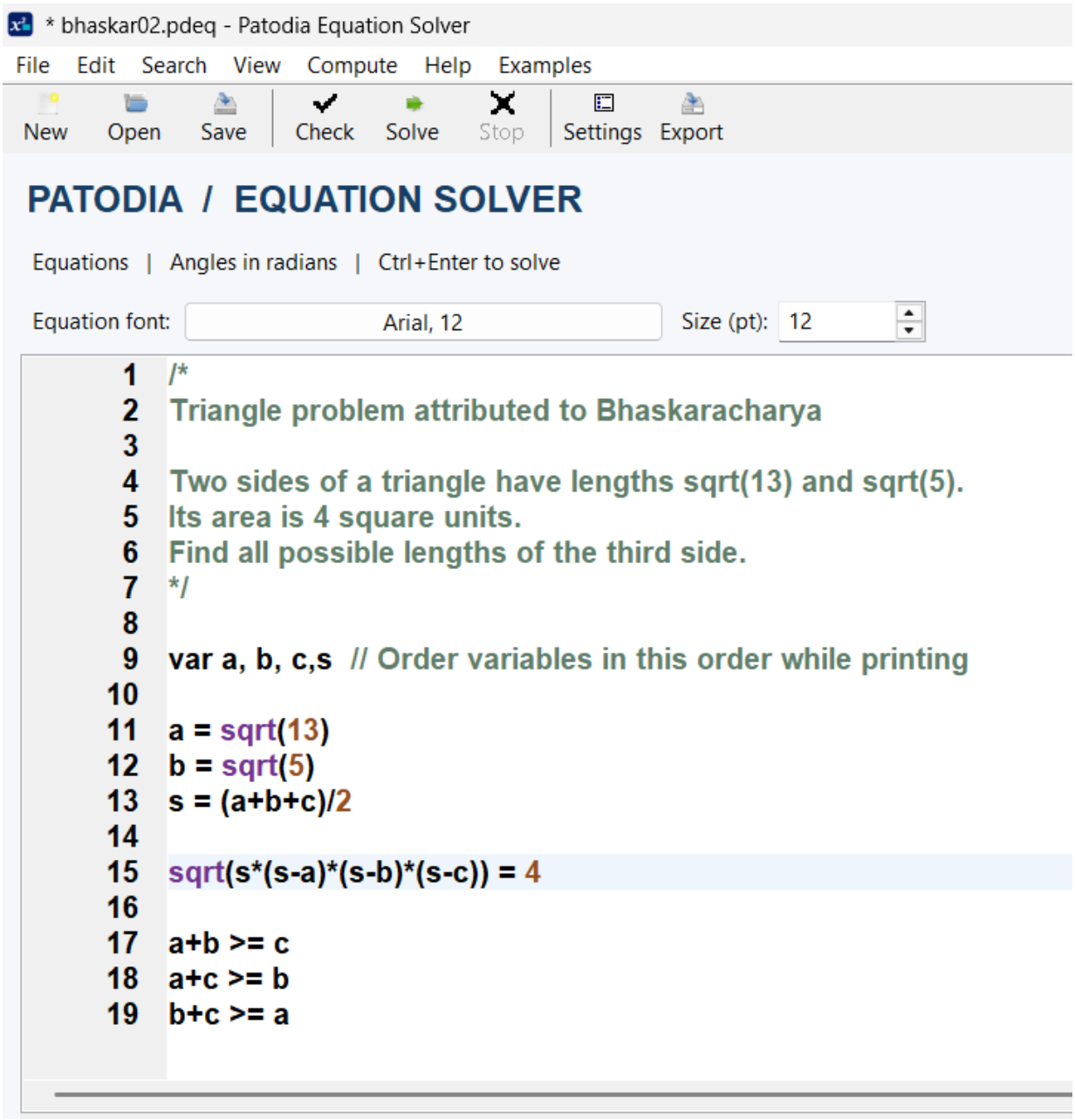
$$(c^2 - 16)(c^2 - 20) = 0$$

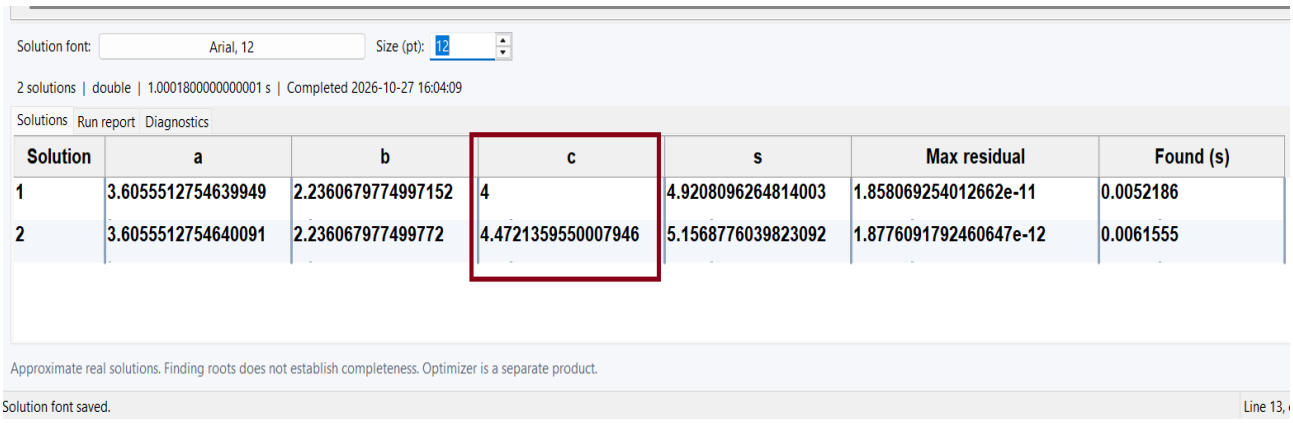
The algebraic candidates are c=4, c=-4, c=sqrt(20) and c=-sqrt(20). A side length must be positive, and the negative candidates also fail the triangle inequalities. Both positive candidates lie strictly between a-b and a+b, so both are admissible.

There is a geometric explanation as well. If C is the included angle between a and b, the area is ab*sin(C)/2. Angles C and pi-C have the same sine, so the same two side lengths can give the same area with two different included angles. The cosine rule then gives different third sides.

$$cos(C) = \frac{a^2 + b^2 - c^2}{2ab}$$

For c=4 the cosine is 1/sqrt(65); for c=sqrt(20) it is -1/sqrt(65). Thus one included angle is acute and the other obtuse. The solver should retain both geometries.





Solution	a	b	c	s	Max residual	Found (s)
1	3.6055512754639949	2.2360679774997152	4	4.9208096264814003	1.858069254012662e-11	0.0052186
2	3.6055512754640091	2.236067977499772	4.4721359550007946	5.1568776039823092	1.8776091792460647e-12	0.0061555

6 7 Compute the height relative to a specified base

A triangle has sides sqrt(10)-sqrt(5) and sqrt(6), and its base has length sqrt(18)-1. Find the perpendicular height corresponding to this base.

The phrase corresponding to this base matters: a triangle has three altitudes. Fix the notation before solving:

$$a = \sqrt{10} - \sqrt{5}, \quad b = \sqrt{6}, \quad c = \sqrt{18} - 1$$

Let h be the perpendicular height to the side c. Combine Heron's formula with A=c*h/2:

$$s(s - a)(s - b)(s - c) = \left(\frac{ch}{2}\right)^2$$

Squaring the area relation allows both signs of h algebraically. The condition h>=0 selects the geometric height. The side lengths are fixed in this problem; the equation determines their altitude rather than another triangle with different sides.

Solver input

var a,b,c,s,h

a = sqrt(10)-sqrt(5)
b = sqrt(6)
c = sqrt(18)-1
s = (a+b+c)/2

s*(s-a)*(s-b)*(s-c) = (c*h/2)^2
h >= 0

The exact height is sqrt(2)-1. Table 5.2 gives reference values for all five reported variables. The decimals are rounded for reading; the exact expressions define the reference quantities.

Variable	Reference value
a	0.926209682668590
b	2.449489742783178
c	3.242640687119285
s	3.309170056285526
h	0.414213562373095

6 8 Check the height using coordinates

An independent construction avoids relying on the same area formula twice. Place the base endpoints at (0,0) and (c,0), and place the third vertex at (t,h). The distances from that vertex to the base endpoints give:

$$t^2 + h^2 = a^2$$

$$(c - t)^2 + h^2 = b^2$$

Subtract the first equation from the second. The height terms cancel, leaving:

$$c^2 - 2ct = b^2 - a^2$$

$$t = \frac{a^2 + c^2 - b^2}{2c} = 2\sqrt{2} - 2$$

Substitution into the first distance equation gives the height directly:

$$h^2 = a^2 - t^2 = 3 - 2\sqrt{2} = \left(\sqrt{2} - 1\right)^2$$

$$h = \sqrt{2} - 1$$

Here t is approximately 0.828427124746190 and lies between zero and c, so the perpendicular foot lies inside the base segment. The area is independently recovered as:

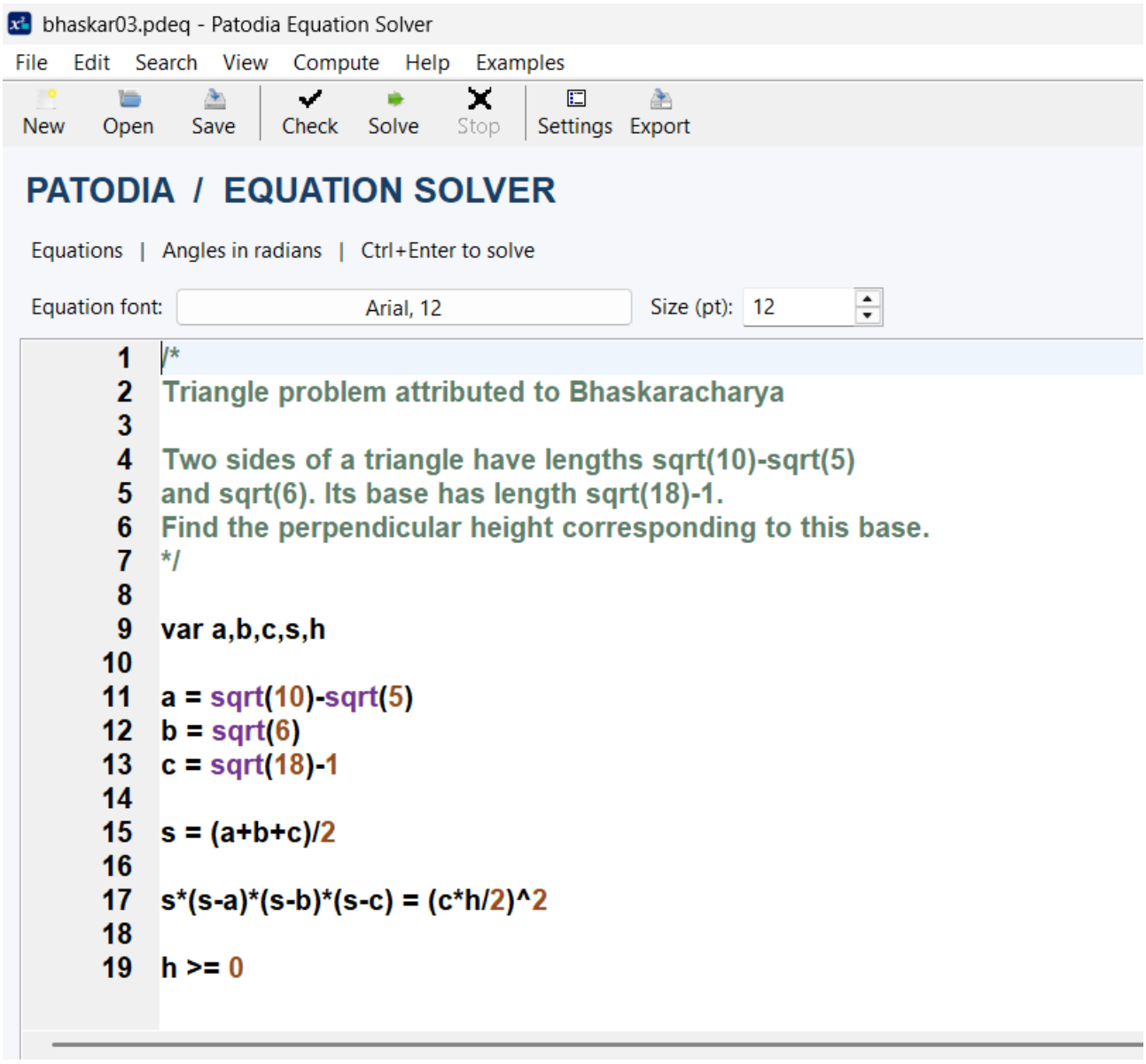
$$A = \frac{ch}{2} = \frac{7 - 4\sqrt{2}}{2} \approx 0.671572875253810$$

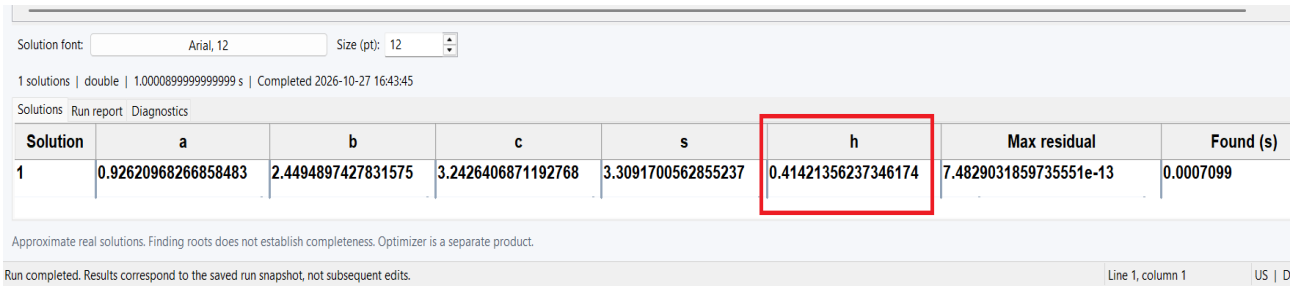
This coordinate formulation can also be entered as a separate solver model. Its variables are only t and h:

```
var t,h

t^2+h^2 = (sqrt(10)-sqrt(5))^2
(sqrt(18)-1-t)^2+h^2 = 6
h >= 0
```

Agreement between the two formulations is a useful check. They express the same geometry but use different intermediate quantities. Extended precision may make the comparison of an irrational answer easier, but additional displayed digits are not a substitute for preserving the original model and checking the residuals.





6 9 A right triangle with perimeter forty

The perimeter of a right-angled triangle is 40 units. The product of its two perpendicular sides is 120 square units. Find its side lengths.

Let x and y denote the perpendicular sides and z the hypotenuse. The right-angle assumption supplies the Pythagorean equation. The perimeter and product conditions supply the other two equations:

$$x^2 + y^2 = z^2$$

$$x + y + z = 40, \quad xy = 120$$

If the word right-angled were omitted, the Pythagorean relation would introduce an assumption not present in the question. A base and its altitude are not generally two sides of a triangle; here they are perpendicular sides because the triangle is explicitly right-angled.

Solver input

var x,y,z

x^2 + y^2 = z^2
x + y + z = 40
x*y = 120

x >= 0
y >= 0
z >= 0

The variables remain real-valued. Their integer answers are a consequence of these particular numerical data; no integer restriction is needed to obtain them. The positive product excludes a zero perpendicular side.

6 10 Verify the sides and count the geometry correctly

Set S=x+y. Using the product xy=120, expand S squared and apply the Pythagorean equation:

$$S^2 = x^2 + y^2 + 2xy = z^2 + 240$$

The perimeter gives S+z=40. Taking the difference of squares therefore gives:

$$(S - z)(S + z) = 240, \quad S - z = 6$$

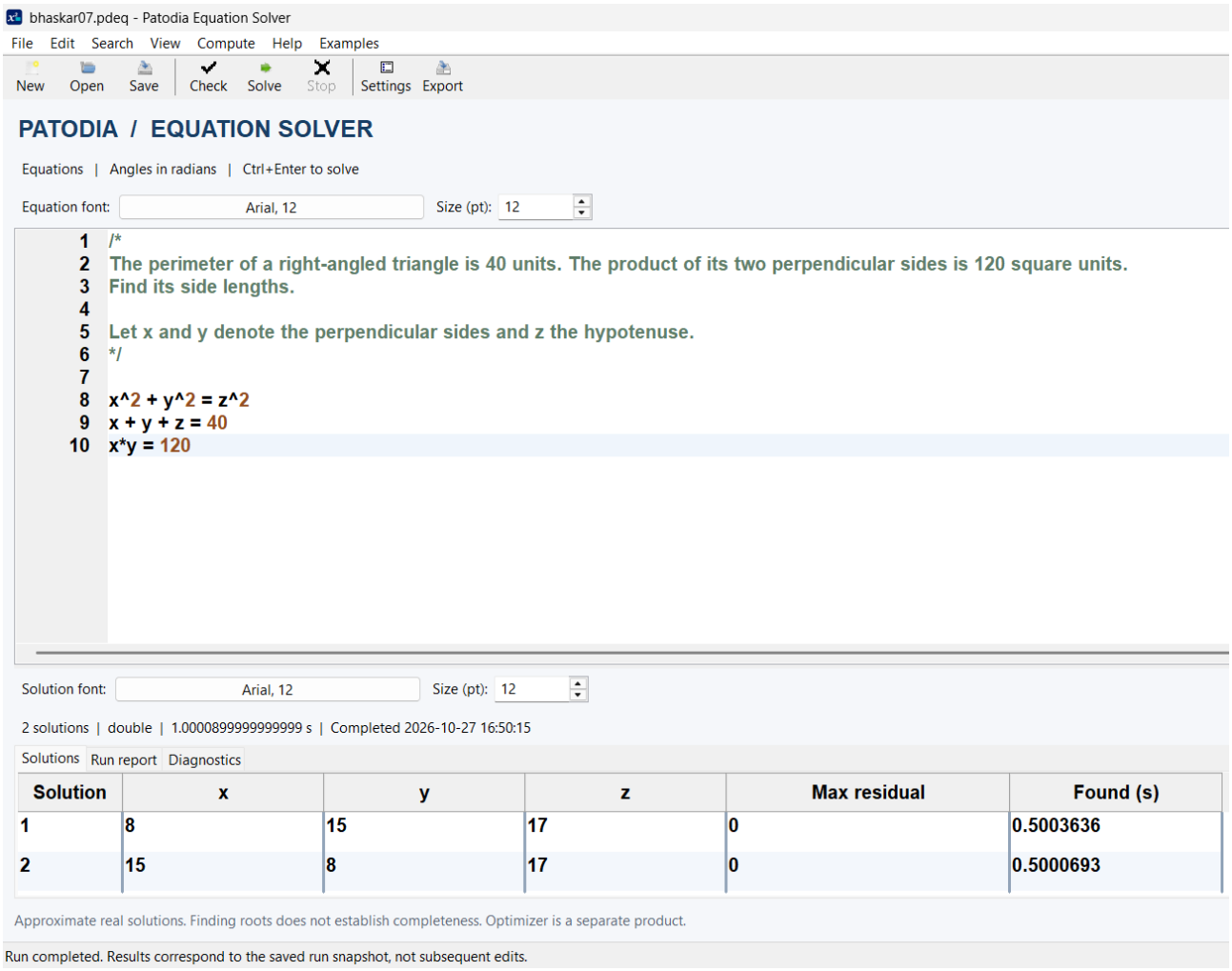
Solving S+z=40 and S-z=6 yields S=23 and z=17. The perpendicular sides are the roots of a quadratic whose sum is 23 and whose product is 120:

$$u^2 - 23u + 120 = 0$$

$$(u - 8)(u - 15) = 0$$

The two ordered solver vectors are (8,15,17) and (15,8,17). In either order, 64+225=289, the perimeter is 40, and the product of the perpendicular sides is 120. The area is half this product, so it is 60 square units.

The triangle has only one shape when exchanging the perpendicular sides is treated as a relabelling. Do not discard a solver vector as numerically wrong merely because it represents the same geometry as another vector. Instead, state which quantity is being counted: ordered vectors or distinct geometric shapes.



6 11 A right triangle with perimeter fifty-six

The perimeter of a right-angled triangle is 56 units. The product of its three side lengths is 4200 cubic units. Find its side lengths.

Again x and y are the perpendicular sides, and z is the hypotenuse. Unlike the previous problem, the specified product now includes z:

$$x^2 + y^2 = z^2$$

$$x + y + z = 56, \quad xyz = 4200$$

Solver input

var x,y,z

x^2 + y^2 = z^2
x + y + z = 56
x*y*z = 4200

x >= 0
y >= 0
z >= 0

The cubic product couples the three lengths. We can nevertheless reduce the equations to a quadratic in the hypotenuse and then recover the perpendicular sides. This is an example of using structure before treating the system as an arbitrary nonlinear search.

6 12 Reject an algebraic candidate that is not a triangle

Write S=x+y=56-z and P=xy. The Pythagorean relation gives P=(S squared-z squared)/2. Substitution simplifies the product of the perpendicular sides to:

$$P = \frac{(56 - z)^2 - z^2}{2} = 1568 - 56z = 56(28 - z)$$

Since Pz=4200, the hypotenuse must satisfy:

$$56z(28 - z) = 4200$$

$$z(28 - z) = 75$$

$$z^2 - 28z + 75 = 0$$

$$(z - 3)(z - 25) = 0$$

Both values are algebraic candidates at this stage. They must still permit real perpendicular sides and satisfy the original geometry.

Candidate z equal to three

When z=3, the remaining sides would have sum S=53 and product P=1400. Their quadratic has discriminant:

$$53^2 - 4(1400) = -2791$$

The discriminant is negative, so no real x and y have that sum and product. There is also a direct geometric contradiction: two nonnegative legs of a right triangle with hypotenuse 3 cannot have a sum of 53. This candidate must be rejected.

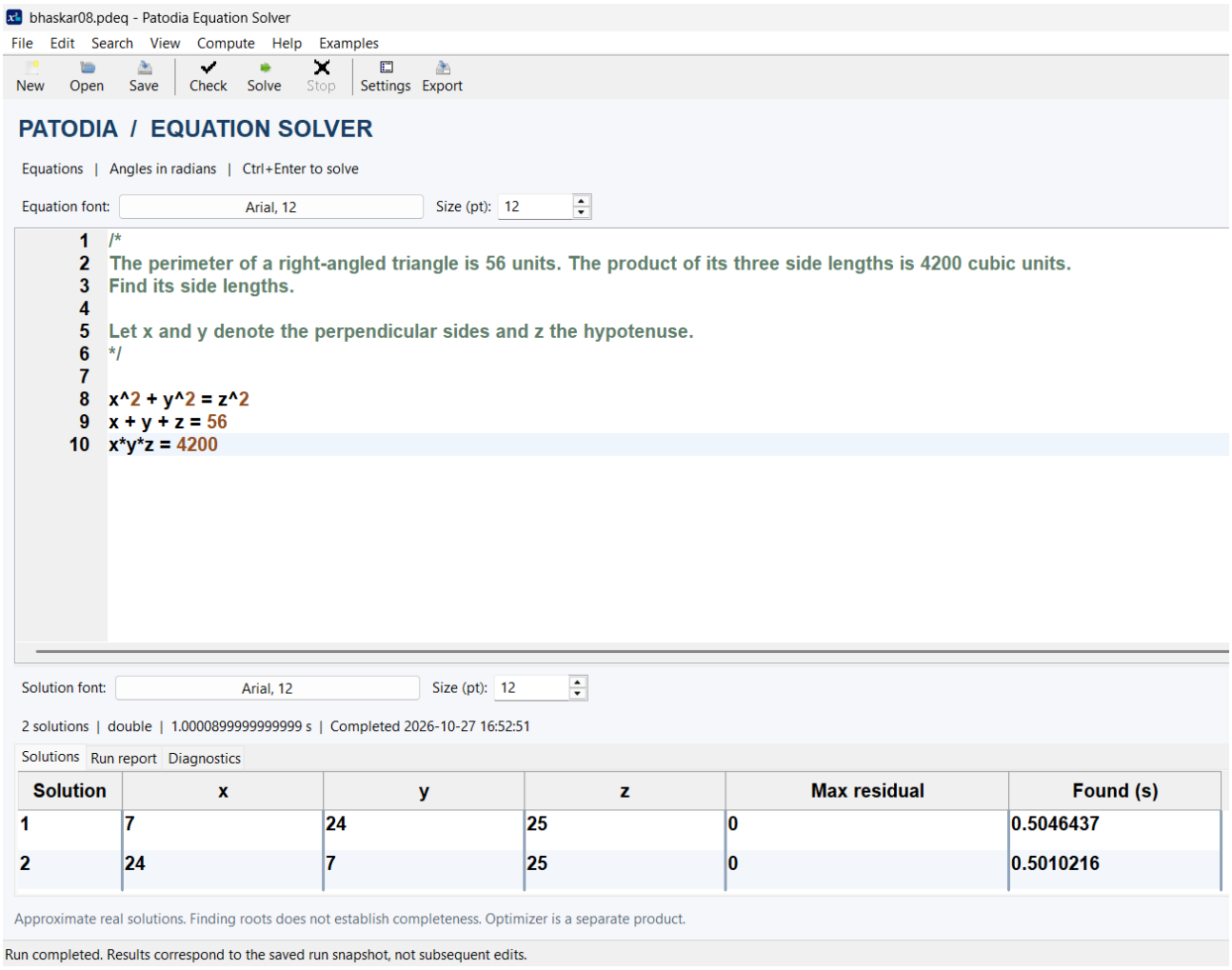
Candidate z equal to twenty five

When z=25, the sum of the perpendicular sides is 31 and their product is 168. Thus:

$$u^2 - 31u + 168 = 0$$

$$(u - 7)(u - 24) = 0$$

The valid ordered vectors are (7,24,25) and (24,7,25). Check the original system: 49+576=625, 7+24+25=56, and 7*24*25=4200. The area is 84 square units. The two ordered vectors again describe one right-triangle shape.



6 13 Compare the five kinds of answer

These examples have different solution counts for different reasons. Table 5.3 separates the mathematical output from its interpretation. The counts refer to the stated domains and the independently derived reference sets.

Problem	Reference solver vectors	Interpretation
Fruit purchases	16	Sixteen different purchases
Unknown third side	2	Two different triangles
Height to the stated base	1 positive height	One fixed triangle and one chosen altitude
Right triangle with perimeter 40	2	One shape, with two orders of the legs
Right triangle with perimeter 56	2	One shape, with two orders of the legs

A finite reference count is not the same as the number returned in an interrupted run. If a completed run returns fewer vectors than expected, compare the complete input, integer declarations, sign restrictions, bounds, solution limit and time allowance. Preserve the actual run report with the screenshot. The mathematical derivations here supply checks that do not depend on discovery order or elapsed time.

For the triangle problems, substitute the complete reported vector into the original equations. Do not mix the semiperimeter from one row with the third side from another. For irrational quantities, use the full exported values when computing numerical residuals; the shortened decimals printed in the text are for explanation.

The fruit example also shows why an underdetermined real system can have a finite integer answer set. Two equations in four real unknowns need not identify isolated points. Positivity, integrality and finite consequences of the budget change the question into a finite enumeration. Chapter 7 develops these discrete ideas further.

6 14 Practice exercises

Exercise 1. Explain why x counts banana bundles in the fruit model. Write the number of bananas and their cost in terms of x.

Exercise 2. Convert the fruit vector (x,y,z,w)=(5,5,3,13) into fruit quantities, and verify both the total count and the total cost.

Exercise 3. Derive w=x+y+z directly from the original two fruit equations. Explain why allowing zero bundle counts changes the problem.

Exercise 4. In the first triangle problem, explain why declaring c as a positive integer would miss a valid geometric answer.

Exercise 5. A triangle has two sides of lengths 3 and 4 and area 6. Find its third side and explain why the two-angle ambiguity disappears in this case.

Exercise 6. Remove h>=0 from the height model. What second algebraic height should be expected, and why is it not a second positive geometric altitude?

Exercise 7. For the perimeter-40 right triangle, calculate the area and explain the difference between its two ordered vectors and its number of shapes.

Exercise 8. For the perimeter-56 right triangle, calculate the inradius using area divided by semiperimeter.

Exercise 9. Explain precisely why z=3 must be rejected in the perimeter-56 derivation. A short inequality argument is sufficient.

Exercise 10. Suppose the word right-angled is removed from either of the last two problem statements. Is the Pythagorean equation still justified? Explain what information would be missing.

6 15 Answers and discussion

Answer 1. One banana bundle contains five bananas and costs $3. Therefore the number of bananas is 5x and their cost is 3x dollars. A declaration that x itself counts individual bananas would require different coefficients.

Answer 2. The fruit quantities are 25 bananas, 35 oranges, 27 apples and 13 mangoes. Their sum is 100. The cost is 15+25+21+39=100 dollars. The solver vector and the physical quantities are different descriptions of the same purchase.

Answer 3. Subtract the count equation from the cost equation to obtain -2x-2y-2z+2w=0. Hence w=x+y+z. Allowing zero removes the requirement to buy every variety. A separate enumeration gives 26 nonnegative vectors, compared with 16 strictly positive vectors; the extra rows are answers to the changed question.

Answer 4. The third sides are 4 and sqrt(20). Only the first is an integer. Integrality is a modelling assumption and must not be imposed on a length merely because one expected answer is a whole number.

Answer 5. The largest possible area formed by sides 3 and 4 is 3*4/2=6. Therefore the included angle is a right angle, and the third side is 5. The supplementary angle of a right angle is the same angle, so it does not produce a second triangle.

Answer 6. The area equation is unchanged when h is replaced by -h. The two algebraic values are sqrt(2)-1 and 1-sqrt(2). A geometric height is a nonnegative distance. In the coordinate picture the signs correspond to reflecting the vertex across the base, not to two positive heights.

Answer 7. The area is 8*15/2=60 square units. The ordered vectors (8,15,17) and (15,8,17) exchange the variable labels on the legs. If rotations and reflections are not counted as different shapes, there is one triangle.

Answer 8. The area is 7*24/2=84 and the semiperimeter is 56/2=28. The inradius is 84/28=3 units. This is an additional geometric quantity obtained from the verified sides.

Answer 9. If z=3 were a right-triangle hypotenuse, each nonnegative leg would be at most 3, and their sum would be at most 6. The perimeter condition instead requires x+y=53. Equivalently, the quadratic for the legs has discriminant -2791 and no real roots.

Answer 10. No. A general triangle does not satisfy x^2+y^2=z^2. Its altitude need not equal either of the other two sides. Perimeter together with an area-related or product condition does not generally specify a unique triangle; an additional appropriate relation would be needed. The right-angle assumption is the source of the third equation in these models.

6 16 Carry the modelling lessons forward

Each problem begins with a definition of the unknowns and ends with a check in the original relationships. The intermediate algebra is useful because it explains why answers exist, why other candidates fail, and what a complete result would mean. The software supplies numerical vectors; the formulation and its interpretation determine what those vectors represent.

The next chapter studies integer and mixed-integer systems in more depth. The fruit model provides a natural starting point: its variables count indivisible purchases, and its equations permit exact finite checks. Congruences and the % operator extend that reasoning to divisibility and residue conditions.

Chapter 7 Integer and Mixed Integer Systems

A count of machines, bundles or components must be a whole number. A length, mass or concentration may vary continuously. When these quantities appear in the same equations, their different domains are part of the model. A decimal answer that nearly satisfies the equations is not automatically a valid answer for a count.

This chapter develops eight worked problems, moving from factor pairs and package counts to integer triangles, Pell equations, congruences, a chain of squares, and two practical mixed integer models. Each problem includes a solver listing, reference answers and an independent explanation. The examples show how bounds, divisibility, symmetry and parity can reduce a search before numerical computation begins.

Use a separate model for each input block. The listings use US number style and explicit multiplication. The var declaration controls the output order; it does not supply starting guesses. The screenshot locations are reserved for your own runs. Solution order, elapsed time and displayed residuals may differ between runs.

7 1 Choose the domain before solving

An integer variable may take negative values, zero or positive values unless further restrictions are imposed. A positive integer starts at one. A binary variable is restricted to zero and one. Variables without an integer or binary declaration remain continuous real variables.

Quantity	Declaration and constraints	Allowed values
Signed quotient k	int k	…, −2, −1, 0, 1, 2, …
Nonnegative count n	int n and n >= 0	0, 1, 2, …
Positive count n	pos int n	1, 2, 3, …
Operating choice b	binary b	0 or 1
Nonnegative length L	L >= 0	Real values including fractions

The declaration var n,L merely names the variables and sets their display order. To make only the count integral, add pos int n and leave L continuous. Declaring every unknown as an integer would exclude legitimate fractional lengths. Conversely, omitting the declaration for n permits fractional counts that may satisfy the equations but have no physical meaning.

Zero needs particular care. A customer who must buy at least one package needs a positive count. A production line that may remain unused needs a nonnegative count. A quotient introduced to express a congruence may be zero or negative and ordinarily needs int rather than pos int.

7 2 Make the search region meaningful

A bounded integer variable has finitely many candidates. For example, the inclusive bound n = [1,20] gives twenty possible counts. A real variable with the same bound still has infinitely many possible values. The presence of bounds therefore does not make a mixed integer problem a finite enumeration of real points.

Derive bounds from the problem whenever possible. A positive factor of 12345 cannot exceed 12345. If three positive sides have perimeter 120, no side of a nondegenerate triangle can reach 60. Bounds of this kind preserve every feasible answer. An arbitrary small bound can silently remove a valid answer.

For a bounded integer system, an exhaustive discrete method or an independent enumeration can establish that the list is complete within the stated bounds. A general numerical run that stops after a time or starting-point budget has a different meaning: it has found candidates, but its stopping condition alone does not establish completeness. Inspect the run report and verify the mathematical argument accompanying each example.

For the runs in this chapter, allow at least 100 solutions so that an output limit does not hide an expected row. Start with a few seconds and increase the budget when needed. The reference counts below come from arithmetic or algebra as well as solver checks; they are not timing guarantees.

7 3 Find every positive factor pair

Find all ordered pairs of positive integers whose product is 12345. This is a useful completeness check because the answer set can be determined exactly by factorization.

$$ab = 12345$$

```
var a,b
pos int a,b
a*b=12345
bound a=[1,12345]
bound b=[1,12345]
```

Factorization gives the reference list

The integer 12345 factors as 3 × 5 × 823. The factor 823 is prime: none of the primes up to its square root divides it. A positive divisor is obtained by independently choosing whether to include each of the three prime factors, so there are $2^3 = 8$ positive divisors. Once a is chosen, b = 12345/a is fixed.

Solution	a	b
1	1	12345
2	3	4115
3	5	2469
4	15	823
5	823	15
6	2469	5
7	4115	3
8	12345	1

The table contains eight ordered vectors. If a and b label distinct quantities, both (5,2469) and (2469,5) must be retained. If the question concerns an unordered pair of factors, add a <= b; exactly four rows remain. That additional inequality changes the requested representation, not the factorization.

A table containing only five rows is incomplete for this model. In particular, a = 5 and a = 15 must not be lost simply because another numerical search found larger factors first. For every row, multiply the displayed integers and compare the result with 12345.

7 4 Count packages with a fixed total

A supplier sells one item only in packages of six and packages of nine. A buyer needs exactly 90 items and must purchase at least one package of each size. Let a count the six-item packages and b count the nine-item packages.

$$6a + 9b = 90$$

```
var a,b
pos int a,b
6*a+9*b=90
bound a=[1,15]
bound b=[1,10]
```

Reduce the equation before enumerating

Every term is divisible by three. Dividing gives 2a + 3b = 30. Reducing this equation modulo two shows that b is even. Write b = 2k; then a = 15 − 3k. Positivity of both counts gives k = 1, 2, 3 or 4.

a	b	Items in six-item packages	Items in nine-item packages
3	8	18	72
6	6	36	54
9	4	54	36
12	2	72	18

These are all four admissible purchases. The bounds in the listing are intentionally easy to justify: neither count can exceed the total divided by its package size. The positive declarations then exclude zero purchases automatically.

A small change can make the model infeasible

Suppose the required total changes from 90 to 91, with the same package sizes. Every possible total 6a + 9b is a multiple of three, whereas 91 is not. No choice of integer counts can satisfy the revised equation, even if a package type may be omitted.

```
var a,b
pos int a,b
6*a+9*b=91
bound a=[1,15]
bound b=[1,10]
```

Here a divisibility argument proves nonexistence. A solver run with no reported solution is consistent with that proof, but an empty result from a general numerical search would not by itself provide the proof. Distinguish a model that is mathematically infeasible from a search that has merely failed to find an answer.

7 5 Enumerate integer right triangles

Find all right triangles with positive integer side lengths and perimeter 120. Let a and b be the perpendicular sides and c the hypotenuse. Require a <= b so that exchanging the perpendicular sides does not produce a duplicate shape.

$$a^2 + b^2 = c^2$$

$$a + b + c = 120, \quad a \leq b$$

```
var a,b,c
pos int a,b,c
a^2+b^2=c^2
a+b+c=120
a<=b
bound a=[1,59]
bound b=[1,59]
bound c=[1,59]
```

Every side is less than half the perimeter. To see this for c, use c < a + b = 120 − c, so 2c < 120. Since c is integral, c <= 59. The other sides are smaller than c. Thus the finite bounds in the listing exclude no positive right triangle of perimeter 120.

a	b	c	Square check
20	48	52	400 + 2304 = 2704
24	45	51	576 + 2025 = 2601
30	40	50	900 + 1600 = 2500

The three rows describe three distinct triangles. Without a <= b, the model has six ordered vectors because the two legs differ in every row. The hypotenuse is not interchangeable with a leg: its role is fixed by the first equation.

An independent finite check

One direct verification is to let a run from 1 to 59 and b run from a to 59. Set c = 120 − a − b, reject c outside 1 through 59, and retain a pair only if a² + b² = c². Testing this finite set gives exactly the three rows above. This establishes completeness within the physical bounds independently of the numerical search strategy.

The familiar 3–4–5 shape appears as (30,40,50), but restricting attention to multiples of that one shape would miss (20,48,52) and (24,45,51). A known family of solutions is useful for checking a search, yet need not be the entire solution set.

7 6 Explore a Pell equation within bounds

Find all positive integer pairs satisfying the following equation with both variables at most 100:

$$x^2 - 2y^2 = 1$$

```
var x,y
pos int x,y
x^2-2*y^2=1
bound x=[1,100]
bound y=[1,100]
```

x	y	x²	2y²	Difference
3	2	9	8	1
17	12	289	288	1
99	70	9801	9800	1

These are all three pairs in the stated box. For an independent check, test each integer y from 1 to 100, form $1 + 2y^2$, and keep it only when it is a perfect square x^2 with x <= 100. Because y has been exhausted and x is positive, there is no untested branch in this box.

A bounded list can belong to an infinite family

The equation has positive solutions beyond this box. From any positive solution (x,y), form a new pair using:

$$x' = 3x + 4y, \quad y' = 2x + 3y$$

Expanding $x'^2 - 2y'^2$ gives $x^2 - 2y^2$, so the new pair also satisfies the equation. Starting at (3,2) gives (17,12), then (99,70), then (577,408). Both coordinates strictly increase, which generates infinitely many distinct positive solutions.

The pair (577,408) is not a missing result from the listing: it is outside the bounds. Increasing both upper bounds to 1000 adds that pair. The next generated pair, (3363,2378), remains outside the larger box. Also note that (1,0) satisfies the equation algebraically but violates pos int y.

This example separates two tasks: enumerating every solution in a specified region and describing an unbounded family. A finite screenshot can demonstrate the first task; it cannot display an infinite set in full.

7 7 Express congruences with the remainder operator

A congruence specifies a remainder rather than an ordinary equality. The statement x ≡ 1 (mod 5) says that x differs from one by an integer multiple of five. In the solver, write x % 5 = 1. For a positive modulus, the remainder lies between zero and one less than the modulus; for example, (−4) % 3 = 2.

Consider the five congruences developed earlier:

$$x \equiv 1 \ (mod\ 5), \quad y \equiv 2 \ (mod\ 6)$$
$$y - x \equiv 2 \ (mod\ 3), \quad x + y \equiv 5 \ (mod\ 9)$$
$$xy \equiv 6 \ (mod\ 7)$$

To obtain a finite, reproducible table, search the inclusive rectangle 1 <= x <= 315 and 1 <= y <= 126:

```
var x,y
pos int x,y
x%5=1
y%6=2
(y-x)%3=2
(x+y)%9=5
(x*y)%7=6
bound x=[1,315]
bound y=[1,126]
```

Parentheses are essential around compound numerators. For instance, (x+y) % 9 reduces the sum, whereas x+y % 9 reduces y alone before adding x. The % symbol is the remainder operator, not a percentage or division sign.

The eighteen representatives

Solution	x	y
1	6	8
2	36	104
3	51	80

Solution	x	y
4	66	2
5	81	68
6	96	116
7	111	92
8	141	62
9	156	38
10	171	86
11	186	26
12	201	74
13	216	50
14	246	20
15	261	122
16	276	44
17	291	110
18	306	32

The first row gives a quick manual check: 6 leaves remainder 1 on division by 5; 8 leaves remainder 2 on division by 6; 8 − 6 = 2; 6 + 8 = 14 leaves remainder 5 on division by 9; and 6 × 8 = 48 leaves remainder 6 on division by 7. The same five tests apply to every other row.

Why the pattern repeats

A sufficient period for x is lcm(5,3,9,7) = 315. A sufficient period for y is lcm(6,3,9,7) = 126. Increasing x by 315 or y by 126 preserves all relevant linear congruences. It also preserves the product congruence because both periods are divisible by seven.

$$x = x_0 + 315m, \quad y = y_0 + 126n$$

For each of the eighteen table rows (x_o,y_o), let m and n be any nonnegative integers. This gives eighteen families of positive solutions. Conversely, reducing any positive solution into the stated rectangle produces one of the table rows. Thus the families describe every positive solution, provided the representative table has been completely enumerated.

For example, x = 6 + 315m and y = 8 + 126n is one family. It is not the entire answer: the other seventeen representatives supply additional families. The rectangle contains 315 × 126 = 39690 candidate pairs, so an independent check can test all pairs against the five remainder conditions directly.

Use a separate integer for each quotient

The same congruences can be expressed without %. Introduce five distinct integer quotients a, b, c, d and e:

$$x = 5a + 1, \quad y = 6b + 2$$
$$y - x = 3c + 2, \quad x + y = 9d + 5$$
$$xy = 7e + 6$$

At (x,y) = (6,8), the quotient values are (a,b,c,d,e) = (1,1,0,1,6). Declaring every quotient positive would wrongly reject this solution because c is zero. For some other solutions c is negative. Reusing y as the quotient in the first congruence would instead impose a relationship on the original unknown y and change the problem.

The current % implementation requires every variable in that model to be declared integer and a positive constant integer modulus. Pure polynomial congruences may be represented through inferred periods when bounds are omitted. For an explicit finite list, supply bounds as above. More general remainder expressions, or remainder expressions combined with ordinary constraints, require finite bounds for every variable. In a model with genuine continuous variables, use integer quotient equations where appropriate instead of adding %.

7 8 Use parity to simplify a chain of squares

The following problem connects four positive integers to four shifted squares and three square products. It illustrates why modular reasoning is valuable even when the input itself contains no % operator.

$$x + 2 = p^2, \quad y + 2 = q^2, \quad z + 2 = r^2, \quad w + 2 = s^2$$
$$xy + 18 = t^2, \quad yz + 18 = u^2, \quad zw + 18 = v^2$$
$$p + q + r + s + t + u + v + 11 = 13^2$$

All eleven variables are positive integers, and p through v have upper bound 158. The sum of these seven square roots is 169 − 11 = 158, so the upper bounds are certainly safe. The first four equations then determine finite values for x, y, z and w.

```
var x,y,z,w,p,q,r,s,t,u,v
pos int x,y,z,w,p,q,r,s,t,u,v
x+2=p^2
y+2=q^2
z+2=r^2
w+2=s^2
x*y+18=t^2
y*z+18=u^2
z*w+18=v^2
p+q+r+s+t+u+v+11=13^2
bound p=[1,158]
bound q=[1,158]
bound r=[1,158]
bound s=[1,158]
bound t=[1,158]
bound u=[1,158]
bound v=[1,158]
```

Adjacent square roots have opposite parity

A square is congruent to zero or one modulo four. Consequently, x = p² − 2 is congruent to two modulo four when p is even, and to three modulo four when p is odd. The same alternatives apply to y and q.

x mod 4	y mod 4	(xy + 18) mod 4	Can this be a square?
2	2	2	No

x mod 4	y mod 4	(xy + 18) mod 4	Can this be a square?
3	3	3	No
2	3	0	Yes
3	2	0	Yes

Therefore p and q have opposite parity, and t is even. Applying the same argument to the other two product equations shows that q and r have opposite parity, r and s have opposite parity, and u and v are even. The sequence p,q,r,s must alternate even–odd–even–odd or odd–even–odd–even.

These are necessary conditions. Passing the modulo-four test does not guarantee that an integer is a square: for example, 12 is zero modulo four but is not a perfect square. The full equations and final sum still have to be checked.

The two reference solutions

Variable	Solution 1	Solution 2
x	2	119
y	23	62
z	62	23
w	119	2
p	2	11
q	5	8
r	8	5
s	11	2
t	8	86
u	38	38
v	86	8

For Solution 1, the product checks are $2 \times 23 + 18 = 64 = 8^2$, $23 \times 62 + 18 = 1444 = 38^2$, and $62 \times 119 + 18 = 7396 = 86^2$. The root sum is $2 + 5 + 8 + 11 + 8 + 38 + 86 = 158$. Adding 11 gives 169, as required.

The second solution reverses the chain. Reversing x,y,z,w also reverses p,q,r,s and exchanges t with v while leaving u unchanged. If the positions are labelled, these are two distinct ordered solutions. If a chain and its reversal are regarded as one arrangement, retain the row with p <= s when presenting the results.

A finite independent verification

Since x is positive, p cannot be one; each of p,q,r,s lies between 2 and 158. For every pair (p,q) in this range, compute $(p^2 - 2)(q^2 - 2) + 18$. Retain the pair only if this is a perfect square t^2 with t <= 158. Build the corresponding pair lists for the other links, join compatible links, and test the sum of the seven roots.

This checks every possible bounded chain and gives exactly the two table rows. It also explains the reduction in work: compatible adjacent pairs can be identified once and reused, instead of treating all eleven quantities as unrelated search coordinates. The parity argument rejects impossible pairs before the square test.

7 9 Combine an integer count with a continuous length

A manufacturer will cut n identical square panels of side length L metres. Their combined area must be 9 square metres. A quotation charges 100 currency units per panel plus a one-time tooling charge of 1000L currency units. The total quoted amount is 1900. Find every feasible count and side length.

The panel count n is a positive integer. The side length L is a continuous nonnegative real variable. Area gives $nL^2 = 9$. Dividing the price equation 100n + 1000L = 1900 by 100 gives the second relationship:

$$nL^2 = 9, \quad n + 10L = 19$$

```
var n,L
pos int n
n*L^2=9
n+10*L=19
L>=0
bound n=[1,19]
bound L=[0,1.9]
```

Because n >= 1 and L >= 0, the quotation implies n <= 19 and L <= 1.8; the displayed bound L <= 1.9 is a safe, slightly looser bound. The positive area automatically excludes L = 0.

n	L in metres	Combined area	Quotation
4	1.5	$4 \times 1.5^2 = 9$	400 + 1500 = 1900
9	1	$9 \times 1^2 = 9$	900 + 1000 = 1900

Verify the mixed integer branches algebraically

Eliminate L using L = (19 − n)/10. Substitution into the area equation gives $n(19 - n)^2 = 900$. On expansion and factorization:

$$n^3 - 38n^2 + 361n - 900 = 0$$

$$(n - 4)(n - 9)(n - 25) = 0$$

The algebraic candidates are n = 4, 9 and 25. For n = 25, the quotation requires L = −0.6, which violates the nonnegative-length condition and the stated bounds. The other two candidates give the table rows, so there are exactly two feasible plans.

This example shows why only n should be declared integral. If L were also an integer, the valid plan with four panels of side 1.5 metres would disappear. The value 1.5 is not numerical error; it is an exact fractional length required by the equations.

No objective has been specified. Both plans meet the same area and quotation. Choosing a plan on another criterion, such as cutting waste or ease of transport, requires additional information. Feasibility alone does not identify a preferred plan.

7 10 Represent a discrete choice with a binary variable

A supplier offers four-kilogram packages in two grades. Grade A costs 8 currency units per package and Grade B costs 12. A buyer chooses one grade for all n packages and adds q kilograms of loose material priced at 5 currency units per kilogram. The purchase must weigh 20.5 kilograms and cost exactly 78.5 currency units.

Let b = 0 select Grade A and b = 1 select Grade B. The package price is then 8 + 4b. The number n is a positive integer, while the loose quantity q is continuous and nonnegative.

$$4n + q = 20.5$$

$$(8 + 4b)n + 5q = 78.5, \quad b \in \{0,1\}$$

```
var n,q,b
pos int n
binary b
4*n+q=20.5
(8+4*b)*n+5*q=78.5
q>=0
bound n=[1,5]
bound q=[0,20.5]
```

At least one package must be purchased, and 4n cannot exceed 20.5. Hence n <= 5. These bounds describe the physical purchase; they are not guesses about where a numerical root might lie.

Grade	b	n	q in kg	Weight check	Cost check
A	0	2	12.5	8 + 12.5 = 20.5	16 + 62.5 = 78.5
B	1	3	8.5	12 + 8.5 = 20.5	36 + 42.5 = 78.5

Check both binary cases

Substitute q = 20.5 − 4n into the cost equation. Simplification gives (12 − 4b)n = 24. With b = 0, this gives n = 2 and q = 12.5. With b = 1, it gives n = 3 and q = 8.5. The binary variable has only these two cases, so the two rows exhaust the feasible possibilities.

If both n and b were continuous, intermediate package grades and fractional package counts could satisfy the algebra. Those choices are not offered by the supplier. If q were declared integer, both correct purchases would be excluded because their loose quantities are half-integers. Each declaration must therefore follow the meaning of its variable.

The term bn makes the model nonlinear, but the meaning is simple: a discrete choice changes the price coefficient. More elaborate applications may use binary variables for an operating mode, a material choice or an equipment option. The equations must still state which choices may occur together and how they affect the continuous quantities.

7 11 Check integrality as well as residuals

A small residual checks how closely a candidate satisfies the equations. It does not check whether the model represents the intended purchase or design. Review the declarations, signs, bounds and units separately. Then substitute the resulting values into every original equation, including the inequalities.

For an integer variable, a displayed value such as 3.000000000000001 may be a numerical representation of three. Check its distance from the nearest integer using the solver tolerance and recompute the equations with the intended integer. Do not accept 3.4 as four merely because an integer count is required. Changing a variable by rounding can destroy the balances.

Consider n + r = 5 and nr = 5 with n a positive integer and r nonnegative real. Eliminating r gives $n^2 - 5n + 5 = 0$, whose roots are $(5 \pm \sqrt{5})/2$. Neither is an integer. The continuous system has two feasible real pairs, but the mixed integer system has no solution. Rounding one coordinate does not repair this incompatibility.

Also distinguish a finite list from an unbounded family. With only $nL^2 = 9$, every permitted positive count n determines a length $L = 3/\sqrt{n}$; without an upper bound on n, there are infinitely many such mixed integer solutions. The quotation in Section 6 9 supplies the extra condition that reduces the possibilities to two plans.

A practical verification sequence is to identify each variable and its unit, enforce its domain, justify bounds, inspect the solver stopping reason, and substitute each candidate into the original model. To claim that every solution has been found, add a finite enumeration, algebraic elimination or another argument that covers the entire stated domain.

For screenshots, make the variable order readable and display enough digits to distinguish candidates. The font controls above Equations and Solutions allow independent font selection and size changes. Capture the actual output and keep execution times tied to that run; the mathematical tables in this chapter are reference values rather than reproduced software timings.

7 12 Practice problems

Work through each problem as a separate model. State the variable domains and bounds before solving, and explain whether the answer is complete in a finite region or belongs to a larger family.

Problem 1 Factor pairs

Find all positive integer solutions of ab = 360 with a <= b. How many ordered solutions would there be if the inequality were removed?

Problem 2 Optional package types

In the 90-item package problem, allow either package count to be zero. Find every nonnegative pair and describe the required declaration change.

Problem 3 A different perimeter

Find all positive integer right triangles with perimeter 60, taking a <= b. State safe finite bounds.

Problem 4 A larger Pell box

Change both upper bounds in the Pell example to 1000. List all positive solutions in the enlarged box.

Problem 5 Two simple congruences

Find all positive integers x <= 100 satisfying x % 5 = 1 and x % 3 = 0.

Problem 6 Incompatible remainders

Can an integer satisfy both x % 4 = 1 and x % 2 = 0? Explain without relying on an unsuccessful search.

Problem 7 Remove a reversed chain

From the square-chain results, retain only the row satisfying p <= s. Which ordered solution remains, and what interpretation justifies this selection?

Problem 8 A minimum panel side

Require the square panels in Section 6 9 to have L >= 1.2. Which plan remains? What happens if every variable is instead declared integral?

Problem 9 A fixed package grade

Set b = 1 in the mixed purchase problem. Find n and q. Then explain why declaring q integer makes the original purchase model infeasible.

Problem 10 A congruence beside a real variable

Model positive integer n <= 5 with n ≡ 1 (mod 3) and a real variable r = n/2. Use an integer quotient instead of %, and list the solutions.

7 13 Answers and mathematical checks

Answer 1 Factor pairs

The twelve pairs with a <= b are (1,360), (2,180), (3,120), (4,90), (5,72), (6,60), (8,45), (9,40), (10,36), (12,30), (15,24) and (18,20). Since $360 = 2^3 \times 3^2 \times 5$, it has $(3 + 1)(2 + 1)(1 + 1) = 24$ positive divisors. None is paired with itself because 360 is not a square, so removing the inequality gives 24 ordered solutions.

Answer 2 Optional package types

Use int a,b together with a >= 0 and b >= 0, or give both variables explicit bounds starting at zero. The six pairs are (0,10), (3,8), (6,6), (9,4), (12,2) and (15,0). The parameterization b = 2k and a = 15 − 3k now permits k = 0 through 5.

Answer 3 A different perimeter

The two triangles are (10,24,26) and (15,20,25). Each positive integer side can safely be bounded from 1 to 29 because every side is less than half the perimeter. Enforce a <= b and test $a^2 + b^2 = c^2$ together with a + b + c = 60.

Answer 4 A larger Pell box

The four pairs are (3,2), (17,12), (99,70) and (577,408). Check all integer y from 1 through 1000 for which $1 + 2y^2$ is a square x^2 and x <= 1000. The next pair from the displayed recurrence is (3363,2378), outside this box.

Answer 5 Two simple congruences

The answers are 6, 21, 36, 51, 66, 81 and 96. The simultaneous conditions give x = 6 + 15k. Positivity and x <= 100 permit k = 0 through 6.

Answer 6 Incompatible remainders

There is no solution. A remainder of one modulo four makes x odd, whereas a remainder of zero modulo two makes it even. The contradiction holds for positive, zero and negative integers alike under the mathematical remainder convention.

Answer 7 Remove a reversed chain

Solution 1 remains: (x,y,z,w) = (2,23,62,119), (p,q,r,s) = (2,5,8,11) and (t,u,v) = (8,38,86). The inequality selects a direction only if a chain and its reversal are to be regarded as the same arrangement. For labelled positions, keep both ordered solutions.

Answer 8 A minimum panel side

Only n = 4 and L = 1.5 meets L >= 1.2. Declaring L integer would exclude this plan. If that integer declaration were added to the original model without the minimum-side restriction, only n = 9 and L = 1 would remain. With both restrictions together there would be no feasible plan.

Answer 9 A fixed package grade

Grade B gives n = 3 and q = 8.5. More generally, the weight equation forces q = 20.5 − 4n, which is a half-integer whenever n is integral. Requiring q to be integral therefore makes the purchase model infeasible, regardless of the selected grade.

Answer 10 A congruence beside a real variable

Declare n and a new quotient k as integers, impose n − 3k = 1, and retain r = n/2 as an ordinary real equation. With 1 <= n <= 5, safe quotient bounds are 0 <= k <= 1. The solutions (n,r,k) are (1,0.5,0) and (4,2,1). Declare n positive, k as int and leave r continuous. The direct % model is unsuitable here because the current remainder implementation requires every variable in that model to be integer.

7 14 From integer structure to polynomial structure

Integer declarations, binary choices and continuous measurements play different roles in a model. Divisibility can prove that a purchase is impossible; symmetry can remove duplicate representations; modular arithmetic can expose repeating families; and a parity argument can reject square candidates before a more expensive test. Bounds turn suitable integer questions into finite tasks, while algebra provides independent checks on mixed integer branches.

The next chapter focuses on polynomial equation systems. Several techniques used here already prepare the way: factorization lists possible counts, elimination reduces coupled equations, and inequalities determine which algebraic roots describe admissible objects. These steps remain valuable whether the unknowns are integral or continuous.

7 15 Current edition workflow and scope

7 16 Exact routes added in the cumulative engine

The bounded Pell examples in this chapter remain useful. Chapters 13 to 15 now develop specialised positive Pell and general signed quadratic methods, weighted squares, exact verification and large literal bounds such as1e500. A declaration that permits signed/zero branches can select a different exact route from the positive-only shortcut. Read the actual arithmetic and integer completion flag. Preserve every decimal coordinate as a string.

7 Sweep exact integer models

A solve job can repeat an exact integer model with different coefficients. Keep large exact constants as literals in the model body.

```
workflow
for p in 50..53
    solve pair within 2
        pos int x, y
        x^2 + @p*y^2 = 817558178467407681621328608 6
    end
    report p
    report pair.status
    report count(pair.roots)
end
```

Expected counts for p = 50, 51, 52, 53 are 7, 0, 0, 6. Expect workflow_complete_with_job_failures because the empty solves are recorded as unaccepted jobs. For exact enumeration, consult the job's stopping reason and integer-completeness flags in the full JSON. The generic search_complete field describes a separate certification scope.

Chapter 8 Polynomial Equation Systems

Polynomial equations connect familiar algebra with the practical task of finding several solutions of a coupled model. A single cubic can have three real roots. Two equations may intersect at several points, meet at a repeated root, or share an entire curve of solutions. The useful question is therefore not only whether a solver finds a small residual, but what mathematical structure explains the answers.

This chapter develops ten worked problems. We begin with factorization and substitution, then examine geometric intersections, symmetric equations, higher powers and separate algebraic branches. A parameter example explains why several nearby numerical candidates can represent one repeated root. A box-design problem shows how physical inequalities reject an algebraic answer. Two larger systems extend the discussion to six variables and to a known set of 120 ordered solutions.

All variables in the main examples are continuous real variables unless stated otherwise. Some exact answers happen to be integers; that does not make the underlying models integer systems. Use each solver listing separately. The listings use US number style, explicit multiplication and var declarations to set the display order. Screenshot locations are marked for your own completed runs.

8 1 Recognize a polynomial model

A polynomial is a finite sum of constant coefficients multiplied by nonnegative integer powers of variables. Products such as x^2y^3 are permitted, and division by a nonzero constant is permitted. Division by an unknown, a negative power of an unknown, a variable inside a square root, and functions such as sin or exp generally move the expression outside the polynomial class.

The total degree of a term is the sum of its variable exponents. The total degree of a nonzero polynomial is the largest degree among its nonzero terms. Thus $x^2 + y^2$ has degree two, while $x^2y^3 + 7x$ has degree five. A system can contain equations of different degrees, and linear equations are themselves polynomial equations.

Expression	Polynomial	Total degree
$x^2 + y^2 - 9$	Yes	2
$x^2y^3 + 7x$	Yes	5
$(x - 1)(x + 2)$	Yes	2
$(x^2 + 1)/3$	Yes	2
$1/x$ or $\sqrt{x}$	No	Not applicable

To solve an equation, we may move every term to one side and set the resulting polynomial equal to zero. For example, $x^2 + y^2 = 9$ becomes $x^2 + y^2 - 9 = 0$. The original equation and this zero form have the same solutions. Factoring, substituting and eliminating variables can then reveal the branches that a numerical search should recover.

The degree alone does not determine the number of real solutions. The equation $x^2 - 1 = 0$ has two real roots, whereas $x^2 + 1 = 0$ has none. In several variables, shared factors or dependent equations may even produce infinitely many solutions. We will use exact algebra to establish the reference answer set whenever possible.

8 2 Recover three roots from a factored cubic

Find all real pairs satisfying a cubic equation in x together with a quadratic rule for y:

$$(x + 2)(x - 1)(x - 3) = 0$$

$$y = x^2 - 1$$

```
var x,y
(x+2)*(x-1)*(x-3)=0
y=x^2-1
bound x=[-3,4]
bound y=[-2,10]
```

Use each factor as a separate branch

A product of real numbers is zero exactly when at least one factor is zero. The cubic therefore gives x = −2, x = 1 or x = 3. The second equation determines one value of y for each choice. No other x can make the first product vanish.

x	y = x² − 1	Reference pair
−2	3	(−2,3)
1	0	(1,0)
3	8	(3,8)

These three pairs are all real solutions, not just three points found inside the displayed bounds. The bounds contain every branch established by factorization. The negative x value and the zero y value must be retained because neither variable was declared positive.

An expanded equation represents the same root set

$$x^3 - 2x^2 - 5x + 6 = 0$$

Expanding the three factors gives the displayed cubic. Replacing the first line of the model with this expanded equation preserves the mathematical solution set. A numerical method may follow different paths in the two forms because polynomial evaluation and derivatives are arranged differently. Compare the resulting pairs by their values rather than by discovery order or elapsed time.

The second equation is essential to the system. It supplies y, which is otherwise unrestricted by the cubic. Conversely, solving y = x² − 1 alone gives an entire parabola rather than three isolated pairs. A root of one equation becomes a system solution only when it satisfies every equation.

8 3 Combine independent polynomial choices

Two independent equations can produce a set of combinations. Consider:

$$(x - 1)(x + 2) = 0, \quad (y - 3)(y + 1) = 0$$

```
var x,y
(x-1)*(x+2)=0
(y-3)*(y+1)=0
bound x=[-3,2]
bound y=[-2,4]
```

The first equation permits x = 1 or x = −2. Independently, the second permits y = 3 or y = −1. Every allowed x can be paired with every allowed y, giving 2 × 2 = 4 ordered vectors.

Solution	x	y
1	−2	−1
2	−2	3
3	1	−1
4	1	3

The order in which the software reports these vectors is immaterial. A reference list should distinguish the four coordinate combinations even when one coordinate repeats across rows. Comparing x values alone would merge different solutions incorrectly.

An additional relationship can remove combinations

Suppose a further condition requires x + y >= 0. The pair (−2,−1) has sum −3 and is rejected. The other three pairs have sums 1, 0 and 4 and remain valid. The boundary pair (1,−1) is included because the inequality is non-strict.

This simple construction provides a useful solver check: the unfiltered model has four mathematical solutions, and the filtered set has three. It also suggests how to build examples with known answer counts. Three independent equations $x^2 = 1$, $y^2 = 4$ and $z^2 = 9$ give $2 \times 2 \times 2 = 8$ sign combinations. A coupling equation can then select a subset.

8 4 Find four intersections of a circle and a parabola

Find every real intersection of the circle of radius three centred at the origin and the parabola $y = x^2 - 4$:

$$x^2 + y^2 = 9, \quad y = x^2 - 4$$

```
var x,y
x^2+y^2=9
y=x^2-4
bound x=[-3,3]
bound y=[-3,3]
```

Substitute and reduce the even polynomial

Substitute $y = x^2 - 4$ into the circle equation. This gives $x^2 + (x^2 - 4)^2 = 9$, or $x^4 - 7x^2 + 7 = 0$. Only even powers of x occur, so set $u = x^2$. The resulting quadratic is:

$$u^2 - 7u + 7 = 0$$

$$u = (7 \pm \sqrt{21})/2$$

Both values of u are positive. Each therefore gives two real values of x, one positive and one negative. For each branch, recover $y = u - 4$. There are four intersections in total.

x approximately	y approximately	Branch of x^2
-2.406509474	1.791287847	(7 + √21)/2
-1.099414459	-2.791287847	(7 − √21)/2
1.099414459	-2.791287847	(7 − √21)/2
2.406509474	1.791287847	(7 + √21)/2

The exact coordinates are x = ±√((7 + √21)/2), y = (−1 + √21)/2, and x = ±√((7 − √21)/2), y = (−1 − √21)/2. Each sign belongs to the corresponding y branch. Mixing an x value from one branch with the other y value generally fails the equations.

The circle guarantees −3 <= x <= 3 and −3 <= y <= 3, so the search box is justified by the geometry. The two signs of x represent reflection in the vertical axis. If an application requires a nonnegative horizontal coordinate, retain the two positive-x intersections; without that condition, all four are valid.

Substitution makes the count transparent: the quadratic gives exactly two nonnegative square values, and each positive square value gives exactly two signs. It also prevents a common mistake with even equations—taking only the positive square root and losing half the solutions.

8 5 Solve a symmetric three variable system

Find all real triples whose sum, sum of pairwise products and product have the following values:

$$x + y + z = 6$$

$$xy + yz + zx = 11, \quad xyz = 6$$

```
var x,y,z
x+y+z=6
x*y+y*z+z*x=11
x*y*z=6
bound x=[0,4]
bound y=[0,4]
bound z=[0,4]
```

Construct a polynomial having the coordinates as roots

Consider the monic cubic Q(t) = (t − x)(t − y)(t − z). Expanding it expresses its coefficients through the three quantities already specified by the equations:

$$Q(t) = t^3 - (x + y + z)t^2 + (xy + yz + zx)t - xyz$$

$$Q(t) = t^3 - 6t^2 + 11t - 6 = (t - 1)(t - 2)(t - 3)$$

Consequently, the multiset of coordinates must be {1,2,3}. Each of these three values occurs once because Q has three distinct roots. The original equations do not distinguish the coordinate positions, so every permutation satisfies them.

Solution	x	y	z
1	1	2	3
2	1	3	2
3	2	1	3
4	2	3	1
5	3	1	2
6	3	2	1

There are 3! = 6 ordered solutions. These are distinct vectors even though they contain the same three numbers. If the quantities are three labelled measurements, the order matters. If only the unordered set is of interest, present the canonical row (1,2,3), with coordinates in increasing order.

The bounds zero through four contain every root of the derived cubic. The coefficient argument proves that there are no additional real triples outside that box either. Notice that no integer declaration was used: the equations themselves force these particular real values.

Repeated coordinates change the permutation count

A vector with repeated entries has fewer distinct permutations. For example, (2,2,5) has three, namely (2,2,5), (2,5,2) and (5,2,2). Counting 3! without accounting for the repeated value would double-count. Symmetry supplies a way to organize solutions; it does not guarantee that every symmetric system has factorially many distinct isolated roots.

8 6 Reduce a quartic without losing signs

The next system combines an even quartic in x with a quadratic equation for y:

$$x^4 - 5x^2 + 4 = 0, \quad y^2 = x + 3$$

```
var x,y
x^4-5*x^2+4=0
y^2=x+3
bound x=[-3,3]
bound y=[-3,3]
```

Set u = x². The quartic becomes u² − 5u + 4 = (u − 1)(u − 4) = 0. Hence x² is one or four, giving x = −2, −1, 1 or 2. For each x, the quantity x + 3 is positive, so the second equation gives two signs of y.

x	y exactly	y approximately
−2	−1	−1.000000000
−2	1	1.000000000
−1	−√2	−1.414213562
−1	√2	1.414213562
1	−2	−2.000000000
1	2	2.000000000
2	−√5	−2.236067977
2	√5	2.236067977

The system has eight real solutions: four choices for x and two for y for each choice. The bounds −3 through 3 contain all eight. Since the first equation admits only those four x values, the table is complete over the real plane.

Taking y = √(x + 3) would produce only the positive branch. That is a different equation from y² = x + 3 unless a nonnegative-y condition is also intended. Likewise, replacing x² = u with x = √u loses negative x values. Treat each square-root recovery as a branch decision rather than a formatting change.

If a reduced equation produced a negative u, it would have to be rejected because u = x² for real x. A zero u would give only one value x = 0 rather than two different signs. These distinctions matter when counting the solutions of higher-degree systems.

8 7 Preserve the branch that a division would remove

Find all real solutions of the following pair of polynomial equations:

$$x(y - 1) = 0, \quad x^2 + y^2 = 5$$

```
var x,y
x*(y-1)=0
x^2+y^2=5
bound x=[-3,3]
bound y=[-3,3]
```

Solve both zero factor cases

The first equation says x = 0 or y = 1. On the x = 0 branch, the circle gives y² = 5, hence y = ±√5. On the y = 1 branch, it gives x² = 4, hence x = ±2. These branches do not overlap at a solution because (0,1) does not lie on the circle.

Branch	x	y
x = 0	0	−√5
x = 0	0	√5
y = 1	−2	1
y = 1	2	1

Dividing the first equation by x would leave y − 1 = 0. That step is valid only under the additional assumption x ≠ 0, so it removes both solutions with x = 0. To divide safely, first solve the zero-divisor branch separately, then solve the branch on which the divisor is nonzero.

Elimination still requires recovery of every coordinate

Squaring the first equation and using x² = 5 − y² gives the necessary relation (5 − y²)(y − 1)² = 0. It identifies the candidate y values −√5, 1 and √5. Recovering x from the original equations gives one x value for each of ±√5 and two x values for y = 1.

The repeated factor at y = 1 does not mean that the full system has a repeated solution there. It combines two different points, (−2,1) and (2,1), whose y coordinates coincide. A polynomial obtained by eliminating a variable can merge coordinate information; its root list is not automatically the final list of vectors.

8 8 Follow two roots as they merge

Consider a family of systems with a real parameter k: x + y = 6 and xy = k. The parameter is a specified constant for each run, not another unknown. Substituting y = 6 − x gives a quadratic whose discriminant determines the real answer count:

$$x^2 - 6x + k = 0, \quad \Delta = 36 - 4k$$

Run the following model first with product eight. Then run the next two variants separately to examine the double-root and no-real-root cases.

```
var x,y
x+y=6
x*y=8
bound x=[0,6]
bound y=[0,6]
```

k	Discriminant	Real solution pairs
8	4	(2,4) and (4,2)
9	0	(3,3) only
10	−4	None

The repeated root case

```
var x,y
x+y=6
x*y=9
bound x=[0,6]
bound y=[0,6]
```

For k = 9, the reduced equation is $(x - 3)^2 = 0$. There is one distinct mathematical pair, (3,3); the repeated factor describes multiplicity rather than two different coordinate vectors. The Jacobian of the original two equations has rows (1,1) and (y,x). Its determinant is x − y, which is zero at (3,3). The two equations meet in a singular configuration.

This is also a useful accuracy lesson. A nearby point $x = 3 + \delta$, $y = 3 - \delta$ satisfies the sum equation exactly, while $xy - 9 = -\delta^2$. With $\delta = 0.00001$, the absolute product residual is only 10^{-10} even though each coordinate differs from three by 10^{-5}.

A numerical run may therefore return several very close candidates near (3,3). They should not be interpreted as several proven exact roots. Preserve the exported digits, compare the coordinate distances, inspect the residuals and use the exact reduction to identify the one mathematical solution. Tighter tolerance or extended precision may help distinguish small errors, but a small residual alone is not a coordinate-error bound at a repeated root.

The no real root case

```
var x,y
x+y=6
x*y=10
bound x=[0,6]
bound y=[0,6]
```

For k = 10, the quadratic discriminant is negative. No real pair exists anywhere, including outside the displayed box. A solver that returns no candidate is consistent with this algebraic proof. The proof comes from the discriminant, not from the amount of time spent searching.

The same family makes a practical parameter study. For k below nine there are two distinct real roots of the quadratic; at nine they merge; above nine they are non-real. If additional bounds or sign conditions are imposed, those must still be checked before calling the resulting pairs feasible.

8 9 Find two dimensions for an open box

A square sheet has side length 10 units. Equal squares of side x are cut from its four corners, and the sides are folded up to make an open box. Find all cuts that give volume 64 cubic units. Let L be the side length of the square base; the height is x.

$$L = 10 - 2x, \quad xL^2 = 64$$

```
var x,L
L=10-2*x
x*L^2=64
x>=0
L>=0
bound x=[0,5]
bound L=[0,10]
```

A physical box needs positive height and positive base length. The listing uses nonnegative constraints and inclusive bounds; the positive volume automatically excludes the degenerate endpoints x = 0 and L = 0. The geometry also implies 0 <= x <= 5 and 0 <= L <= 10.

Separate algebraic roots from physical boxes

Eliminate L from the volume equation. After dividing by the nonzero constant four, the cubic becomes x(5 − x)² = 16. Expanding and factoring gives:

$$x^3 - 10x^2 + 25x - 16 = (x - 1)(x^2 - 9x + 16) = 0$$

The three real algebraic cuts are x = 1 and x = (9 ± √17)/2. Recovering L = 10 − 2x determines which candidates form boxes.

x approximately	L approximately	Interpretation
1.000000000	8.000000000	Valid box
2.438447187	5.123105626	Valid box
6.561552813	−3.123105626	Reject negative base length

The two physical solutions are exactly (x,L) = (1,8) and ((9 − √17)/2, 1 + √17). Their heights and base sizes differ, but both give volume 64. The third algebraic solution is not a numerical failure; it is a real root of the polynomial that violates the geometry.

The model asks for feasible dimensions, not the maximum possible volume. Both boxes satisfy the specified requirement. An optimization question would require an objective and a separate analysis; it should not be inferred merely because two designs are available.

8 10 Close a six variable monomial cycle

A larger polynomial system can sometimes have a short exact proof. Consider six variables connected by the following cycle:

```
var x1,x2,x3,x4,x5,x6
x1^2*x2=1
x2^2*x3=1
x3^2*x4=1
x4^2*x5=1
x5^2*x6=1
x6^2*x1=1
bound x1=[0.1,2]
bound x2=[0.1,2]
bound x3=[0.1,2]
bound x4=[0.1,2]
bound x5=[0.1,2]
bound x6=[0.1,2]
```

Each equation has degree three, but the product pattern is more informative than the degree. No variable can be zero because every equation has right-hand side one. Each next variable equals the reciprocal of the square of its predecessor. Because squares of nonzero real numbers are positive, every variable is positive.

Express the entire cycle through the first coordinate

Successive substitution gives $x_2 = x_1^{-2}$, $x_3 = x_1^{4}$, $x_4 = x_1^{-8}$, $x_5 = x_1^{16}$ and $x_6 = x_1^{-32}$. The last equation, $x_6^2 x_1 = 1$, then becomes $x_1^{-63} = 1$. For a positive real x_1 this forces $x_1 = 1$, and every other coordinate follows as one.

x1	x2	x3	x4	x5	x6
1	1	1	1	1	1

Thus the system has exactly one real solution, not merely one solution in the displayed box. The bounds 0.1 through 2 are convenient search limits containing the already established answer. The positivity and cycle argument rule out any additional real solution beyond them.

Negative powers appeared only during the mathematical derivation, where nonzero variables had already been established. The solver input remains polynomial: it uses nonnegative integer exponents and products throughout. This distinction prevents a valid algebraic explanation from being confused with the original model class.

Substituting the six ones makes every equation exact. This particular exact substitution does not imply that floating-point arithmetic will evaluate arbitrary polynomial roots exactly. Irrational answers in the earlier examples still require numerical approximation and residual checks.

8 11 Use symmetry to understand 120 solutions

The final worked example prescribes five power sums of five real coordinates. It is large enough to demonstrate why a numerical search can return a valid but incomplete subset of a known solution set.

```
var x1,x2,x3,x4,x5
x1+x2+x3+x4+x5=0
x1^2+x2^2+x3^2+x4^2+x5^2=5/3
x1^3+x2^3+x3^3+x4^3+x5^3=0
x1^4+x2^4+x3^4+x4^4+x5^4=1
x1^5+x2^5+x3^5+x4^5+x5^5=0
bound x1=[-1,1]
bound x2=[-1,1]
bound x3=[-1,1]
bound x4=[-1,1]
bound x5=[-1,1]
```

Define p_k as the sum of the kth powers of the five coordinates. The equations prescribe $p_1 = 0$, $p_2 = 5/3$, $p_3 = 0$, $p_4 = 1$ and $p_5 = 0$. Because every equation is unchanged by permuting coordinate positions, any solution generates further solutions by permutation.

Recover the five coordinate values

Let e_1 through e_5 be the elementary symmetric sums: e_1 is the sum of the coordinates, e_2 the sum of all pairwise products, and e_5 their product. The coefficients of $Q(t) = (t - x_1)(t - x_2)(t - x_3)(t - x_4)(t - x_5)$ are these sums with alternating signs.

The relations between power sums and elementary symmetric sums give $e_1 = 0$, $e_2 = -p_2/2 = -5/6$ and $e_3 = 0$. The fourth relation is $p_4 + e_2 p_2 + 4e_4 = 0$, so $e_4 = 7/72$. The fifth gives $e_5 = 0$. Hence:

$$Q(t) = t^5 - (5/6)t^3 + (7/72)t$$

$$72Q(t) = t(72t^4 - 60t^2 + 7)$$

One root is zero. For the other four roots set u = t², which gives 72u² − 60u + 7 = 0 and therefore u = (5 ± √11)/12. Both values of u are positive. Define:

$$a = \sqrt{((5 - \sqrt{11})/12)}, \quad b = \sqrt{((5 + \sqrt{11})/12)}$$

Root	Approximate value
−b	-0.832497487001
−a	-0.374541409554
0	0.000000000000
a	0.374541409554
b	0.832497487001

The five roots are distinct and all lie between −1 and 1. The multiset of coordinates of any solution is therefore {−b,−a,0,a,b}. Conversely, these values have the required power sums: opposite signs cancel the odd powers, 2a² + 2b² = 5/3, and 2a⁴ + 2b⁴ = 1.

Count vectors rather than only coordinate values

Every permutation is an ordered solution. Since all five entries are distinct, there are 5! = 120 such vectors. This count follows from the polynomial coefficient argument and direct substitution; it does not depend on a particular search run finding all 120.

For example, (−b,−a,0,a,b) and (0,b,−a,−b,a) are different ordered vectors that satisfy the same symmetric equations. Presenting just the five individual root values would describe the coordinate set, but would not enumerate the 120 assignments to the five named variables.

Before running this example, set the maximum solution count to at least 200. A smaller cap can stop the search before all 120 are returned. A time or starting-point limit may also leave a subset even when every reported row is valid. Compare the number of distinct recovered permutations with the mathematical reference count, and record the actual search settings.

If only one unordered configuration is needed, present the sorted vector (−b,−a,0,a,b). If coordinates label five different positions, keep all permutations. Symmetry handling is an interpretation decision: it must match what the variable names represent.

8 12 Distinguish isolated roots from a shared curve

Having as many equations as variables does not guarantee finitely many solutions. The equations x² − y² = 0 and 2x² − 2y² = 0 repeat the same condition. Their common solution set is the union of the lines y = x and y = −x. Even inside a finite square, these lines contain infinitely many points.

A finite table of candidates cannot enumerate that entire set. To obtain isolated points, add an independent restriction such as y = 1; the two surviving pairs are (1,1) and (−1,1). Before interpreting a large numerical output list as many isolated roots, check whether equations share a factor or merely repeat one another.

Algebraic transformations can also change the answer set if their conditions are omitted. Multiplying x − 1 = 0 by x + 2 introduces the additional solution x = −2. Dividing by an expression can remove its zero branch, as Section 7 7 showed. Squaring an equation can introduce candidates that require checking in the original form.

For example, √x = x − 2 implies x = (x − 2)² after squaring. The resulting polynomial has candidates x = 1 and x = 4, but only four satisfies the original square-root equation. The omitted sign condition x − 2 >= 0 rejects one. Solving the polynomial transformation is useful only when the original domain and signs are restored.

8 13 Verify polynomial solutions numerically

For each reported vector, evaluate every original equation, including the inequalities. Compare coordinate values at full exported precision rather than judging only rounded screenshots. Exact factorization or a complete elimination argument provides a reference answer set; numerical residuals check the candidates against that set.

Near a simple root, a small residual often accompanies a small coordinate error when the local derivatives are well behaved. Near a repeated root or singular system, the connection can be much weaker, as the δ² residual in Section 7 8 demonstrates. Do not interpret every pair of slightly different decimal vectors as two distinct mathematical roots.

Residual scale also matters. Multiplying an equation by one million preserves its exact zero set but changes its absolute residual by the same factor. A tolerance comparison should therefore account for equation scaling. When comparing factored and expanded forms, preserve the model actually used and check the original equations as well.

Bounds describe the intended region; they do not generally prove that no root exists elsewhere. In this chapter the global counts come from algebra: factorization, branch analysis or the polynomial whose roots must be the coordinate values. For the 120-vector example, discovering fewer rows remains partial even if every residual is excellent.

Use the font controls above Equations and Solutions when preparing screenshots. Keep variable order, signs and enough digits visible. The tables in this chapter contain mathematical reference values, not invented screenshots, timings or precision measurements. The repeated-root and moment examples should be labelled according to what the actual run returns.

8 14 Practice problems

Treat each question as a separate model. State the domains, justify the bounds and explain which algebraic argument supports the solution count.

Problem 1 Restrict the cubic

In Section 7 2, add x >= 0. Which solution pairs remain? Does a zero y value remain admissible?

Problem 2 Three independent choices

Find every real solution of x² = 1, y² = 4 and z² = 9. Explain the total without listing the same sign combination twice.

Problem 3 Select one side of the picture

In the circle-and-parabola problem, retain only x >= 0. Give the two exact pairs and explain why neither has x = 0.

Problem 4 A repeated coordinate

Solve x + y + z = 9, xy + yz + zx = 24 and xyz = 20. How many distinct permutations occur?

Problem 5 Another even quartic

Solve x⁴ − 13x² + 36 = 0 together with y = x². Find all real pairs.

Problem 6 A zero branch with a sign restriction

In Section 7 7, require x >= 0. Which three pairs remain? Which of them would division by x remove?

Problem 7 Change the product parameter

Solve x + y = 6 and xy = 5. Explain why no real pair can satisfy the same sum when the product exceeds nine.

Problem 8 Check the rejected box root

For the open-box problem, compute the base length at x = (9 + √17)/2. Why must this real algebraic root be rejected?

Problem 9 A dependent pair of equations

Inside −2 <= x <= 2 and −2 <= y <= 2, how many solutions satisfy $x^2 - y^2 = 0$ and $2x^2 - 2y^2 = 0$? What changes if y = 1 is also imposed?

Problem 10 Build a second moment benchmark

Take the five distinct coordinate values −2, −1, 0, 1 and 2. Compute their first five power sums, write the corresponding symmetric system, and determine its number of ordered solutions.

8 15 Answers and mathematical checks

Answer 1 Restrict the cubic

The pairs (1,0) and (3,8) remain. The pair (−2,3) violates x >= 0. A zero y value is permitted because y remains a continuous real variable without a positivity condition.

Answer 2 Three independent choices

The eight vectors are (±1,±2,±3), with the three signs chosen independently. Each coordinate has exactly two choices, so there are $2^3 = 8$ ordered solutions. Bounds x in [−1,1], y in [−2,2] and z in [−3,3] contain every one.

Answer 3 Select one side of the picture

The two pairs are (√((7 − √21)/2),(−1 − √21)/2) and (√((7 + √21)/2),(−1 + √21)/2). Both squared x values are positive, so x is nonzero. They are the positive-x members of the two reflected pairs.

Answer 4 A repeated coordinate

The cubic whose roots are the coordinates is $t^3 - 9t^2 + 24t - 20 = (t - 2)^2(t - 5)$. Thus the multiset is {2,2,5}. There are three distinct vectors: (2,2,5), (2,5,2) and (5,2,2). The repeated entry reduces the permutation count from six to three.

Answer 5 Another even quartic

Set $u = x^2$. The reduced polynomial is $u^2 - 13u + 36 = (u - 4)(u - 9)$. Hence x = ±2 or ±3. The four pairs are (−3,9), (−2,4), (2,4) and (3,9). Since y is determined by x^2 rather than by y^2, there is no independent sign choice for y.

Answer 6 A zero branch with a sign restriction

The remaining pairs are (0,−√5), (0,√5) and (2,1). The pair (−2,1) violates the new condition. Division by x would still remove both zero-x pairs, even though they meet x >= 0.

Answer 7 Change the product parameter

The quadratic is $x^2 - 6x + 5 = (x - 1)(x - 5)$, so the pairs are (1,5) and (5,1). In general the discriminant is $36 - 4k$. It is negative for $k > 9$, proving the absence of real solutions regardless of a numerical search.

Answer 8 Check the rejected box root

The corresponding base length is $L = 10 - (9 + \sqrt{17}) = 1 - \sqrt{17}$, approximately −3.123105626. A physical base length must be positive. The volume expression uses L^2 and therefore does not by itself reject a negative L; the geometric constraints do.

Answer 9 A dependent pair of equations

There are infinitely many solutions: the two diagonal line segments $y = x$ and $y = -x$ inside the square. The second equation is exactly twice the first and supplies no new restriction. Adding $y = 1$ leaves (1,1) and (−1,1).

Answer 10 Build a second moment benchmark

The power sums are $p_1 = 0$, $p_2 = 10$, $p_3 = 0$, $p_4 = 34$ and $p_5 = 0$. Set the sum of the kth powers of five unknown coordinates equal to these five values. Their monic polynomial is $t(t^2 - 1)(t^2 - 4) = t^5 - 5t^3 + 4t$, which fixes the coordinate multiset as {−2,−1,0,1,2}. All five entries are distinct, so every permutation gives one of exactly 120 ordered real solutions. Bounds [−2,2] for each coordinate contain the entire set.

8 16 From algebraic branches to transcendental systems

Polynomial structure can make a multiple-solution problem understandable before the search begins. Factorization identifies branches, substitution reduces variables, symmetry explains permutations, and physical inequalities select admissible designs. Repeated roots and shared factors show why a numerical candidate list must still be interpreted mathematically.

The next chapter considers transcendental systems containing exponential, logarithmic and trigonometric functions. Many habits developed here still apply: specify the domain, preserve every valid branch, check the original equations and separate a successful numerical search from a proof of completeness. New domain restrictions and periodic behaviour will make those habits even more important.

Chapter 9 Transcendental Equation Systems

Exponential, logarithmic and trigonometric functions describe growth, attenuation, temperature change, angles and oscillation. When these functions appear in coupled equations, the possible answers may depend strongly on domains and periodic branches. A logarithm needs a positive argument; a tangent has poles; and one trigonometric relationship can describe infinitely many angles unless a region is specified.

This chapter develops ten worked problems, from a single exponential intersection to logarithmic balances, periodic systems, cooling calibration, hyperbolic functions and amplitude–phase recovery. A tangent-and-circle example has four real intersections and requires explicit treatment of its poles. A three-variable cycle shows how a larger transcendental system can inherit a two-solution structure from a simpler equation.

Each example includes a copyable solver listing, mathematical reference values and an independent explanation. The listings use US number style and continuous real variables. The var declaration sets a readable display order. Use each input block separately, and insert your own screenshots at the marked locations. Decimal tables are rounded reference values; execution times and reported residuals should come from your actual runs.

9 1 Know the function and its real domain

A transcendental system contains functions such as exp, ln, sin or cos that are not polynomial expressions in the unknowns. Some such equations have unique solutions, some have several isolated solutions, and others repeat indefinitely. The function definitions and their domains are part of the mathematical model.

Solver notation	Meaning	Real domain or restriction
exp(x)	Natural exponential e^x	Every real x; result is positive
ln(x) or log(x)	Natural logarithm	x must be positive
log10(x)	Base-ten logarithm	x must be positive
sin(x), cos(x)	Trigonometric functions	Every real x; angles in radians
tan(x)	Tangent	Exclude $\pi/2 + k\pi$
asin(x), acos(x)	Principal inverse sine and cosine	Argument between −1 and 1
sinh(x), cosh(x)	Hyperbolic sine and cosine	Every real x

The solver uses radians. An angle of 60 degrees must therefore be entered as pi/3, not as 60. The constant pi can be used in expressions and bounds. Inverse trigonometric functions return principal values: asin gives an angle from $-\pi/2$ to $\pi/2$, while acos gives one from zero to π. These principal values are not complete lists of periodic angles.

The exponential is always positive, so exp(x) = y automatically implies y > 0. A logarithm equation should be accompanied by bounds consistent with its argument being positive. Do not replace ln(x) with ln(abs(x)) merely to avoid an invalid evaluation: that changes the domain and can change the intended model.

Restrictions also matter during transformations. The identity ln(xy) = ln(x) + ln(y) is valid for positive real x and y. The equation $\ln(x^2) = 2\ln(x)$ is valid only for positive x, even though the expression $\ln(x^2)$ itself is defined for negative nonzero x as well. Preserve the original domain when rewriting a system.

9 2 Specify the region and preserve every branch

A periodic equation has no useful finite root count until the interval is stated. For example, sin(x) = 0 has infinitely many real roots x = kπ, but only five in the closed interval [−7,7]. A root outside a chosen region is not missing from an enumeration restricted to that region.

A pole is different from a root. Tangent is undefined at π/2 + kπ. A numerical method may encounter very large function values near such a point, but the pole cannot be accepted as a mathematical solution. Bracketing or scanning should be done within continuous branches rather than across a discontinuity.

For the examples here, permit at least 100 reported solutions and begin with a few seconds of search time. Increase the budget when needed. A search that stops after a time or starting-point budget may have found only a subset. The reference counts in this chapter are justified separately through monotonicity, convexity, algebraic reduction or explicit periodic formulas.

Keep the original equations available when comparing alternative forms. Replacing exp(x) = y by x = ln(y) preserves the real solution set when y > 0, but the two residuals have different scales. Their numerical behaviour and reported residual magnitudes need not be identical.

9 3 Intersect an exponential curve with a line

Find all real pairs satisfying an exponential relationship and a fixed sum:

$$exp(x) = y, \quad x + y = 3$$

```
var x,y
exp(x)=y
x+y=3
bound x=[0,2]
bound y=[0,3]
```

Reduce to a strictly increasing function

Substitute y = exp(x) into the sum equation. Define F(x) = exp(x) + x − 3. Its derivative is exp(x) + 1, which is positive for every real x. Thus F is strictly increasing and can have at most one real zero.

At x = 0, F is −2. At x = 2, F is exp(2) − 1, which is positive. Continuity therefore gives exactly one root between zero and two. Recovering y from the sum gives the reference pair below.

x approximately	y approximately
0.792059968431	2.207940031569

The answer is unique over the entire real plane, not just inside the displayed box, because of the strict monotonicity. Substitution checks both equations: the two coordinates add to three, and the second equals the exponential of the first. No elementary closed form is required to define or verify this root.

The bounds contain the proven root and keep the search in a modest range. They do not supply a starting estimate. A different equation, such as exp(x) = 3x, need not have the same answer count; the next exponential example will show two intersections.

9 4 Solve two logarithmic balance equations

Find all positive real pairs with sum five and the following logarithmic balance:

$$x + y = 5, \quad ln(x) + ln(y) = ln(4)$$

```
var x,y
x+y=5
ln(x)+ln(y)=ln(4)
bound x=[0.1,5]
bound y=[0.1,5]
```

Both logarithm arguments must be positive. The lower bound 0.1 keeps the search away from zero and contains the exact solutions established below. The sum equation also makes five a safe upper bound for each positive coordinate.

Use the logarithm identity within its domain

For positive x and y, combine the logarithms to get ln(xy) = ln(4). Since the natural logarithm is one-to-one on the positive reals, this is equivalent to xy = 4. Substitute y = 5 − x to obtain:

$$x^2 - 5x + 4 = (x - 1)(x - 4) = 0$$

x	y	Product check	Logarithmic check
1	4	1 × 4 = 4	ln(1) + ln(4) = ln(4)
4	1	4 × 1 = 4	ln(4) + ln(1) = ln(4)

The two ordered pairs are (1,4) and (4,1). Both are in the logarithm domain, and the quadratic allows no additional candidates. If x and y represent labelled quantities, retain both rows. If only an unordered pair is needed, present the row with x <= y as a representative.

The transformed polynomial equations are a convenient independent check. They do not make it safe to ignore logarithm domains in other problems. In this case the candidate roots are positive; in another model a product equation might admit negative values that are invalid in the original logarithms.

9 5 Enumerate periodic sine solutions in an interval

Find all real pairs in the specified interval satisfying:

$$sin(x) = 1/2, \quad y = cos(x), \quad 0 \leq x \leq 2\pi$$

```
var x,y
sin(x)=0.5
y=cos(x)
bound x=[0,2*pi]
bound y=[-1,1]
```

Recover both angular branches

The principal value asin(1/2) is π/6. Sine has the same positive value at the supplementary angle 5π/6. These are the only two solutions between zero and 2π. The cosine equation determines y for each angle.

x exactly	y exactly	x approximately	y approximately
π/6	√3/2	0.523598775598	0.866025403784
5π/6	−√3/2	2.617993877991	-0.866025403784

Taking only the inverse-sine principal value would lose the second solution. Inverse functions select a branch; they do not generally enumerate every angle satisfying the original trigonometric equation.

Describe the family beyond the search interval

$$x = \pi/6 + 2k\pi, \quad y = \sqrt{3}/2$$

$$x = 5\pi/6 + 2k\pi, \quad y = -\sqrt{3}/2$$

Here k can be any integer. These two families describe all real solutions. Restricting x to [0,2π] selects the two rows above. Enlarging the interval to [0,4π] selects four rows. The endpoints zero, 2π and 4π are not roots because their sine is zero.

The cosine values distinguish the two branches even though the sine values agree. When a variable represents a physical phase, state whether angles separated by one full turn should count as different numerical vectors or as the same phase modulo 2π.

9 6 Find a cosine fixed point

A fixed point is a value unchanged by a function. Find x satisfying cos(x) = x, and recover a second coordinate y = sin(x).

```
var x,y
cos(x)=x
y=sin(x)
bound x=[-1,1]
bound y=[-1,1]
```

Any real solution must have −1 <= x <= 1 because cosine always lies in that range. On this interval F(x) = cos(x) − x has derivative −sin(x) − 1, which is strictly negative. The endpoint values have opposite signs: F(−1) = cos(1) + 1 is positive, and F(1) = cos(1) − 1 is negative.

x approximately	y approximately
0.739085133215	0.673612029183

There is therefore exactly one real x and one corresponding pair. A direct check uses cos(x) ≈ x and sin(x) ≈ y. The identity $x^2 + y^2 = 1$ also follows from these equations, but it is only an additional check: a point on the unit circle is not automatically a fixed point of cosine.

The fixed-point notation describes a mathematical relationship. It does not require the user to supply an initial iterate or to solve the equation manually by repeated substitution. The same model can be submitted directly to Patodia Equation Solver.

9 7 Find two branches of an exponential equation

Find all real pairs satisfying exp(x) = 3x and y = exp(−x). The second coordinate can be interpreted as an attenuation factor associated with x.

```
var x,y
exp(x)=3*x
y=exp(-x)
bound x=[0,2]
bound y=[0,1]
```

Locate the minimum before counting roots

Let F(x) = exp(x) − 3x. Its first derivative is exp(x) − 3 and its second derivative is exp(x), which is positive. The function decreases until x = ln(3), then increases. At its minimum:

$$F(ln(3)) = 3 - 3ln(3) < 0$$

Since F(0) = 1 and F(2) = exp(2) − 6 > 0, there is one root on each side of the minimum inside (0,2). The single minimum rules out any further real roots. In particular, a negative x cannot satisfy the equation because the exponential is positive while 3x is negative.

x approximately	y approximately
0.619061286736	0.538449650261
1.512134551658	0.220438937109

Both solutions have positive x and attenuation y between zero and one. At a root, exp(x) = 3x also gives y = 1/(3x), which is a convenient independent check of the second coordinate. The smaller x corresponds to the larger attenuation factor.

This example shows why positivity of the exponential does not imply uniqueness of a coupled system. The answer count depends on the other side of the equation. Here a line intersects a strictly convex curve twice; in Section 8 3, the reduced equation was strictly increasing and had one root.

9 8 Calibrate an exponential cooling model

An object is initially at 100 degrees, and the surrounding temperature is 20 degrees. Its temperature after t minutes is modelled by T(t) = 20 + 80exp(−kt), where k is a positive cooling constant measured per minute. After ten minutes the object is at 60 degrees. Find k and the time when its temperature reaches 30 degrees.

The observation at ten minutes gives 80exp(−10k) = 40. The target temperature gives 80exp(−kt) = 10. The unknowns are the real-valued constant k and the elapsed time t:

```
var k,t
80*exp(-10*k)=40
80*exp(-k*t)=10
bound k=[0,1]
bound t=[0,60]
```

Use the observation to identify the decay rate

From exp(−10k) = 1/2, take natural logarithms to obtain k = ln(2)/10. The target equation exp(−kt) = 1/8 then gives kt = ln(8) = 3ln(2), so t = 30 minutes.

Quantity	Exact value	Approximate value and unit
k	ln(2)/10	0.069314718056 per minute
t	30	30 minutes

The temperature falls halfway from the initial value to the surroundings every ten minutes. The differences above ambient are 80 initially, 40 after ten minutes, 20 after twenty minutes and 10 after thirty minutes. Thus the target temperature is 20 + 10 = 30 degrees, consistent with the equations.

This is parameter calibration and evaluation of a supplied exponential law. The solver is not numerically integrating a differential equation here. The model assumes constant ambient temperature and a constant cooling coefficient; the equations give results for those assumptions.

The stated bounds include the analytically determined rate and time. Neither k nor t should be declared integer simply because the target time happens to be exactly thirty. A different target temperature would generally produce a fractional time.

9 9 Recover both signs in a hyperbolic system

Hyperbolic functions are built from exponentials. Consider the system cosh(x) = 2 with y = sinh(x):

```
var x,y
cosh(x)=2
y=sinh(x)
bound x=[-2,2]
bound y=[-2,2]
```

By definition, cosh(x) = (exp(x) + exp(−x))/2 and sinh(x) = (exp(x) − exp(−x))/2. Let u = exp(x), which must be positive. The first equation becomes u + 1/u = 4, so:

$$u^2 - 4u + 1 = 0, \quad u = 2 \pm \sqrt{3}$$

Both values of u are positive and their product is one. Taking logarithms gives x = ±ln(2 + √3). The corresponding y values are ±√3, with the same sign as x.

x exactly	y exactly	x approximately	y approximately
−ln(2 + √3)	−√3	−1.316957896925	−1.732050807569
ln(2 + √3)	√3	1.316957896925	1.732050807569

The identity cosh²(x) − sinh²(x) = 1 supplies another check: with cosh(x) = 2, the second coordinate has square three. However, that identity alone does not choose the sign. The oddness of sinh and the evenness of cosh explain the two reflected solution pairs.

Selecting only the nonnegative inverse-hyperbolic-cosine branch would lose the negative-x solution unless a physical condition explicitly requires x >= 0. The listing uses the supported cosh and sinh functions directly and does not need an inverse-function call.

9 10 Recover amplitude and phase from two components

Suppose two measured components of a signal are both one. Write the positive amplitude as exp(x) and the phase angle as y. The real equations are:

$$exp(x)cos(y) = 1, \quad exp(x)sin(y) = 1$$

```
var x,y
exp(x)*cos(y)=1
exp(x)*sin(y)=1
bound x=[-2,2]
bound y=[-2*pi,2*pi]
```

Use the sum of squares to identify amplitude

Square the two equations and add them. Since cos²(y) + sin²(y) = 1, the result is exp(2x) = 2. Thus the amplitude is √2 and its logarithm is x = ln(2)/2. The two positive components also give cos(y) = sin(y) = 1/√2.

$$x = ln(2)/2, \quad y = \pi/4 + 2k\pi$$

For integer k, these are all phase branches. The interval [−2π,2π] contains exactly two, corresponding to k = −1 and k = 0:

x approximately	y approximately
0.346573590280	-5.497787143782

x approximately	y approximately
0.346573590280	0.785398163397

These two numerical vectors describe angles differing by a complete turn. If phase is defined only modulo 2π, they represent the same physical phase. If the model explicitly asks for all angles within the stated interval, both numerical vectors belong in the solution table.

Taking the arctangent of a ratio can lose information. The ratio of sine and cosine components does not by itself determine the quadrant, and a principal inverse angle does not include all periodic copies. In this example the signs of both components identify the first-quadrant phase before full turns are added.

9 11 Solve a tangent and circle system without accepting poles

Find every real intersection of the following transcendental relationship and the circle of radius three:

$$tan(x) = exp(y), \quad x^2 + y^2 = 9$$

```
var x,y
tan(x)=exp(y)
x^2+y^2=9
bound x=[-3,3]
bound y=[-3,3]
```

The circle forces both coordinates into [−3,3]. The tangent poles at −π/2 and π/2 lie inside this interval and are not admissible roots. Since exp(y) is positive, tangent must also be positive. Inside the x bounds, this leaves only the intervals [−3,−π/2) and (0,π/2).

The four reference intersections

x approximately	y approximately
-2.807363680962	-1.057690485358
-1.652378202070	2.503926172499
0.049766498954	-2.999587187528
1.496659042944	2.600002251764

Check every row in both original equations. The first pair is in the negative-x, negative-y region; the second has negative x and positive y. The other two have positive x, with one negative and one positive y. A negative y is valid because exp(y) remains positive.

The reference values were obtained independently by reducing to one variable and bracketing within continuous tangent branches. The rows are not inferred merely from a scatter plot or from the number of candidates returned by a timed run.

An independent argument for exactly four intersections

On a branch where tangent is positive, put h(x) = ln(tan(x)). The first equation fixes y = h(x). The circle condition becomes F(x) = x² + h(x)² − 9 = 0. This function is continuous on each of the two allowed intervals.

On the negative-x interval, F(−3) = ln²(tan(−3)) is positive. At x = −3π/4, h is zero and F = 9π²/16 − 9 is negative. As x approaches −π/2 from the left, h tends to positive infinity and F becomes positive without bound. There is therefore a zero on each side of −3π/4.

On the positive-x interval, F tends to positive infinity near both endpoints: ln(tan(x)) tends to negative infinity near zero and to positive infinity near π/2. At x = π/4, h = 0 and F = π²/16 − 9 is negative. This gives two further zeros, one on each side of π/4.

To rule out additional zeros, differentiate on either allowed interval. The identities h′ = 2/sin(2x) and cos(2x) = −tanh(h) give:

$$F''(x) = 2 + 8(1 + h(x)tanh(h(x)))/sin^2(2x) > 0$$

For every real h, the product h tanh(h) is nonnegative. Thus F is strictly convex on each interval. A strictly convex function with the sign patterns just established has exactly two zeros there. The two intervals together give exactly four real solution pairs. No root lies on a tangent pole or in a negative-tangent interval.

An equivalent formulation avoids tangent evaluations

For this particular system, multiply the tangent equation by cos(x). An equivalent listing is:

```
var x,y
sin(x)=exp(y)*cos(x)
x^2+y^2=9
bound x=[-3,3]
bound y=[-3,3]
```

This listing evaluates only sine, cosine and the exponential. If cos(x) were zero, sin(x) would be ±1, and the first equation would be impossible. Hence every solution of the transformed system has nonzero cosine and can be divided back to recover tan(x) = exp(y). The transformation introduces no solutions at the poles in this example.

The two formulations have the same exact roots, but their residuals and numerical behaviour differ. In particular, multiplying by cosine changes the scale of the first residual. Compare the candidate coordinates and substitute into the original tangent equation before making an accuracy claim.

9 12 Extend two exponential branches to three variables

Consider a cyclic system in three real unknowns:

```
var x,y,z
exp(x)=3*y
exp(y)=3*z
exp(z)=3*x
bound x=[0,2]
bound y=[0,2]
bound z=[0,2]
```

All three variables must be positive because each is one third of an exponential. Define g(t) = exp(t)/3. The equations say y = g(x), z = g(y) and x = g(z). The function g is strictly increasing.

A monotone cycle forces equal coordinates

Suppose x < y. Applying the increasing function g gives y < z; applying it again gives z < x. This contradicts x < y < z. The assumption x > y leads to the reverse contradiction. Therefore x = y, and the equations then give z = x as well.

Every solution consequently has the form (a,a,a), where exp(a) = 3a. Section 8 7 established exactly two real scalar roots of this equation. Both lie between zero and two, so the cyclic model has exactly the following two real vectors:

x approximately	y approximately	z approximately
0.619061286736	0.619061286736	0.619061286736

x approximately	y approximately	z approximately
1.512134551658	1.512134551658	1.512134551658

The three-variable system does not require all combinations of the two scalar roots. A mixed vector using the smaller value in one position and the larger value in another fails the cyclic equations. The monotonicity argument forces the three coordinates to use the same branch.

This example illustrates how structure can reduce a larger nonlinear model. A successful search is checked against two equal-coordinate reference vectors, and the proof excludes any additional unequal-coordinate solution, both within the box and outside it.

9 13 Check domains and residuals in the original equations

A candidate must satisfy every function domain before its residual is meaningful. Logarithm arguments must be positive, denominators nonzero, inverse-trigonometric arguments within their real ranges, and tangent arguments away from poles. A value that cannot be evaluated in the original model is not repaired merely by rewriting an expression.

Use each original equation to check a transformed solution. Taking logarithms, multiplying by a function, squaring an equation and selecting an inverse branch all need justification. The sine–cosine version of the tangent problem was proved equivalent because its solutions cannot have zero cosine; that reasoning should not be assumed automatically for every multiplication.

A small residual is useful evidence, but it is not a universal bound on coordinate error. The local derivatives and equation scaling matter. Exponentials can magnify small changes in one region, logarithms can be sensitive near zero, and tangent changes rapidly near a pole. Record the model form, bounds, tolerance and precision when comparing runs.

Extended precision may reduce rounding error in a supported calculation, but it does not turn a pole into a valid point, restore a discarded periodic branch, or by itself establish completeness. A claim about all solutions needs a domain argument or a procedure that covers the entire stated region. The proofs in this chapter provide reference counts independently of solver timings.

For screenshots, use the font controls above Equations and Solutions, keep the variable order fixed and show enough digits to distinguish branches. Preserve full exported values when checking residuals. The reference tables intentionally omit invented discovery times; a completed run should supply its own timing and precision information.

9 14 Practice problems

Build each question as a separate model. State the real domain and bounds, and explain which branches are included. Where an exact value is available, use it to check the numerical answer.

Problem 1 An exponential logarithmic pair

Solve exp(x) = y and ln(y) + x = 2, with y positive. Give the exact solution and a finite box containing it.

Problem 2 Change a logarithmic balance

Replace the sum in Section 8 4 by x + y = 6 and the logarithmic right-hand side by ln(8). Find all positive solution pairs.

Problem 3 Enlarge a periodic interval

In Section 8 5, change the x bounds to [0,4π]. List every x value exactly and state its corresponding y.

Problem 4 Preserve a logarithm domain

Find the real solutions of $\ln(x^2) = 0$. Would replacing the equation by $2\ln(x) = 0$ preserve every solution? Explain.

Problem 5 An exponential equation with no real root

Can $\exp(x) = 2x$ have a real solution? Use the minimum of $\exp(x) - 2x$ to justify your answer.

Problem 6 Change the cooling target

Keep the calibrated cooling constant from Section 8 8. At what time does the temperature reach 40 degrees?

Problem 7 A different hyperbolic level

Solve $\cosh(x) = 3$ with $y = \sinh(x)$. Give both exact solution pairs and check the hyperbolic identity.

Problem 8 Choose a conventional phase interval

In Section 8 10, restrict y to [0,2π]. Which amplitude–phase vector remains? How many distinct physical phases were represented before this restriction?

Problem 9 Restrict the tangent system to one quadrant

In the tangent-and-circle problem, impose $x >= 0$ and $y >= 0$. Which reference pair remains, and why can a point at $x = \pi/2$ not be accepted?

Problem 10 Change the coefficient in the cycle

Replace every coefficient three in Section 8 12 by two. Does the three-variable system have a real solution? Use monotonicity and the scalar result from Problem 5.

9 15 Answers and mathematical checks

Answer 1 An exponential logarithmic pair

Since $y = \exp(x)$, its logarithm is x. The second equation becomes $2x = 2$, giving $x = 1$ and $y = \exp(1)$. Bounds x in [0,2] and y in [1,4] contain the unique pair and keep the logarithm argument positive.

Answer 2 Change a logarithmic balance

For positive coordinates the log equation gives $xy = 8$. Combining it with $x + y = 6$ gives $x^2 - 6x + 8 = (x - 2)(x - 4) = 0$. The two pairs are (2,4) and (4,2).

Answer 3 Enlarge a periodic interval

The four angles are π/6, 5π/6, 13π/6 and 17π/6. The associated y values are √3/2, −√3/2, √3/2 and −√3/2 respectively. The endpoints zero and 4π are not roots.

Answer 4 Preserve a logarithm domain

The equation $\ln(x^2) = 0$ requires $x^2 = 1$, so $x = -1$ or $x = 1$. Its domain excludes zero but permits either sign. Replacing it by $2\ln(x) = 0$ forces x positive and loses −1. The identity $\ln(x^2) = 2\ln(\text{abs}(x))$ is valid for nonzero real x, but $2\ln(x)$ has the narrower positive domain.

Answer 5 An exponential equation with no real root

Let $F(x) = \exp(x) - 2x$. Its unique minimum is at $x = \ln(2)$, where $F = 2 - 2\ln(2)$ is positive. Hence F never reaches zero and there is no real solution.

Answer 6 Change the cooling target

The difference above ambient must fall from 80 to 20, a factor of one quarter. Thus $\exp(-kt) = 1/4$ and $kt = 2\ln(2)$. With $k = \ln(2)/10$, the time is exactly 20 minutes.

Answer 7 A different hyperbolic level

The two pairs are $(-\ln(3 + \sqrt{8}), -\sqrt{8})$ and $(\ln(3 + \sqrt{8}), \sqrt{8})$. They follow from $u = \exp(x)$ satisfying $u^2 - 6u + 1 = 0$. The check is $3^2 - (\sqrt{8})^2 = 1$, with signs paired according to the oddness of sinh.

Answer 8 Choose a conventional phase interval

Only $(x,y) = (\ln(2)/2, \pi/4)$ remains. The two representatives in $[-2\pi, 2\pi]$ differed by 2π and therefore described one physical phase modulo a complete turn.

Answer 9 Restrict the tangent system to one quadrant

The surviving pair is approximately (1.496659042944, 2.600002251764). The other positive-x intersection has negative y, and the remaining two have negative x. At $x = \pi/2$ tangent is undefined, so the original equation has no value to satisfy; it is not a root.

Answer 10 Change the coefficient in the cycle

The increasing map is now $g(t) = \exp(t)/2$. The same order argument forces $x = y = z$. Their common value would have to satisfy $\exp(a) = 2a$, which Problem 5 proved impossible over the reals. The cyclic system therefore has no real solution.

9 16 From transcendental equations to scientific models

Transcendental systems require the same careful modelling as polynomial systems, together with attention to function domains and repeated branches. Monotonicity can establish uniqueness, convexity can explain two intersections, and periodic formulas can describe infinitely many solutions while selecting a finite list within bounds. A physical measurement model may add further choices about units and equivalent phases.

The next chapter applies equation systems to chemistry and chemical engineering. Concentrations, balances, equilibrium relationships and temperature-dependent laws will place these mathematical tools in a scientific setting. The habits developed here—state the domain, preserve valid branches, check the original equations and justify the answer count—will remain essential.

Chapter 10 Chemistry and Chemical Engineering

Chemical models combine conservation with relationships describing equilibrium, reaction rates, separation and heat transfer. A solver may find a mathematically accurate root that has a negative concentration, an impossible phase fraction or an inconsistent unit conversion. Formulating the balances and interpreting the variables therefore matter as much as finding a small residual.

This chapter develops ten worked problems. We begin with blending and molecular association, then consider weak-acid equilibrium, buffer preparation and solubility. Reactor balances introduce conversion and intermediate products. A binary flash calculation separates liquid and vapour compositions, an Arrhenius fit identifies temperature dependence, and a non-isothermal reactor provides an example with three admissible steady states.

The numerical data are illustrative model inputs, not measured plant data or a thermodynamic property database. Each example states its assumptions and units. Treat its coefficients as fixed for reproducing the mathematical reference answers; additional displayed digits do not imply corresponding experimental accuracy. Use each solver listing separately, with US number style and the variable order specified by var. Marked locations are reserved for your own screenshots.

10 1 Define the basis and the physical domain

Begin with a basis: one litre of solution, a specified molar feed, a mass flow per hour, or a dimensionless reference state. Then distinguish material balances, equilibrium relationships and rate laws. A material balance counts what enters, leaves, forms or disappears. An equilibrium equation relates species in an equilibrium state. A kinetic law specifies a reaction rate and generally does not impose chemical equilibrium.

Units must agree on both sides of every physical equation. A concentration in mol/L is not the same numerical quantity as a concentration in mmol/L or µmol/L. A mole fraction differs from a mass fraction. A temperature in an Arrhenius denominator must be an absolute temperature, normally in kelvin.

Quantity	Usual model restriction	Interpretation
Concentration	Nonnegative	Amount of a species per volume
Mole or mass fraction	Between zero and one	State which kind of fraction
Conversion X	Usually $0 \le X \le 1$ here	Fraction of inlet A consumed
Vapour fraction β	$0 \le \beta \le 1$	Moles of vapour per mole of feed
Absolute temperature T	Positive	Use kelvin in temperature laws

Equilibrium constants must be tied to the quantities used in their definition. Thermodynamic activity-based constants use dimensionless activities; concentration-based coefficients depend on the concentration convention [1]. The solution examples below explicitly use ideal concentration approximations and assigned concentration-form coefficients. These are not interchangeable with a general activity model.

IUPAC defines pH through hydrogen-ion activity [2]. In an ideal dilute concentration approximation, we use $pH \approx -\log_{10}(c_H/c^\circ)$, where c_H is the hydrogen-ion concentration and $c^\circ = 1$ mol/L. The ratio inside the logarithm is dimensionless. At appreciable non-ideality, activity coefficients and an appropriate property model are needed to relate the same composition to thermodynamic pH.

The solver does not attach physical units to a variable name. State the units in the problem and preserve them when entering numerical coefficients. Bounds enforce the intended physical domain; they do not correct a missing species, an incorrect reaction coefficient or a wrong unit conversion.

10 2 Blend two streams to a target mass fraction

Two nonreacting liquid streams contain the same dissolved solute. Their solute mass fractions are 0.10 and 0.40. Determine the feed rates required to produce 100 kg/h of mixture with solute mass fraction 0.25. Assume no loss, reaction or accumulation. Let F1 and F2 be the feed mass flow rates in kg/h.

$$F1 + F2 = 100$$

$$0.10F1 + 0.40F2 = 0.25(100) = 25$$

```
var F1,F2
F1+F2=100
0.10*F1+0.40*F2=25
bound F1=[0,100]
bound F2=[0,100]
```

F1 in kg/h	F2 in kg/h	Solute flow in kg/h
50	50	0.10(50) + 0.40(50) = 25

Eliminate F1 using F1 = 100 − F2. The solute balance becomes 10 + 0.30F2 = 25, giving F2 = 50 and F1 = 50. The solvent balance is then automatic: the inlet solvent flows are 45 and 30 kg/h, totalling the required 75 kg/h.

This example is linear, which is appropriate for its assumptions. Chemical engineering models need not be nonlinear merely because they describe a process. More complicated density or energy relationships would require additional equations, but they are not needed for this mass-fraction blending task.

The target fraction lies between the two feed fractions. A target above 0.40 or below 0.10 cannot be achieved with nonnegative amounts of these two streams alone. That physical feasibility check can be made before any numerical run.

10 3 Conserve monomer units in a dimerization equilibrium

Consider the illustrative equilibrium 2A ⇌ B, in which each molecule of B contains two monomer units. Let a = [A]/Cref and b = [B]/Cref be normalized concentrations. Choose the total monomer-unit concentration as Cref, and assign an equilibrium coefficient that gives b = 2a².

$$b = 2a^2, \quad a + 2b = 1$$

```
var a,b
b=2*a^2
a+2*b=1
bound a=[0,1]
bound b=[0,0.5]
```

The stoichiometric coefficient belongs in the balance

The balance counts one monomer unit in A and two in B. Substituting b = 2a² gives 4a² + a − 1 = 0. Its nonnegative root and the corresponding dimer concentration are:

$$a = (\sqrt{17} - 1)/8, \quad b = (1 - a)/2$$

a normalized	b normalized
0.390388203	0.304805898

The other quadratic root has negative a and violates the concentration domain. For a >= 0, the function $4a^2 + a - 1$ is strictly increasing, so the displayed solution is the only physical one. Substitution verifies both equilibrium and conservation.

Writing a + b = 1 would conserve the number of molecules rather than the number of monomer units. Dimerization changes the number of molecules, so that altered balance describes a different problem. A small residual cannot reveal this modelling error unless the original stoichiometry is checked.

The coefficient two in $b = 2a^2$ is an assigned normalized equilibrium parameter. If a dimensional concentration coefficient Kc were used instead, the normalized coefficient would be Kc Cref. The numerical parameter must therefore be adjusted when the concentration scale changes.

10 4 Calculate a weak acid equilibrium with water ionization

An illustrative monoprotic acid HA is dissolved in water at nominally 25 °C, with analytical acid concentration Ctot = 0.010 mol/L. Use the concentration approximation with assigned coefficients $Ka,c = 1.8 \times 10^{-5}$ mol/L and $Kw,c = 1.0 \times 10^{-14}$ $(mol/L)^2$. The species are H^+, A^-, HA and OH^-. No other ions, added salt or dissolved carbon dioxide are included.

Let cH, cA, cHA and cOH denote concentrations in mol/L. The mass-action, water, analytical and electroneutrality equations are:

$$cH\ cA = Ka,c\ cHA, \quad cH\ cOH = Kw,c$$

$$cA + cHA = Ctot, \quad cH = cA + cOH$$

Scale concentrations before entering the model

Use H, A, HA and OH for the numerical concentrations in µmol/L. Thus $cH = H \times 10^{-6}$ mol/L, and similarly for the other species. The analytical total becomes 10000 in these units. Dividing the acid product equation by $(10^{-6})^2$ makes its coefficient 18, and the water product becomes 0.01:

```
var H,A,HA,OH
H*A=18*HA
H*OH=0.01
A+HA=10000
H=A+OH
bound H=[0.1,10001]
bound A=[0,10000]
bound HA=[0,10000]
bound OH=[0.0000001,0.1]
```

These bounds contain the physical solution and keep every concentration nonnegative. Charge balance with the nonzero water product implies H >= 0.1 in the scaled units. Since A <= 10000, the same equations give an upper bound below 10001. The OH bounds are consistent with OH = 0.01/H and are deliberately slightly wider than required at the lower end.

Species variable	Numerical value in µmol/L	Concentration in mol/L
H	415.359529684	4.153595297e-04
A	415.359505608	4.153595056e-04

Species variable	Numerical value in µmol/L	Concentration in mol/L
HA	9584.640494392	9.584640494e-03
OH	2.407552803e-05	2.407552803e-11

Convert the hydrogen concentration before calculating pH

$$pH \approx 6 - log10(H)$$

The concentration-based reference value is pH ≈ 3.381575821, conventionally reported as about 3.38 for this illustrative chemistry. The six appears because H is a numerical concentration in µmol/L. Taking −log10(H) directly would miss the factor 10^{-6} and shift the answer by six pH units.

The many digits are useful for checking the chosen equations, not as a claim about an experimental acid constant or an activity-corrected pH. The table shows why water ionization can be small here without being set to zero: OH is nonzero and participates in both the water product and charge balance.

Check the four variable model through one scalar equation

Eliminate the other species in scaled units. For a given positive H, A = 18(10000)/(H + 18), HA = 10000 − A and OH = 0.01/H. Charge balance then requires:

$$H - 180000/(H + 18) - 0.01/H = 0$$

Its derivative is $1 + 180000/(H + 18)^2 + 0.01/H^2$, which is positive for H > 0. The residual changes from negative to positive across the physical bounds, so exactly one positive root exists. This supplies an independent check on the four-variable solution and its completeness within the model.

10 5 Design an approximate acid base buffer composition

Prepare one litre of an illustrative HA/A^- buffer with total formal acid-family concentration 100 mmol/L and target pH 5.00. Use the same nominal Ka,c/c° value 1.8×10^{-5} as in the preceding ideal concentration example. Thus pKa = $-\log10(1.8 \times 10^{-5})$, approximately 4.744727495.

Let a and ha be the numerical conjugate-base and acid concentrations in mmol/L. In the buffer approximation, their equilibrium values are taken as approximately equal to the formal amounts supplied, and activity corrections and small dissociation changes are neglected. The Henderson–Hasselbalch relation [3] gives:

$$pH \approx pKa + log10(a/ha), \quad a + ha = 100$$

```
var a,ha
a+ha=100
log10(a/ha)=5+log10(0.000018)
bound a=[1,99]
bound ha=[1,99]
```

Find the ratio before the individual amounts

The target requires a/ha = 10^(5 − pKa) = 1.8. Combining this with the total gives ha = 100/2.8 and a = 180/2.8. The ratio and total determine one positive pair.

a in mmol/L	ha in mmol/L
64.285714286	35.714285714

For one litre, the numerical values are also the required formal millimoles of the two acid-family forms. A real preparation would translate these amounts into the chosen reagents and account for their assay and volume. The model here specifies a composition, not a detailed laboratory preparation procedure.

The calculation deliberately uses the same nominal dissociation coefficient as the preceding example. It does not treat the rounded textbook value pKa = 4.76 as identical to $-\log10(1.8 \times 10^{-5})$. Even a small change in the chosen constant changes the target ratio.

Do not apply the formal-amount approximation automatically to an unbuffered acid, a highly dilute mixture or a composition close to depletion of one form. In such cases the full species and charge balances, as in Section 9 4 with appropriate additional ions, are the relevant model.

10 6 Include a common ion in a solubility balance

Consider an illustrative salt $MX_2(s)$ in equilibrium with M^{2+} and two X^- ions. A soluble background salt initially supplies 1.00 mmol/L of X^- and its spectator counterion. Assume excess solid MX_2 remains, volume is unchanged, free-ion activities are approximated by concentrations, and complex formation is absent.

Assign the concentration-form solubility coefficient $K_{sp,c} = 4.00 \times 10^{-9}$ $(mol/L)^3$. IUPAC defines the thermodynamic solubility product through ion activities [4]; the concentration expression used here is the stated ideal approximation. Let s be the dissolved M concentration and x the total X concentration, both as numerical values in mmol/L.

$$x = 1 + 2s, \quad sx^2 = 4$$

```
var s,x
x=1+2*s
s*x^2=4
bound s=[0,1]
bound x=[1,3]
```

The factor two follows from the dissolution stoichiometry. Because a millimolar scale is 10^{-3} mol/L and the concentration product has total degree three, $K_{sp,c}$ is divided by 10^{-9}, giving the numerical right-hand side four.

s in mmol/L	x in mmol/L
0.697429337	2.394858674

Substituting x = 1 + 2s gives $s(1 + 2s)^2 = 4$. The left-hand side increases strictly for s >= 0, so there is one admissible solubility. The upper bound s <= 1 is safe because without the added common ion the equation would be $4s^3 = 4$, giving s = 1 mmol/L; adding X reduces the dissolved amount.

In mol/L, the dissolved M concentration is approximately 0.000697429337. The background X must be included in the total x; replacing x by 2s would instead describe dissolution in water without that common ion. A saturation equality also presumes solid is available: if all supplied solid dissolves before saturation, a different phase condition is needed.

10 7 Apply a steady state balance to a second order reactor

A liquid-phase, isothermal continuous stirred-tank reactor carries out A → B with disappearance rate $r = kC^2$. The inlet A concentration is C0 = 2 mol/L, the volumetric flow rate is Q = 10 L/min, the volume is V = 100 L and k = 0.1 L/(mol·min). Assume perfect mixing, equal inlet and outlet volumetric flows, constant density and no inlet B.

At steady state, the molar inflow of A minus its outflow equals its consumption rate [5]. If C is the outlet concentration, and X is the fractional conversion:

$$Q(C0 - C) = VkC^2, \quad X = (C0 - C)/C0$$

```
var C,X
10*(2-C)=100*0.1*C^2
X=(2-C)/2
bound C=[0,2]
bound X=[0,1]
```

The two sides of the first equation both have units mol/min. The residence time is V/Q = 10 minutes. With the assigned numbers the concentration equation reduces to $C^2 + C - 2 = 0$, with roots C = 1 and C = −2. Only one is a physical concentration.

C in mol/L	Conversion X	Molar-rate check
1	0.5	In 20 − out 10 = consumed 10 mol/min

The reactor consumes half the inlet A, and the outlet B concentration is 1 mol/L for the stated one-to-one stoichiometry and equal volume flow. At this steady state, the A outlet concentration equals the well-mixed tank concentration. These are algebraic steady-state balances, not numerical integration of a transient reactor equation.

A positive disappearance rate r was defined explicitly. If a convention using a signed species generation rate rA were adopted instead, rA would be negative for A and the balance signs would have to reflect that choice. Define the rate convention before writing the equation.

10 8 Track an intermediate in consecutive reactions

An isothermal, constant-density CSTR carries out A → B → C with irreversible first-order rate constants k1 = 0.5 min^{-1} and k2 = 0.25 min^{-1}. The residence time is two minutes. The feed contains 1 mol/L of A and no B or C. The stoichiometric conversion is one molecule to one molecule at each step, and the volumetric flow is unchanged.

Let a, b and c be outlet concentrations in mol/L. The steady balances are:

$$1 - a = \tau k1\, a, \quad b = \tau(k1\, a - k2\, b), \quad c = \tau k2\, b$$

```
var a,b,c
1-a=2*0.5*a
b=2*(0.5*a-0.25*b)
c=2*0.25*b
bound a=[0,1]
bound b=[0,1]
bound c=[0,1]
```

Formation and consumption both enter the intermediate balance

The A balance gives a = 1/(1 + τk1) = 1/2. The B balance then gives b = τk1 a/(1 + τk2) = 1/3. Finally, c = τk2 b = 1/6. All three balances are linear for these fixed first-order coefficients.

a in mol/L	b in mol/L	c in mol/L	Total
1/2	1/3	1/6	1 mol/L

Half the A has reacted, but only one third of the inlet molar concentration appears as intermediate B; the remainder of the converted A has proceeded to C. Conversion of A and yield of the desired intermediate are therefore different quantities.

Omitting the term −k2 b would treat B as stable and overpredict its outlet concentration. The independent check a + b + c = 1 follows from the one-to-one sequence and the unchanged volume flow. It should be verified even though the three species balances already imply it.

10 9 Split a binary feed into equilibrium liquid and vapour

A binary feed has overall mole fractions z1 = z2 = 0.5. At an assigned temperature and pressure, use fixed illustrative equilibrium ratios K1 = y1/x1 = 2 and K2 = y2/x2 = 0.5. Let β be the molar vapour fraction of the feed, x1,x2 the liquid mole fractions, and y1,y2 the vapour mole fractions.

The component balances are zi = (1 − β)xi + βyi, and each phase composition sums to one. Only one component balance is independent once both phase normalizations and the overall feed total are imposed. Use:

```
var beta,x1,x2,y1,y2
0.5=(1-beta)*x1+beta*y1
x1+x2=1
y1+y2=1
y1=2*x1
y2=0.5*x2
bound beta=[0,1]
bound x1=[0,1]
bound x2=[0,1]
bound y1=[0,1]
bound y2=[0,1]
```

Check the phase compositions and the feed reconstruction

Component	Feed zi	Liquid xi	Vapour yi
1	1/2	1/3	2/3
2	1/2	2/3	1/3

The vapour fraction is β = 1/2. Each phase contains half the total feed moles. For component one, the reconstructed feed is (1/2)(1/3) + (1/2)(2/3) = 1/2, and the second component gives the same overall fraction. Both phase compositions sum to one, and their ratios reproduce the assigned K values.

Use a scalar flash equation as an independent check

Eliminate the phase compositions using xi = zi/[1 + β(Ki − 1)]. The standard Rachford–Rice equation [6] becomes:

$$0.5/(1 + \beta) - 0.25/(1 - 0.5\beta) = 0$$

On [0,1], its denominators remain positive. The function is strictly decreasing, with value 0.25 at zero and −0.25 at one, so it has one root. Substitution gives β = 0.5, consistent with the five-variable model.

The K values here are assigned inputs, not calculated fluid properties. A full flash model needs temperature, pressure and a suitable equilibrium/property description. If a calculated two-phase fraction lies outside [0,1], do not clip it and call the result a solved two-phase flash. The feed may belong to a single-phase region, which requires the appropriate phase test and equations.

10 10 Fit an Arrhenius law at two temperatures

An illustrative first-order rate constant is 0.1 min^{-1} at 300 K and 0.2 min^{-1} at 320 K. Fit the temperature-independent parameters in k(T) = A exp(−E/(RT)), the Arrhenius form [7]. Define B = E/R, measured in kelvin.

To take logarithms consistently, choose the rate reference kref = 1 min^{-1} and let L = ln(A/kref). Then ln(k/kref) = L − B/T. The two observed rate ratios are numerically 0.1 and 0.2:

```
var B,L
L-B/300=ln(0.1)
L-B/320=ln(0.2)
bound B=[0,10000]
bound L=[-10,20]
```

Eliminate the logarithmic pre exponential factor

Subtract the first equation from the second. This gives B(1/300 − 1/320) = ln(2), so B = 4800ln(2). Recover L from either observation, then A = kref exp(L).

Parameter	Reference value	Unit or convention
B	3327.106466688	K
L	8.787769796	ln(A/kref), dimensionless
A	6553.6	min^{-1}
E using R = 8.314	27.661563164	kJ/mol

The arithmetic has a useful exact check: B/300 = 16ln(2), so A = 0.1 × 2^{16} = 6553.6 min^{-1}. At 320 K, B/320 = 15ln(2), giving k = 6553.6/2^{15} = 0.2 min^{-1}. For E, multiply B by R = 8.314 J/(mol·K) and divide by 1000 to obtain kJ/mol.

The fit passes through the two assigned data points exactly. It is not evidence that a measured reaction obeys this law over every temperature. Temperature ranges, parameter uncertainty and possible mechanism changes belong to a real kinetic assessment. In particular, replacing 300 and 320 K by Celsius values in the reciprocal-temperature equation would be a different, incorrect Arrhenius model.

The entered equations are linear in B and L, even though the original rate law is exponential. A useful transformation can simplify parameter estimation while preserving the original units and the exponential prediction used for verification.

10 11 Find three steady states of a non isothermal reactor

A well-mixed reactor carries out an irreversible, first-order exothermic reaction. Use a constant-density, constant-heat-capacity, single-liquid-phase model with equal inlet and outlet flow rates. Feed and coolant temperatures are both Tf = Tc = 300 K. The example is intentionally specified through dimensionless parameters rather than claimed plant measurements.

Define c = CA/CA0 and θ = (T − Tf)/Tf. Let τ = V/Q, Da = τk(Tf), γ = E/(RTf), βh = (−ΔH)CA0/(ρCpTf), and κ = UA/(ρCpQ). Here Cp is a mass-specific heat capacity, ΔH is the molar reaction enthalpy, and βh is positive for an exothermic reaction. Choose Da = 0.01, γ = 25, βh = 1 and κ = 1.

Couple the material and energy balances

The Arrhenius ratio is k(T)/k(Tf) = exp[γθ/(1 + θ)]. The normalized steady material balance is 1 − c = Da exp[γθ/(1 + θ)]c. The energy balance with coolant at feed temperature is (1 + κ)θ = βh(1 − c). These are the steady forms of the material and heat balances discussed in standard CSTR modelling [5].

$$1 - c = 0.01exp(25\theta/(1 + \theta))c, \quad 2\theta = 1 - c$$

```
var c,theta
1-c=0.01*exp(25*theta/(1+theta))*c
2*theta=1-c
bound c=[0,1]
bound theta=[0,0.5]
```

The domain c in [0,1] is the nonnegative reactant fraction for the feed-only-A model. The energy balance then confines θ to [0,0.5], corresponding to temperatures between 300 and 450 K. The endpoints do not satisfy both equations, so every solution has positive concentration and positive temperature rise.

State	c	θ	Conversion 1 – c	T in K
1	0.988610488	0.005694756	0.011389512	301.708427
2	0.592657804	0.203671098	0.407342196	361.101329
3	0.027240243	0.486379879	0.972759757	445.913964

These are three different admissible algebraic steady states for exactly the same feed and assigned parameters. The state at the largest temperature has the greatest conversion. Its being a root, however, does not establish dynamic stability or make it a preferred operating point.

Count the admissible roots independently

Eliminate c using $c = 1 - 2\theta$. For $0 < \theta < 0.5$, both sides of the material balance are positive, so taking logarithms is valid. Define:

$$g(\theta) = ln(2\theta/[0.01(1 - 2\theta)]) - 25\theta/(1 + \theta)$$

The derivative is $g'(\theta) = 1/[\theta(1 - 2\theta)] - 25/(1 + \theta)^2$. Setting it to zero gives the quadratic $51\theta^2 - 23\theta + 1 = 0$, whose two roots in the physical interval are:

$$\theta_1 = (23 - 5\sqrt{13})/102, \quad \theta_2 = (23 + 5\sqrt{13})/102$$

These turning points are approximately 0.048747486 and 0.402232906. The function rises, then falls, then rises. Its limit is negative infinity near zero and positive infinity near 0.5. Moreover, $g(\theta_1) \approx 1.217756$ is positive and $g(\theta_2) \approx -1.151679$ is negative. Thus each monotone interval contains exactly one zero, giving exactly three admissible steady states.

Independent scalar bracketing on these three intervals recovers the table values. Substituting $c = 1 - 2\theta$ then verifies both original balances. This reference count is stronger evidence than a timed run that happens to return three rows; it also identifies when a short run has found only part of the expected answer set.

A steady state count is not a stability result

Dynamic stability requires the time-dependent concentration and energy balances and an appropriate perturbation or Jacobian analysis. The static equations alone do not classify the three roots as stable or unstable. The heat-generation and heat-removal structure explains why multiplicity is possible; a separate dynamic model determines what happens after a disturbance.

10 12 Check the chemistry as well as the residual

For each root, verify total material or elemental conservation, charge balance when ions are present, composition normalization, and the assumed phase and sign conditions. Recalculate dimensional concentrations and temperatures before interpreting pH, conversion or kinetic parameters. A correct numerical solution cannot compensate for a missing counterion or a stoichiometric factor omitted from a balance.

Scaling should be explicit. If all concentrations are divided by a reference scale s, a degree-two concentration product changes by s^2 and a degree-three product by s^3. The weak-acid example and the solubility example therefore require different changes in their numerical constants. Report the reference scale beside the variable definitions.

Distinguish the number of algebraic roots from the number of physically admissible roots. The dimer model and second-order CSTR each have a negative concentration branch that is rejected. The non-isothermal CSTR has three admissible steady states. The flash model requires phase fractions and compositions in their physical ranges.

Numerical tolerance describes the entered residuals, whose magnitudes depend on units and scaling. It does not supply uncertainty in an equilibrium coefficient, validate an activity approximation, or establish that all roots have been found. Use the independent reductions in this chapter to check both the reference values and their completeness under the stated assumptions.

For your screenshots, enlarge the equation and solution fonts, preserve the declared variable order and keep unit labels in the caption. Record actual timings and precision from the run. The reference tables here provide mathematical checks without inventing software screenshots or plant-performance claims.

10 13 Practice problems

Use a separate model for each exercise. State the basis and units, retain the physical bounds, and explain the independent check supporting the answer.

Problem 1 An infeasible blend

In Section 9 2, keep the feed fractions 0.10 and 0.40 but require a product fraction of 0.55 at 100 kg/h. Can nonnegative feed rates meet the target?

Problem 2 A missing stoichiometric factor

Replace a + 2b = 1 by a + b = 1 in the dimer model. Solve the altered equations, then check whether that solution conserves one unit of monomer.

Problem 3 Convert a scaled concentration

Use H = 415.359529684 in the weak-acid example to calculate the concentration in mol/L and the approximate pH. What offset occurs if H is logged directly without conversion?

Problem 4 Equal acid and base in a buffer

In Section 9 5, set the target pH equal to the assigned pKa. What normalized ratio and mmol/L amounts are required for the same total 100 mmol/L?

Problem 5 Remove the common ion

Set the background X concentration in Section 9 6 to zero. Find s and x in mmol/L, and compare the solubility with the common-ion result.

Problem 6 Increase the reactor volume

Double the volume of the second-order CSTR from 100 to 200 L, keeping Q, C0 and k fixed. Find the new nonnegative concentration and conversion.

Problem 7 Prevent further conversion of the intermediate

In Section 9 8, set k2 = 0 while retaining the feed and residence time. What are a,b,c and the fraction of feed A appearing as B?

Problem 8 Change the flash feed

Change the overall fraction z1 to 0.4, with z2 = 0.6 and the same K values. Find β and the two phase compositions. Would z1 = 0.8 admit the same two-phase formulation with β between zero and one?

Problem 9 Recover dimensional Arrhenius parameters

From the fitted B and L in Section 9 10, recover A and E using R = 8.314 J/(mol·K). Why must the temperatures used in the fit be kelvin?

Problem 10 Interpret the three steady states

Which state in Section 9 11 has the greatest conversion, and what is its temperature? Does this establish that it is dynamically stable?

10 14 Answers and mathematical checks

Answer 1 An infeasible blend

No. The target exceeds the richer feed fraction, so it is outside the range of nonnegative mixtures. Algebra gives F2 = 150 kg/h and F1 = −50 kg/h. The negative feed rate violates the physical domain.

Answer 2 A missing stoichiometric factor

The altered equations give a = 0.5 and b = 0.5. Their true monomer total is a + 2b = 1.5, not one. They solve the altered mathematical problem but do not satisfy the original conservation law.

Answer 3 Convert a scaled concentration

The hydrogen concentration is approximately $4.15359529684 \times 10^{-4}$ mol/L. The concentration-based pH is approximately 3.381575821. Logging the numerical µmol/L value directly gives approximately −2.618424179, six units below the intended result.

Answer 4 Equal acid and base in a buffer

The Henderson–Hasselbalch concentration ratio a/ha is one. The total then gives a = ha = 50 mmol/L. The target pH is the chosen pKa, approximately 4.744727495, within the stated buffer approximation.

Answer 5 Remove the common ion

Now $x = 2s$ and $sx^2 = 4$, so $4s^3 = 4$. Hence s = 1 mmol/L and x = 2 mmol/L. This is greater than the dissolved-M concentration 0.697429337 mmol/L with the background common ion.

Answer 6 Increase the reactor volume

The residence time doubles to 20 minutes. The concentration equation becomes $2C^2 + C - 2 = 0$. Its nonnegative root is $C = (\sqrt{17} - 1)/4 \approx 0.780776406$ mol/L, giving conversion $X = (2 - C)/2 \approx 0.609611797$. The other root is negative.

Answer 7 Prevent further conversion of the intermediate

The A balance still gives a = 0.5 mol/L. With no consumption of B, b = 0.5 mol/L and c = 0. The total is one, and half the inlet A appears as B. Intermediate yield now equals A conversion for this simplified one-to-one sequence.

Answer 8 Change the flash feed

The K values and phase normalizations still give liquid composition (1/3,2/3) and vapour composition (2/3,1/3). The first component balance gives $\beta = 0.2$ for $z1 = 0.4$. If $z1 = 0.8$, it would require $\beta = 1.4$, outside [0,1]; no admissible two-phase root exists for that formulation. This does not mean that no physical single-phase flash state exists.

Answer 9 Recover dimensional Arrhenius parameters

$A = \exp(L) \times 1\ \text{min}^{-1} = 6553.6\ \text{min}^{-1}$. The activation energy is $E = BR \approx 27661.563$ J/mol, or 27.661563 kJ/mol with the stated rounded R. The Arrhenius reciprocal temperature is absolute; using Celsius would alter the mathematical temperature law and the inferred parameters.

Answer 10 Interpret the three steady states

The third state has $c \approx 0.027240243$ and conversion $1 - c \approx 0.972759757$. Its temperature is approximately 445.913964 K. These values identify an admissible algebraic steady state, not its dynamic stability. A transient balance model and stability analysis are needed for that classification.

10 15 References for the model definitions

The references below support terminology and standard balance or constitutive forms. The numerical examples, assigned coefficients, transformations and reference calculations in this chapter are specified within the worked problems.

[1] IUPAC Gold Book. Equilibrium constant, E02177.
https://goldbook.iupac.org/terms/view/E02177

[2] IUPAC Gold Book. pH, P04524.
https://goldbook.iupac.org/terms/view/P04524

[3] IUPAC Gold Book. Henderson–Hasselbalch equation, H02781.
https://goldbook.iupac.org/terms/view/H02781

[4] IUPAC Gold Book. Solubility product, S05742.
https://goldbook.iupac.org/terms/view/S05742

[5] LearnChemE. Multiple Steady States in a Non-Isothermal CSTR, screencasts and governing balances.
https://learncheme.com/quiz-yourself/interactive-self-study-modules/multiple-steady-states-in-a-non-isothermal-cstr/multiple-steady-states-in-a-non-isothermal-cstr-screencasts/

[6] Chemical Engineering Design Library. Rachford–Rice Equation Solvers, Chemicals documentation.
https://chemicals.readthedocs.io/chemicals.rachford_rice.html

[7] IUPAC Gold Book. Arrhenius equation, A00446.
https://goldbook.iupac.org/terms/view/A00446

10 16 From chemical models to wider engineering applications

Chemical equation systems become useful when their balances, rate laws, equilibrium relationships and units describe the same physical situation. The examples show how positive concentrations, charge balance, phase conditions and energy coupling select admissible roots. Independent algebra and scalar checks make the numerical results easier to assess.

The next chapter broadens the discussion to physics and engineering applications. The same modelling discipline carries forward: define the quantities, choose a consistent unit system, impose physical bounds, seek alternative solutions where they are possible and verify each candidate against the original equations.

Chapter 11 Physics and Engineering Applications

Physics and engineering models turn balances, geometry and constitutive laws into equations. The unknowns may be a launch angle, a tension, an electrical operating point, a temperature or a flow rate. A useful solution must satisfy the equations and belong to the intended physical domain. Multiple roots may describe alternative trajectories or configurations, while other roots may be excluded by the assumptions.

This chapter develops ten examples for Patodia Equation Solver. Each includes a physical statement, variable definitions, a copyable input, reference values and an independent check. The examples progress from motion and statics to nonlinear circuits, thermal balances, fluid systems, vibration, mechanism geometry and parameter estimation. They illustrate algebraic models and explicit response formulas; the solver is not being used here as a numerical differential equation integrator.

The numerical parameters are assigned for teaching and reproducibility. They are not measurements or equipment specifications. Extra digits in the reference tables support arithmetic checks, rather than claims of experimental accuracy. Run each input separately. The var declaration fixes the requested output order, and marked locations are reserved for screenshots from your own runs.

11 1 Establish units directions and model limits

Write the physical assumptions before entering the equations. Choose a coordinate direction, state whether a quantity is signed or nonnegative, and use one consistent set of units. A negative coordinate can be meaningful even when a negative cable tension is not. A pressure head is measured in units of length, whereas an electrical potential is measured in volts.

Model quantity	Units used here	Domain or convention
Launch angle θ	radians	Measured above the horizontal
Cable tension	N	Nonnegative for a taut cable
Diode current i	mA	Forward direction chosen positive
Radiating temperature	K	Absolute temperature in fourth powers
Pipe flow Q	L/s	Prescribed forward direction
Angular frequency ω	rad/s	Positive; f = ω/(2π) in Hz

The trigonometric listings use radians. If an angle is specified in degrees, convert it before evaluating sine or cosine. Temperature differences can have the same numerical value in kelvin and degrees Celsius, but a temperature raised to the fourth power must be an absolute temperature. Likewise, the argument of an exponential must be dimensionless.

A solver returns roots of the equations actually entered. It cannot infer a missing torque equation, distinguish a flow coefficient calibrated in L/s from one calibrated in m^3/s, or determine whether a simplified device law remains accurate. Independent balances and reductions make those modelling choices reviewable. Bounds restrict the search domain; a root count applies only within that domain and under the stated model.

11 2 Find two launch angles for the same range

A projectile is launched at 20 m/s and lands at its launch elevation, 30 m horizontally away. Use constant downward acceleration g = 9.81 m/s², neglect drag and assume unobstructed flight. Find all upward launch angles between zero and π/2 and their positive flight times. Let θ be the angle in radians and t the time in seconds. The horizontal and vertical constant-acceleration equations provide the model [1].

$$20\,cos(\theta)t\ =\ 30,\quad 20\,sin(\theta)t\ -\ (9.81/2)t^2\ =\ 0$$

```
var theta,t
20*cos(theta)*t=30
20*sin(theta)*t-0.5*9.81*t^2=0
bound theta=[0.001,1.57]
bound t=[0.001,5]
```

θ in radians	t in seconds
0.413386715	1.637973753
1.157409612	3.734008520

For a positive flight time, the vertical equation gives t = 40 sin(θ)/9.81. Substituting into the horizontal equation yields sin(2θ) = 30(9.81)/400 = 0.73575. Therefore θ_1 = asin(0.73575)/2 and $\theta_2 = \pi/2 - \theta_1$. These are the only two angles in the upward forward quadrant because 2θ spans (0,π).

The angles are approximately 23.685314° and 66.314686°. The steeper trajectory takes longer. Neither root is an erroneous duplicate: both satisfy the same range and elevation requirements, but describe different paths.

The stated upper bound of 5 s covers both flights; the longest possible equal-elevation flight at this speed is 40/9.81, approximately 4.08 s. The horizontal equation already excludes t = 0. With these assumptions the maximum range is 400/9.81, approximately 40.775 m. A range above that value has no such trajectory, and equality gives a single angle of π/4.

11 3 Resolve the tensions in two supporting cables

A massless joint carries a vertical downward load of 1000 N. Two taut cables pull away from the joint along the unit directions (−0.6,0.8) and (0.8,0.6), with upward y and rightward x positive. Find the tensions T1 and T2 in newtons. The load acts at the joint, so all forces are concurrent; a separate rigid-body lever arm is absent.

$$-0.6T1\ +\ 0.8T2\ =\ 0,\quad 0.8T1\ +\ 0.6T2\ =\ 1000$$

```
var T1,T2
-0.6*T1+0.8*T2=0
0.8*T1+0.6*T2=1000
bound T1=[0,2000]
bound T2=[0,2000]
```

T1 in N	T2 in N
800.000	600.000

Horizontal equilibrium gives T2 = 0.75T1. The vertical equation then becomes 1.25T1 = 1000, giving T1 = 800 N and T2 = 600 N. The opposing horizontal components are −480 and +480 N; the upward components are 640 and 360 N. Their sum supports the 1000 N load exactly.

Both direction vectors have unit length, so their coefficients are direction cosines rather than cable lengths. Multiplying a tension by these components gives force components with the same units as the applied load. The coefficient determinant is −1, which establishes a unique algebraic solution.

For this concurrent-force joint, force balance is sufficient. A beam with forces applied at separated points generally also needs moment balance [2]. If a different joint model returned a negative cable tension, that cable would be required to push. A slack-cable model or a different support arrangement would then be needed.

11 4 Solve a resistor and diode operating point

A 5 V ideal DC source drives a 1 kΩ resistor in series with an idealized junction diode. Let V be the forward diode voltage in volts and i the current in milliamperes. Assign saturation current Is = 10^{-12} A and the combined ideality and thermal-voltage parameter nVT = 0.05 V. Neglect breakdown, leakage beyond this law, series resistance inside the diode and self-heating.

Kirchhoff voltage balance [3] gives V + Ri = 5. The forward branch is represented by the exponential diode law [4]. With current measured numerically in mA, Is becomes 10^{-9} mA, and a current of 1 mA produces a 1 V drop across the 1 kΩ resistor. Thus the entered numerical system is:

$$V + i = 5, \quad i = 10^{-9}[exp(V/0.05) - 1]$$

```
var V,i
V+1*i=5
i=0.000000001*(exp(V/0.05)-1)
bound V=[0,5]
bound i=[0,5]
```

V in volts	i in mA
1.104158773	3.895841227

Eliminate i and define f(V) = V + 10^{-9}[exp(V/0.05) − 1] − 5. On [0,5], f(0) = −5 and f(5) is positive. Its derivative is 1 + (10^{-9}/0.05)exp(V/0.05), which is strictly positive. Exactly one root therefore exists in this forward-bias interval. Scalar bracketing gives the same voltage and current as the coupled system.

A further check is power balance. In milliwatts, source power is 5i, resistor dissipation is i^2, and diode dissipation is Vi. Multiplying the voltage equation by i gives 5i = i^2 + Vi. This check also exposes an accidental use of amperes in a listing whose current scale is milliamperes.

The parameters are illustrative rather than a fitted model of a named component. The solution is the operating point of these two equations. Temperature-dependent or high-current behavior would require a more complete device model and suitable parameter data.

11 5 Balance convection and thermal radiation

An isothermal surface receives 100 W of heating. Its exposed area is 0.1 m^2, its convection coefficient is 10 W/(m^2·K), and its emissivity is 0.8. Both ambient air and a large surrounding enclosure are at 300 K. Assume the surface sees only that enclosure, has no other heat losses, and has constant properties. Use σ = 5.67 × 10^{-8} W/(m^2·K^4), a rounded radiation constant for this example.

Let T be surface temperature in kelvin, qc the convective heat loss in watts and qr the net radiative heat loss in watts. The standard convection and radiation forms [5] give:

$$qc = hA(T - 300), \quad qr = \varepsilon\sigma A(T^4 - 300^4)$$

$$qc + qr = 100$$

```
var T,qc,qr
qc=10*0.1*(T-300)
qr=0.8*0.0000000567*0.1*(T^4-300^4)
qc+qr=100
bound T=[300,500]
bound qc=[0,100]
bound qr=[0,100]
```

T in K	qc in W	qr in W
360.299968992	60.299968992	39.700031008

After eliminating the two heat rates, the scalar heat-loss function is F(T) = (T − 300) + 4.536 × 10^{-9}(T^4 − 300^4) − 100. It is negative at 300 K and positive at 500 K. Its derivative is 1 + 4(4.536 × 10^{-9})T^3, strictly positive for positive T. Consequently this domain contains exactly one steady temperature.

The convection-only estimate would be 400 K. Net radiation is positive above 300 K, so adding radiation lowers the required temperature below 400 K. The computed partition qc + qr = 100 W verifies energy conservation. All fourth-power temperatures, including the enclosure temperature, must be in kelvin.

This is a uniform-temperature surface model. It does not calculate internal temperature gradients, view factors for a complicated enclosure, or variation of emissivity and convection coefficient. Those effects require additional physical relationships rather than a tighter numerical tolerance.

11 6 Determine surface temperatures across a wall

A plane wall separates air at 100 °C from air at 20 °C. Use area 1 m², wall thickness 0.05 m, conductivity 0.2 W/(m·K), hot-side convection coefficient 10 W/(m²·K) and cold-side coefficient 20 W/(m²·K). Assume steady one-dimensional conduction, no heat generation, no radiation and constant properties. Let T1 and T2 be the hot and cold wall surface temperatures in °C and q the heat transfer rate in watts.

The convection resistances are 0.10 and 0.05 K/W; the wall resistance L/(kA) is 0.25 K/W. The same heat rate passes through all three resistances [5]. Because only differences of temperature occur, Celsius is consistent here.

$$q = 10(100 - T1) = (T1 - T2)/0.25 = 20(T2 - 20)$$

```
var T1,T2,q
q=10*(100-T1)
q=(T1-T2)/0.25
q=20*(T2-20)
bound T1=[20,100]
bound T2=[20,100]
bound q=[0,1000]
```

T1 in °C	T2 in °C	q in W
80.000	30.000	200.000

The total resistance is 0.10 + 0.25 + 0.05 = 0.40 K/W. Therefore q = (100 − 20)/0.40 = 200 W. The hot film drops by q(0.10) = 20 K, the wall by 50 K and the cold film by 10 K. This gives surface temperatures of 80 °C and 30 °C.

The monotone temperature order 100 > T1 > T2 > 20 is a useful physical check for this passive wall. It follows from positive thermal resistances and positive heat flow. The coupled equations are linear, and the resistance sum supplies an independent unique solution without searching.

Do not confuse q with heat flux. Here q is a rate in W. Dividing by the area gives flux in W/m²; the numerical values happen to agree because the chosen area is 1 m². For another area, every resistance must be updated consistently.

11 7 Split a flow between parallel pipes

Two passive branches join the same upstream and downstream nodes. Their total forward flow is 6 L/s. Assign head-loss laws H1 = 2Q1² and H2 = 8Q2², where the numerical Q values are in L/s and head losses are in metres. The coefficients therefore carry units m/(L/s)². Assume steady incompressible flow and no devices that add head within either branch.

Continuity requires the branch flows to sum to the supplied flow. The common endpoints impose the same head loss H across both branches. Quadratic loss laws are a useful assigned approximation for this example; detailed pipe models may require a friction factor that varies with Reynolds number [6].

$$Q1 + Q2 = 6, \quad H = 2Q1^2, \quad H = 8Q2^2$$

```
var Q1,Q2,H
Q1+Q2=6
H=2*Q1^2
H=8*Q2^2
bound Q1=[0,6]
bound Q2=[0,6]
bound H=[0,300]
```

Q1 in L/s	Q2 in L/s	H in metres
4.000	2.000	32.000

Equating the head losses gives Q1² = 4Q2². The nonnegative-flow domain selects Q1 = 2Q2; continuity then gives Q1 = 4 and Q2 = 2 L/s. Both head losses equal 32 m. The branch with the larger loss coefficient carries less flow.

Without the sign restrictions, squaring would also allow Q1 = −2Q2, producing a second algebraic branch. The assumed head-loss laws apply here to forward flows only. A network allowing reversals needs signed losses such as KQ|Q| together with a stated node orientation; it should not use KQ² as a signed pressure difference for every direction.

Changing the flow scale changes the coefficients. Since 1 L/s = 0.001 m³/s, expressing Q numerically in m³/s would multiply these quadratic coefficients by one million. Keeping the original coefficients after that conversion would define a different hydraulic model.

11 8 Locate a pump and system curve intersection

A pump has the illustrative head curve Hp = 50 − 0.5Q², and its connected system requires Hs = 10 + 0.3Q². Use Q in L/s and heads in metres. The 10 m term is a static head requirement. These assigned curves are assumed valid over 0 ≤ Q ≤ 10 L/s; no claim about a commercial pump is intended.

$$H = 50 - 0.5Q^2, \quad H = 10 + 0.3Q^2$$

```
var Q,H
H=50-0.5*Q^2
H=10+0.3*Q^2
bound Q=[0,10]
bound H=[0,50]
```

Q in L/s	H in metres
7.071067812	25.000000000

Subtract the equations to obtain 40 = 0.8Q². The forward-flow operating point is Q = √50 L/s and H = 25 m. The negative square root lies outside the prescribed flow domain. The pump head decreases and system head increases with positive Q, so there is only one forward intersection in this range.

At the intersection the pump supplies exactly the head the system requires. A root alone does not determine motor power: hydraulic power also needs fluid density and gravity, and shaft power additionally needs pump efficiency. Nor does a curve intersection by itself establish acceptable suction conditions, mechanical limits or transient stability.

For a numerical run, check both original curves at the returned Q. A residual in metres and a residual in squared flow units are different quantities; reducing to one equation does not remove the need to preserve the original units in the interpretation.

11 9 Find two frequencies giving the same vibration amplitude

A mass of 1 kg is connected to a spring of stiffness 100 N/m and a viscous damper with coefficient 2 N·s/m. A sinusoidal force of amplitude 10 N drives the system. For the steady sinusoidal response, find all positive angular forcing frequencies giving displacement amplitude X = 0.2 m. Transient motion is assumed to have decayed.

The linear equation of motion is m x″ + c x′ + kx = F0 cos(ωt). Its steady response amplitude [7] is X = F0/√[(k − mω²)² + (cω)²]. We use this explicit algebraic relation; no time integration is needed. Let omega denote ω in rad/s and X the nonnegative amplitude in metres.

$$X^2[(100 - \omega^2)^2 + (2\omega)^2] = 100, \quad X = 0.2$$

```
var omega,X
X^2*((100-omega^2)^2+(2*omega)^2)=100
X=0.2
bound omega=[0.001,20]
bound X=[0.001,1]
```

ω in rad/s	X in metres
7.220153754	0.200000000
11.994556255	0.200000000

Set u = ω². With X = 0.2, the equation becomes (100 − u)² + 4u = 2500, or u² − 196u + 7500 = 0. Thus u = 98 ± √2104. Both values are positive, giving exactly two positive frequencies, √(98 − √2104) and √(98 + √2104). Substitution into the unsquared amplitude formula confirms both roots.

The two frequencies lie on different sides of the amplitude peak. For this damped displacement response, the peak occurs at ω = √(k/m − c²/(2m²)) = √98 rad/s, rather than exactly at the undamped natural frequency √(k/m) = 10 rad/s. This follows by minimizing the squared denominator with respect to u. The peak amplitude is 10/√396, approximately 0.502519 m, so the target 0.2 m is reachable on both sides.

The positivity of X matters because squaring an amplitude relation loses its sign. The listing explicitly fixes positive X. Frequency in hertz is f = ω/(2π), not ω itself. These two forcing frequencies are alternative inputs that produce the same amplitude; they are not two simultaneous frequencies of free vibration.

11 10 Find both positions of a two link joint

Two rigid links of length 5 units connect fixed pivots at A = (0,0) and B = (6,0) to a common free joint P = (x,y). Find the joint positions. The ideal links rotate freely at their pivots; assume no obstacle or assembly constraint excludes either side of the line AB. The distance equations are:

$$x^2 + y^2 = 25, \quad (x - 6)^2 + y^2 = 25$$

```
var x,y
x^2+y^2=25
(x-6)^2+y^2=25
bound x=[0,6]
bound y=[-5,5]
```

x in length units	y in length units
3.000	-4.000
3.000	4.000

Subtract the first equation from the second. This gives −12x + 36 = 0, hence x = 3. Then y² = 25 − 9 = 16, giving y = ±4. Each configuration places the free joint exactly 5 units from both pivots. The two roots represent reflected assembly positions.

For an upper-only assembly, add y >= 0 or replace its bound by [0,5]; the physical restriction then retains only (3,4). If the pivots were 10 units apart, the circles would be tangent and the joint would lie at (5,0). A larger pivot spacing would make these two fixed link lengths geometrically incompatible.

This is a position calculation, with no force or acceleration model attached. An actual mechanism may require joint limits, collision checks, loading and a choice of assembly branch. Finding every distance-equation root is useful because it makes those additional physical selections explicit.

11 11 Estimate cooling parameters from two observations

An object starts at 100 °C. Its measured temperatures for this illustrative exact-data problem are 70 °C after 5 min and 50 °C after 10 min. Assume uniform object temperature, a constant surrounding temperature Ta, and an exponential cooling response T(t) = Ta + (100 − Ta)exp(−kt), where k is positive in min^{-1}. Estimate Ta and k.

This response is the closed-form solution of the lumped Newton cooling model. It assumes the temperature difference drives a constant effective heat-transfer coefficient and that internal gradients can be neglected. We fit its two unknown parameters to two prescribed observations, using minutes consistently in kt.

$$Ta + (100 - Ta)exp(-5k) = 70$$

$$Ta + (100 - Ta)exp(-10k) = 50$$

```
var Ta,k
Ta+(100-Ta)*exp(-5*k)=70
Ta+(100-Ta)*exp(-10*k)=50
bound Ta=[0,49]
bound k=[0.001,1]
```

Ta in °C	k in inverse minutes
10.000000000	0.081093022

Let r = exp(−5k). Equal observation intervals imply 70 − Ta = (100 − Ta)r and 50 − Ta = (100 − Ta)r². Eliminating r gives (70 − Ta)² = (100 − Ta)(50 − Ta). Expanding yields 4900 − 140Ta = 5000 − 150Ta, so Ta = 10 °C. Then r = 60/90 = 2/3 and k = ln(3/2)/5.

The bounds Ta < 50 and k > 0 enforce cooling toward an environment below both observations. They also make the elimination legitimate: 100 − Ta is nonzero, and the fitted r is between zero and one. This establishes one admissible parameter pair. The predicted temperature after 15 min is 10 + 90(2/3)³ = 36.666667 °C, which offers a further model prediction to test with new data.

Passing through two observations exactly is not evidence that real cooling data follow this law over every time interval. Extra observations can reveal nonuniform temperature, varying surroundings or measurement error. With more observations than unknown parameters, an estimation or least-squares formulation is generally needed rather than forcing every noisy measurement into an exact equality.

11 12 Check physical validity and solution completeness

Start verification by substituting each root into the original equations. For a coupled model, check every balance rather than only the reduced equation. The diode should satisfy voltage and power balance; the thermal model should partition exactly 100 W; the two pipe branches should have equal head loss; and both mechanism distances should equal 5.

The independent reductions in this chapter also provide reference counts. The projectile range relation has two upward angles, the vibration quadratic has two positive frequencies, and the link geometry has two reflected joint positions. The other worked models each have one admissible root in the stated domain. A short numerical search that returns fewer rows has not thereby disproved the missing reference roots.

A small residual does not establish that a support can carry a load, a diode model fits a component, a pump can operate safely or a steady response is dynamically stable. Those conclusions need the corresponding physical criteria and data. Numerical precision describes the solution of a specified model; uncertainty in coefficients and assumptions is a separate matter.

For screenshots, enlarge the equation and solution fonts, preserve the declared variable order and record the units in the caption. Use actual run times and residuals from your own run. The tables in this chapter provide reproducible reference values without claiming a particular execution speed.

11 13 Practice problems

Solve each variant separately. State the revised domain, calculate the expected roots independently where possible, and verify the original physical equations before interpreting a numerical result.

Problem 1 Maximum projectile range

In Section 10 2, keep the speed at 20 m/s but require a horizontal range of 45 m at the launch elevation. Is there an upward launch angle under the same assumptions?

Problem 2 Increase the joint load

In Section 10 3, increase the downward load to 1500 N without changing the cable directions. Find both tensions.

Problem 3 Change the current scale

Rewrite Section 10 4 using current I in amperes instead of i in milliamperes. State the two equations and convert the reference current.

Problem 4 Remove thermal radiation

In Section 10 5, set emissivity to zero while leaving the heater and convection parameters unchanged. Find T, qc and qr.

Problem 5 Increase wall thickness

Double the wall thickness in Section 10 6 to 0.10 m while retaining conductivity, area and both film coefficients. Find q, T1 and T2.

Problem 6 Increase total pipe flow

In Section 10 7, increase the supplied flow to 9 L/s and retain both loss coefficients. Find the branch flows and the common head loss.

Problem 7 Raise the static head

In Section 10 8, replace the system curve by $H_s = 20 + 0.3Q^2$. Find the new forward operating point. What happens if the static term is raised above 50 m?

Problem 8 Locate the response peak

In Section 10 9, derive the angular frequency and amplitude at the maximum steady displacement response. How many positive frequencies give amplitude 0.6 m?

Problem 9 Restrict the mechanism assembly

In Section 10 10, require the free joint to lie above the pivot line. Find the remaining position. Then move the second pivot to (10,0) and determine the limiting configuration.

Problem 10 Predict a later temperature

Using the fitted model in Section 10 11, predict the temperature after 20 min. Convert k into s^{-1} and explain why inserting that value into an exponent with time still in minutes is incorrect.

11 14 Answers and mathematical checks

Answer 1 Maximum projectile range

No. The maximum range is $v_0^2/g = 400/9.81 \approx 40.774720$ m. A required range of 45 m would make $\sin(2\theta) = 1.103625$, outside $[-1,1]$. Increasing the search duration cannot create a root of this model.

Answer 2 Increase the joint load

$T_1 = 1200$ N and $T_2 = 900$ N. The load increased by a factor of 1.5, and the linear equilibrium system makes both tensions increase by the same factor. The vertical components are 960 and 540 N, totalling 1500 N.

Answer 3 Change the current scale

Use $V + 1000I = 5$ and $I = 10^{-12}[\exp(V/0.05) - 1]$. The reference current is $I \approx 0.003895841227$ A; the voltage is unchanged. The resistor coefficient and saturation current both change numerically with the chosen current unit.

Answer 4 Remove thermal radiation

$T = 400$ K, $q_c = 100$ W and $q_r = 0$. The required convective temperature rise is $100/(10 \times 0.1) = 100$ K. Removing radiation requires a higher surface temperature than the original combined-loss model.

Answer 5 Increase wall thickness

The wall resistance becomes 0.50 K/W and the total resistance becomes 0.65 K/W. Therefore $q = 80/0.65 \approx 123.076923$ W, $T_1 = 100 - q/10 \approx 87.692308$ °C, and $T_2 = 20 + q/20 \approx 26.153846$ °C. Each resistance gives the same heat rate.

Answer 6 Increase total pipe flow

Q1 = 6 L/s, Q2 = 3 L/s and H = 72 m. The ratio Q1/Q2 remains two. Increasing total flow by 1.5 multiplies each branch flow by 1.5 and each quadratic head loss by 2.25.

Answer 7 Raise the static head

Equating the curves gives $30 = 0.8Q^2$, so $Q = \sqrt{37.5} \approx 6.123724$ L/s and H = 31.25 m. If the static head exceeds 50 m, the required system head exceeds the maximum assigned pump head even at zero flow, so there is no forward-flow intersection of these curves.

Answer 8 Locate the response peak

The squared amplitude denominator is $u^2 - 196u + 10000$, where $u = \omega^2$. Its minimum occurs at u = 98, so $\omega_{peak} = \sqrt{98} \approx 9.899495$ rad/s and $X_{peak} = 10/\sqrt{396} \approx 0.502519$ m. A target of 0.6 m exceeds this maximum, hence has no positive-frequency solution under the model.

Answer 9 Restrict the mechanism assembly

The upper configuration is (3,4). With the pivots at (0,0) and (10,0), the circles are tangent at (5,0). That point lies on the pivot line; it is admissible if $y \geq 0$ is allowed, but fails a strictly-above requirement $y > 0$.

Answer 10 Predict a later temperature

$T(20) = 10 + 90(2/3)^4 = 250/9 \approx 27.777778$ °C. The rate $k = \ln(3/2)/5 \approx 0.081093022$ min^{-1} becomes $k/60 \approx 0.001351550$ s^{-1}. Pair that rate with elapsed time in seconds so that the product kt remains dimensionless.

11 15 References for the physical models

These references support the standard laws and model forms. The problem statements, numerical parameters, reductions, root counts and worked calculations in this chapter are specified independently for the present examples. No textbook figures or worked problem statements are reproduced.

[1] OpenStax University Physics Volume 1 Section 4 3 Projectile Motion
https://openstax.org/books/university-physics-volume-1/pages/4-3-projectile-motion

[2] OpenStax University Physics Volume 1 Section 12 1 Conditions for Static Equilibrium
https://openstax.org/books/university-physics-volume-1/pages/12-1-conditions-for-static-equilibrium

[3] OpenStax University Physics Volume 2 Section 10 3 Kirchhoff Rules
https://openstax.org/books/university-physics-volume-2/pages/10-3-kirchhoffs-rules

[4] ngspice User Manual Diode DC Transient and AC Model Equations
https://nmg.gitlab.io/ngspice-manual/diodes/diodeequations/diodedc,transientandacmodelequations.html

[5] OpenStax University Physics Volume 2 Section 1 6 Mechanisms of Heat Transfer
https://openstax.org/books/university-physics-volume-2/pages/1-6-mechanisms-of-heat-transfer

[6] EPANET Project User Manual Network Model Pipe Head Loss Formulas
https://github.com/EPANET-Project/epanet-manual/blob/dev/docs/3_network_model.rst

[7] OpenStax University Physics Volume 1 Section 15 6 Forced Oscillations
https://openstax.org/books/university-physics-volume-1/pages/15-6-forced-oscillations

11 16 Extend a model without losing its meaning

The worked examples can be extended by adding the missing physical effects explicitly: drag to a trajectory, a temperature-dependent device law, a radiative term to a wall, signed losses to a pipe network, or additional

measurements to a parameter fit. Each extension changes the equations and may change the number of solutions. Retain the simpler model as a reference case so that the revised model can be checked in a known limit.

Before accepting an engineering answer, identify what each root represents, which bounds follow from the physics and what independent calculation supports the result. These habits make multiple-solution discovery useful: alternative roots become physical possibilities to examine, rather than rows to accept without interpretation.

Chapter 12 Economics and Other Applications

Economic and data models turn assumptions about prices, production, growth and observations into equations. Patodia Equation Solver can find values that satisfy those relationships, including alternative admissible roots. The interpretation remains part of the modelling task: an equilibrium is a balance under stated assumptions, a break-even point is a zero of profit, and a fitted parameter is conditional on the chosen model.

This chapter develops ten worked examples in the same format as the preceding chapter. Each provides a problem statement, named variables, copyable solver input, reference values, an independent reduction and a location for a screenshot. The examples cover market equilibrium, a tax wedge, interacting markets, break-even output, production requirements, input-output balances, an integer allocation, exponential decay, logistic growth and parameter ambiguity.

All coefficients and observations are constructed for teaching and reproducibility. Monetary units are generic and quantities refer to a stated model period. The tables contain independently calculated reference values, not measured solver timings or claimed test runs. Run each listing separately; var fixes its output order. Extra decimal places support numerical checking rather than empirical accuracy.

12 1 State the economic meaning and admissible domain

Before entering an equation, define what each unknown measures. A unit price differs from total revenue; gross sector output differs from final demand; a growth constant per year differs from a percentage per month. A monetary equation must use the same currency and accounting period throughout. The numerical coefficients below incorporate their assigned units.

Bounds should express the question being asked. A nonnegative price restriction excludes an algebraic root with a negative price. A whole-number declaration is appropriate for indivisible batches, but not automatically for a quantity measured in tonnes. Continuous approximations can be useful, provided their interpretation is explicit.

Market equilibrium means that the specified supply and demand quantities agree [1]. It does not establish stability, efficiency or predictive accuracy. Likewise, solving first-order conditions would locate stationary candidates, not automatically global optima. This chapter uses root and feasibility problems; no built-in profit maximization or statistical estimation capability is assumed.

Quantity	Numerical convention	Domain or interpretation
Price	Currency units per item	Nonnegative in the market examples
Flow quantity	Items or output units per period	Continuous unless declared integer
Revenue and cost	Currency units per period	Same period and currency
Growth rate	Inverse time	Time basis stated in each model
Capacity K	Population units	Above the observed population here
Batch count	Whole batches	Nonnegative integer

12 2 Find a nonlinear supply and demand equilibrium

Let p be price in currency units per item and q the number of items traded per period, treated as continuous. The assigned supply is $q = p^2$ and demand is $q = 30 - p$. Find the equilibrium with nonnegative price and quantity. The coefficients carry the units needed to map the numerical price into quantity.

```
var p,q
q=p^2
q=30-p
bound p=[0,30]
bound q=[0,30]
```

p	q
5.000000000	25.000000000

Eliminating q gives $p^2 + p - 30 = 0 = (p - 5)(p + 6)$. The algebraic pairs are (5,25) and (−6,36). Only the first satisfies the declared domain. Supply and demand both equal 25 at p = 5, so the original equations agree independently of the factorization.

For $p \geq 0$ the reduced function has derivative $2p + 1 > 0$. It changes sign between 0 and 30, establishing exactly one admissible equilibrium. A numerical run that finds one row agrees with this argument; the row count alone would not prove uniqueness.

Replace the demand intercept 30 by A while preserving all other coefficients. The positive equilibrium is $p = (\sqrt{(1 + 4A)} - 1)/2$ for $A > 0$ and $q = p^2$. At A = 30, $dp/dA = 1/11$ and $dq/dA = 10/11$. These are local model sensitivities, not estimates of a real market response.

12 3 Separate buyer and seller prices under a unit tax

Use the preceding market with a hypothetical tax of 2 currency units per item. Let pb be the buyer price, ps the seller receipt per item and q the quantity per period. Assume full compliance, no other wedge and the same supply and demand curves. This is a constructed tax-incidence calculation [2], with no jurisdiction-specific tax rule.

```
var pb,ps,q
q=ps^2
q=30-pb
pb=ps+2
bound pb=[2,30]
bound ps=[0,28]
bound q=[0,30]
```

pb	ps	q
6.815072906	4.815072906	23.184927094

Substitute $pb = ps + 2$ to obtain $ps^2 + ps - 28 = 0$. The admissible seller price is $(\sqrt{113} - 1)/2$; the other root is negative. Buyer price exceeds seller price by exactly 2, and $q = ps^2 = 28 - ps$. The reduced polynomial is strictly increasing for $ps \geq 0$, so this domain contains one root.

Relative to the untaxed price of 5, the buyer price rises by about 1.815073 and the seller receipt falls by about 0.184927. Their sum is the 2-unit wedge. Multiplying the wedge by q gives tax revenue of approximately 46.369854 currency units per period. Both quantity equations use their own side of the transaction price.

These incidence values follow from the assigned curves. They do not follow merely from naming the buyer or seller as the party that remits the tax. The system also contains no welfare calculation, administration cost or adjustment dynamics.

12 4 Solve two interacting markets

Two goods have prices p1 and p2 and quantities q1 and q2 per period. Assign demand q1 = 20 − 2p1 + p2 and q2 = 18 + p1 − 2p2. Assign supply q1 = 2p1 and q2 = 3p2. Positive cross-price coefficients describe substitution within this illustrative linear demand specification.

```
var p1,p2,q1,q2
q1=20-2*p1+p2
q2=18+p1-2*p2
q1=2*p1
q2=3*p2
bound p1=[0,20]
bound p2=[0,20]
bound q1=[0,100]
bound q2=[0,100]
```

p1	p2	q1	q2
6.210526316	4.842105263	12.421052632	14.526315789

Market clearing reduces the system to 4p1 − p2 = 20 and −p1 + 5p2 = 18. Their determinant is 19, so the linear system has a unique solution. Solving gives p1 = 118/19 and p2 = 92/19; supply then gives q1 = 236/19 and q2 = 276/19.

Check each demand directly: 20 − 236/19 + 92/19 = 236/19, while 18 + 118/19 − 184/19 = 276/19. All four coordinates satisfy the positive bounds. Solving the markets separately at an arbitrarily fixed price for the other good would change the problem.

This model permits feedback through prices, but does not include household budgets, income effects or an economy-wide resource balance. Its unique static equilibrium makes no claim about how prices adjust over time.

12 5 Find both break even outputs

A firm faces the assigned inverse demand p = 100 − q and total cost C = 900 + 20q per period. Let q be output, p the selling price and R total revenue, all on a consistent numerical scale. Assume all output is sold at that one price and use 0 ≤ q ≤ 100 to retain nonnegative prices. Find every output at which revenue equals cost.

```
var q,p,R,C
p=100-q
R=p*q
C=900+20*q
R=C
bound q=[0,100]
bound p=[0,100]
bound R=[0,10000]
bound C=[0,10000]
```

q	p	R	C
13.542486889	86.457513111	1170.849737787	1170.849737787
66.457513111	33.542486889	2229.150262213	2229.150262213

Profit is π(q) = R − C = −q² + 80q − 900. Its zeros are q = 40 ± √700, both in the declared domain. Substitution gives R = C at both outputs. A higher output can have lower revenue because the assigned price falls with output.

Completing the square gives π(q) = 700 − (q − 40)². Profit is positive strictly between the two roots, negative outside them and zero at the endpoints. The maximum in this continuous model is 700 at q = 40, p = 60. That maximum follows analytically from the square; the input asks only for break-even roots.

An actual production decision may require capacity, indivisibilities, competitive responses and estimated costs. Break-even roots identify the zero-profit boundaries under the given assumptions. They are not two equally profitable recommendations or a request for the solver to optimize.

12 6 Meet a production target and a fixed budget

An illustrative production function is Q = 2√(KL), with K and L positive numerical units of capital service and labour service per period. Each unit costs 10 and 20 currency units respectively. Find all input combinations producing Q = 40 while spending exactly 600. These are target-and-budget equalities, not a cost-minimization command.

```
var K,L
2*sqrt(K*L)=40
10*K+20*L=600
bound K=[0.001,60]
bound L=[0.001,30]
```

K	L
20.000000000	20.000000000
40.000000000	10.000000000

The output target implies KL = 400 because K and L are positive. The budget gives K = 60 − 2L. Substitution produces L² − 30L + 200 = 0 = (L − 10)(L − 20). Thus the two ordered input combinations are (40,10) and (20,20). Each has KL = 400 and cost 600.

For a further check, along the target curve the cost is C(L) = 4000/L + 20L. Its derivative vanishes at L = √200 and K = 20√2. The minimum cost is 400√2, approximately 565.685425. Since 600 exceeds this minimum, the chosen budget intersects the target curve twice.

A budget equal to that minimum would give one tangent combination; a smaller budget could not meet the target under this model. Squaring the output equation is legitimate here because the square root and target are positive. It would require a separate sign check for a negative target.

12 7 Balance two sectors and final demand

Let x and y be gross outputs of two sectors on a common value scale per period. Sector 1 uses 0.2 units of its own output and 0.1 units of sector 2 output per unit produced. Sector 2 uses 0.3 units of sector 1 and 0.2 units of its own output per unit produced. Final demands are 50 for sector 1 and 40 for sector 2. Intermediate requirements are proportional to gross output.

```
var x,y
x=0.2*x+0.3*y+50
y=0.1*x+0.2*y+40
bound x=[0,200]
bound y=[0,200]
```

x	y
85.245901639	60.655737705

Collecting terms gives 0.8x − 0.3y = 50 and −0.1x + 0.8y = 40. The determinant is 0.61, giving x = 5200/61 and y = 3700/61. Both outputs are positive and below the selected search ceilings.

Check allocation rather than adding final demands alone. Sector 1 intermediate use is 0.2x + 0.3y = 2150/61; adding 50 gives 5200/61. Sector 2 intermediate use is 0.1x + 0.2y = 1260/61; adding 40 gives 3700/61. Gross output includes intermediate requirements, so it exceeds final demand.

The coefficient description fixes the matrix orientation: each production sector is a column of input requirements. Transposing the coefficients would produce a different model. These fixed technical coefficients assume no substitution, no capacity limits and consistent valuation; the calculation does not model price formation.

12 8 Allocate whole production batches

A workshop produces whole batches of products A and B. One A batch consumes 3 machine-hours and 2 labour-hours; one B batch consumes 2 machine-hours and 4 labour-hours. A schedule must use exactly 30 machine-hours and 40 labour-hours. Let a and b be nonnegative integer batch counts. Find a feasible schedule.

```
var a,b
int a,b
3*a+2*b=30
2*a+4*b=40
bound a=[0,10]
bound b=[0,10]
```

a continuous batches	b continuous batches
5	7.5 is not integer

The continuous equations imply a + 2b = 20. Subtracting this from 3a + 2b = 30 gives a = 5 and then b = 7.5. The coefficient determinant is 8, so there is no other continuous solution. Consequently there is no integer schedule satisfying both exact-use conditions. The table shows the excluded continuous candidate, not a solver solution.

The proper expected integer result is no admissible solution. Rounding b to 8 uses 31 machine-hours and 42 labour-hours; rounding to 7 uses 29 and 38. Neither satisfies the original equalities. This illustrates why integer declarations can change feasibility even for a small linear system.

If available hours are capacities rather than exact-use requirements, replace the equalities by inequalities. That revised model can admit many schedules, including (4,8), which uses 28 machine-hours and 40 labour-hours. Selecting among feasible schedules needs an objective and an optimization method; a feasible root is not automatically the most profitable schedule.

12 9 Recover exponential decay parameters

An exact synthetic response follows y(t) = a exp(bt). Its values are y(0) = 2 and y(2) = 2exp(−1). Time is in days and y is in response units, so a has response units and b has units day^{-1}. Find a positive amplitude and a nonpositive rate using the explicit exponential form [3].

```
var a,b
a=2
a*exp(2*b)=2*exp(-1)
bound a=[0.001,10]
bound b=[-2,0]
```

a	b per day
2.000000000	-0.500000000

The first observation fixes a = 2. Division by 2 gives exp(2b) = exp(−1). The real exponential is one-to-one, so b = −0.5 is the unique admissible rate. This model predicts y(4) = 2exp(−2), approximately 0.270670566.

The half-life solves exp(bt) = 1/2, giving t = −ln(2)/b = 2ln(2), approximately 1.386294361 days. Written in hours, the same response uses b = −0.5/24 per hour. The exponential argument must remain dimensionless.

With more noisy observations than unknowns, exact simultaneous equalities are generally inconsistent. Least squares instead chooses parameters to minimize an error objective. This exact-data example supplies a reproducible reference without asserting that a root finder performs regression or quantifies measurement uncertainty.

12 10 Recover logistic growth parameters

Let N(0) = 1 population unit and use the explicit logistic response N(t) = K/[1 + (K − 1)exp(−rt)]. The exact synthetic observations are N(1) = 20/11 and N(2) = 40/13. Time is in years, r is in year^{-1} and K is the carrying capacity in population units. The standard logistic form [4] is evaluated algebraically, without numerical time integration.

```
var K,r
K/(1+(K-1)*exp(-r))=20/11
K/(1+(K-1)*exp(-2*r))=40/13
bound K=[4,100]
bound r=[0.001,3]
```

K	r per year
10.000000000	0.693147181

Put u = exp(−r), so 0 < u < 1. The first observation gives (K − 1)u = 11K/20 − 1, and the second gives (K − 1)u² = 13K/40 − 1. Eliminating u yields (11K/20 − 1)² = (K − 1)(13K/40 − 1). Expansion reduces this to 9K(K − 10)/400 = 0.

Only K = 10 lies in [4,100]. Substitution gives u = 1/2 and r = ln(2). The excluded algebraic value K = 0 cannot reproduce the positive observations in the original equations. The positive denominators throughout the stated domain also rule out poles. These checks establish exactly one admissible parameter pair.

The next prediction is N(3) = 10/[1 + 9/8] = 80/17, approximately 4.705882353. The bound K ≥ 4 is an assumption above the observations, not a general law of logistic models. Two exact points can identify this constructed model while still giving weak empirical support for its long-term capacity in real data.

12 11 Distinguish parameter pairs with identical summaries

Two positive coefficients a and b are known to have product 12 and sum 7. They are labelled parameters, so exchanging their values gives a different ordered vector unless the application explicitly treats the labels as interchangeable. Find every pair within [1,6].

var a,b
a*b=12
a+b=7
bound a=[1,6]
bound b=[1,6]

a	b
3.000000000	4.000000000
4.000000000	3.000000000

Eliminate b = 7 − a to get $a^2 - 7a + 12 = 0 = (a - 3)(a - 4)$. Both resulting ordered pairs satisfy the two original equations. These are exactly the two roots in the domain. Their shared product and sum do not distinguish which label receives the larger value.

If independent application knowledge requires a < b, only (3,4) remains. That restriction adds information; it should not be inserted merely to hide an alternative root. An independent labelled observation, such as 2a + b = 10, likewise selects (3,4).

If the sum observation were replaced by 2ab = 24, the two equations would be dependent. Within the same bounds, every pair b = 12/a with $2 \le a \le 6$ would satisfy them. Bounds make that continuum finite in extent but do not turn it into finitely many identifiable parameter pairs.

12 12 Verify feasibility root counts and interpretation

Substitute each candidate into every original equation and check the declared bounds and integer conditions. In market models, verify both sides at the appropriate price; in accounting models, reconstruct each use of output; in parameter models, reproduce each original observation. Residuals in money, quantity and response units are not directly comparable without an explicit scaling convention.

The independent reductions establish one admissible root for Examples 1, 2, 3, 6, 8 and 9; two for Examples 4, 5 and 10; and none for the integer formulation in Example 7. These counts apply to the exact inputs and stated domains. A search budget that returns fewer roots does not invalidate the algebraic reference count, and no timing or completeness certification is inferred from these tables.

A small residual addresses internal equation consistency. It does not show that estimated demand is causal, that fixed production coefficients remain valid after a large change, or that a fitted growth curve forecasts reliably. Keep parameter uncertainty, model choice and numerical precision separate. Additional digits cannot repair missing information or an unsuitable functional form.

For each screenshot, show the input and solution columns at a readable size. Use actual residuals, search settings and timings from the run. Record the stopping reason when no admissible row is found. The integer example has an algebraic infeasibility argument, so a screenshot of an unsuccessful search is supporting documentation rather than the proof.

12 13 Practice problems

Problem 1 Shift demand

In Example 1 change demand to q = 42 − p and extend the quantity bound to [0,42]. Find the admissible equilibrium and the excluded algebraic price.

Problem 2 Increase the unit tax

In Example 2 use a tax of 4 instead of 2. Find both prices, quantity and tax revenue.

Problem 3 Change final demand in a market

In Example 3 replace the demand intercept 20 by 24. Keep the other demand and supply coefficients fixed. Find all four equilibrium coordinates.

Problem 4 Raise fixed cost

In Example 4 replace fixed cost 900 by 1600, and then by 1700. Determine how many break-even outputs lie in [0,100].

Problem 5 Reduce the production budget

In Example 5 reduce the budget to 500. Is the output target attainable? Give an independent reason.

Problem 6 Increase sector final demand

In Example 6 increase final demand for sector 1 from 50 to 60 while holding sector 2 final demand at 40. Find gross outputs.

Problem 7 Restore integer feasibility

In Example 7 replace exact machine-hour use 30 by 32 and keep labour use 40. Find the integer schedule.

Problem 8 Convert the decay time basis

In Example 8 express b per hour and calculate the response after six days.

Problem 9 Predict logistic growth

In Example 9 calculate N(4) and solve for the year at which N = 5.

Problem 10 Add a labelled observation

In Example 10 add 2a + b = 10. Which ordered pair remains? Explain why ab = 12 and 2ab = 24 alone cannot identify the pair.

12 14 Worked answers

Answer 1 Shift demand

Elimination gives $p^2 + p - 42 = (p - 6)(p + 7)$. The admissible pair is p = 6, q = 36. The other price is −7 and violates the domain; its algebraic quantity is 49. Extend the original quantity ceiling of 30 to include the revised equilibrium; otherwise the old bound would exclude q = 36.

Answer 2 Increase the unit tax

The seller equation is $ps^2 + ps - 26 = 0$. Thus $ps = (\sqrt{105} - 1)/2 \approx 4.623475383$, $pb = ps + 4 \approx 8.623475383$ and $q = 26 - ps \approx 21.376524617$. Tax revenue is $4q \approx 85.506098468$ per period. The negative seller-price root is inadmissible.

Answer 3 Change final demand in a market

The reduced equations become $4p1 - p2 = 24$ and $-p1 + 5p2 = 18$. Therefore $p1 = 138/19 \approx 7.263157895$ and $p2 = 96/19 \approx 5.052631579$. Quantities are $q1 = 276/19 \approx 14.526315789$ and $q2 = 288/19 \approx 15.157894737$.

Answer 4 Raise fixed cost

With fixed cost 1600, profit is $-(q - 40)^2$. There is one distinct break-even output $q = 40$, with $p = 60$ and $R = C = 2400$. The repeated polynomial root is one distinct output. With fixed cost 1700, profit is $-100 - (q - 40)^2 < 0$ for every real q, so no break-even output exists.

Answer 5 Reduce the production budget

No. Along $KL = 400$, minimum cost is $400\sqrt{2} \approx 565.685425$, above 500. Equivalently, substituting $K = 50 - 2L$ gives $2L^2 - 50L + 400 = 0$, whose discriminant is $2500 - 3200 = -700$. There is no real target-and-budget intersection.

Answer 6 Increase sector final demand

The right sides of the reduced sector equations become 60 and 40. Thus $x = 6000/61 \approx 98.360655738$ and $y = 3800/61 \approx 62.295081967$. Relative to the original outputs, the increases are 800/61 and 100/61; both sectors respond because intermediate requirements link them.

Answer 7 Restore integer feasibility

From $a + 2b = 20$ and $3a + 2b = 32$, obtain $a = 6$ and $b = 7$. Check machine use $18 + 14 = 32$ and labour use $12 + 28 = 40$. Both counts are integers within the stated bounds.

Answer 8 Convert the decay time basis

The hourly rate is $-1/48 \approx -0.020833333$ $hour^{-1}$. After six days, or 144 hours, $y = 2\exp(-3) \approx 0.099574137$. Both time bases give the same dimensionless exponent.

Answer 9 Predict logistic growth

$N(4) = 10/[1 + 9/16] = 160/25 = 6.4$. Setting $N = 5$ gives $\exp(-rt) = 1/9$, so $t = \ln(9)/\ln(2) \approx 3.169925001$ years. This is the midpoint population, not the time to reach the asymptotic carrying capacity.

Answer 10 Add a labelled observation

The pair (3,4) gives $2a + b = 10$; (4,3) gives 11 and is excluded. In contrast, $ab = 12$ and $2ab = 24$ express the same information. Within the original bounds, $b = 12/a$ for $2 \le a \le 6$ is a continuous family, not an identified pair.

12 15 References for the model forms

The following sources support the standard equilibrium, tax-incidence and growth concepts. All numerical examples, parameter choices, root-count arguments and exercises in this chapter are constructed independently. The production and accounting examples are defined explicitly by their equations rather than presented as empirical estimates.

[1] OpenStax Principles of Economics 3e Section 3 1 Demand Supply and Equilibrium in Markets for Goods and Services

https://openstax.org/books/principles-economics-3e/pages/3-1-demand-supply-and-equilibrium-in-markets-for-goods-and-services

[2] OpenStax Principles of Economics 3e Section 5 3 Elasticity and Pricing
https://openstax.org/books/principles-economics-3e/pages/5-3-elasticity-and-pricing

[3] OpenStax Calculus Volume 2 Section 2 8 Exponential Growth and Decay
https://openstax.org/books/calculus-volume-2/pages/2-8-exponential-growth-and-decay

[4] OpenStax Calculus Volume 2 Section 4 4 The Logistic Equation
https://openstax.org/books/calculus-volume-2/pages/4-4-the-logistic-equation

12 16 Extend a model with evidence and independent checks

Extend one feature at a time. A demand shift can be studied by changing a single intercept; a production extension can add a stated capacity; a growth fit can be challenged with a new observation. Preserve the original case as a regression reference and record the equations, units, bounds and expected root count for every variant.

When observations are noisy, choose an estimation framework that represents their errors. When a decision requires the best feasible alternative, formulate the objective and use appropriate optimization methods. When an equation admits several roots, retain their labels and interpret each one before selecting a branch. These distinctions let equation solving contribute a precise, reviewable calculation to a broader modelling task.

Chapter 13 Pell Equations and Very Large Integer Bounds

13 1 A recurrence explains the returned sequence

Begin with x^2-2*y^2=1 over positive integers. The identity (x+y*sqrt(2))*(x-y*sqrt(2))=1 suggests multiplying known unit expressions. The smallest positive nontrivial unit is 3+2*sqrt(2). If (x,y) is a solution, multiplication by that unit gives X=3*x+4*y and Y=2*x+3*y. Direct expansion verifies X^2-2*Y^2=x^2-2*y^2. No decimal approximation to sqrt(2) is required to check the recurrence.

Starting at (1,0) gives (3,2), (17,12), (99,70), (577,408), (3363,2378) and (19601,13860). The initial pair belongs to the nonnegative family but is excluded by pos int y. This small distinction explains many apparent disagreements between a textbook list and a Desktop result. Always state whether the zero branch and signed reflections are included.

For D=31 the fundamental positive pair is (1520,273). Verification gives 1520^2=2310400 and 31*273^2=2310399. The next recurrence is X=1520*x+8463*y and Y=273*x+1520*y. The enormous jump between successive solutions is a mathematical property, not evidence that the engine skipped all intermediate integer values by accident. A real curve can have sparse lattice intersections.

13 2 Bounds select members of an infinite family

A complete finite list needs a finite selection rule. For positive D=2 solutions, x increases under the recurrence. Thus the first four pairs and a proof that the next x is 3363 establish the positive set with x<=1000. Merely asking for four rows does not establish that the set was exhausted; the same prefix can be produced by a row cap on an unrestricted sequence.

Compare three experiments: bound x=[1,1000] with a generous row cap, no explicit upper bound with Maximum solutions=4, and bound x=[100,1000]. They ask different questions. The first requests four points in a finite interval, the second requests a prefix, and the third returns only (577,408). The reason and integer completion field must correspond to those domains.

The 2.0.1 literal-bound facility permits 1e500 without converting it to a floating-point infinity. It compares decimal values on eligible exact routes and rounds nonintegral endpoints inward. For an integer upper endpoint 17.9, x<=17.9 means x<=17; a lower endpoint 17.1 means x>=18. For positive Pell D=2, [17.1,99] therefore retains (99,70) but excludes (17,12). This rounding should be understood mathematically before checking its implementation.

An interval around a 500-digit solution is not equivalent to requesting 500 solutions. Digit count measures coordinate magnitude; row count measures how many points were returned. Export full JSON rather than counting visible digits in a clipped cell. When independently checking huge values, use an arbitrary-size integer type or exact decimal arithmetic with enough capacity, never a standard spreadsheet numeric cell.

13 3 Generalised right hand sides and sign branches

For x^2-2*y^2=-1, (1,1), (7,5) and (41,29) are positive examples. Multiplying by 3+2*sqrt(2) preserves the right-hand side -1 exactly as it preserves +1. For general N, several inequivalent seed orbits can occur; blindly applying the +1 recurrence to one chosen point does not prove all solutions have been generated.

The integer discriminant and square cases change the structure. If D=0, x^2=N constrains x while y can be free. If D is a positive square, x^2-D*y^2 factors over integers and reduces to divisor conditions or degenerate lines. If N=0, families can differ from the familiar nonsquare +1 problem. These are reasons to read the selected route rather than using the word Pell as a guarantee of one algorithm.

Signed integer declarations can include (x,y), (-x,y), (x,-y) and (-x,-y). If y=0, some reflections coincide. Lexicographic output order depends on declared variable order and the route's sorting; it is not always recurrence order in a signed model. Count pairs with labels and deduplicate only mathematically identical coordinates.

13 4 Independently check a huge answer

A robust check parses the returned x and y strings as integers, evaluates x*x-D*y*y, compares with N and checks the inclusive bounds and domain signs. Also compare with an independently generated recurrence when the fundamental unit and seed orbit are established. Exact substitution detects a corrupted digit; a recurrence check detects an omitted or unexpected sequence member.

For a standard positive +1 family, retain the predecessor and successor around the requested interval. Those neighbours help verify that the sequence crosses each endpoint correctly. An exact endpoint hit is especially valuable: choose x equal to one known recurrence coordinate and verify inclusion. Then move the endpoint just beyond it and verify exclusion. A rounded floating-point reference cannot test that boundary meaningfully for 1000-digit coordinates.

The Desktop's long-value viewer is a display aid with an unresolved clipping report. It should not be used as the mathematical verifier. The raw value, project bound text, JSON export and independent integer check together provide more reliable evidence. Preserve the current Ctrl+C behaviour when replacing screenshots; do not insert formatting separators into the underlying coordinate.

13 5 Practice with worked answers

1. Generate the next D=2 pair after (99,70). Applying the recurrence gives (577,408); substitution gives 577^2-2*408^2=1.

2. List the positive D=2 pairs with 10<=x<=100. They are (17,12) and (99,70). The preceding x is 3 and the succeeding x is 577.

3. Decide whether (1,0) belongs under pos int x,y. It does not because y must exceed zero. Under nonneg int it is admissible.

4. Explain why negative x values do not create a new positive orbit. They are sign reflections, excluded by the positive declaration, though they are distinct signed pairs.

5. Verify (7,5) for N=-1. The difference is 49-50=-1. It is not a +1 solution and must not be accepted under an incorrect target.

6. Describe the exact integer domain of x in [1.2,17.9]. It is the integers from 2 through 17; inward rounding does not invent a noninteger candidate.

7. Is a complete positive enumeration of x<=1000 also a complete real root set of the conic? No. The real curve has infinitely many points; integer and real completeness are different statements.

8. If only the first returned row is needed, why retain the stopping reason? It identifies whether that row is an intended sequence prefix, the only point in the selected interval or part of an interrupted search.

13 6 A laboratory record for this chapter

Run the bounded standard example, the -1 example and an endpoint-inclusion test in both the model-text and dialog-bound forms. Save each as a separate project. Export JSON and verify the identity with exact arithmetic. Record returned count, applicable completion flag, time limit and row cap. Repeat one case with signed declarations and predict its reflections before solving.

The exercise is not a race to the largest decimal string. It shows how arithmetic structure makes enormous coordinates manageable while still requiring explicit domains, resource limits and independent checks. Decimal size alone does not predict the difficulty of general N or factorisation-dependent reductions.

13 7 Distinguish several unit orbits in one problem

13 8 The target seven does not follow just one seed

For x^2-2*y^2=7, both(3,1) and(5,3) are positive seeds. Multiplication by3+2*sqrt(2) takes(3,1) to(13,9), then(75,53), while the other seed gives(27,19), then(157,111). Interleaving by x produces3,5,13,27,75,157,... . A recurrence started only at(3,1) misses the entire second orbit.

```
pos int x,y
x^2-2*y^2=7
bound x=[1,200]
```

The six positive pairs with x<=200 are(3,1),(5,3),(13,9),(27,19),(75,53),(157,111). The next x values from those two streams are437 and915 respectively, so these branches produce no further points below200. A complete proof also needs the representative-orbit argument that these are the relevant families; finding two seeds experimentally is not itself that proof.

For an independent small check, enumerate y from 1 to141, because x<=200 implies2*y^2=x^2-7<40000. Compute2*y^2+7 exactly and test for an integer square x in range. This exhaustive bounded reference does not reproduce the production orbit algorithm and is therefore useful as a separate test.

Change the target to x^2-61*y^2=2. Modulo four,61 is1 and square differences can be0,1 or3, never2. This proves impossibility for all integers, not just a bounded positive box. Compare the reason with a deliberately row-capped successful target: both may show few rows, but only one has an impossibility argument.

13 9 Document the arithmetic rather than a decimal approximation

The expression sqrt(D) explains the algebraic unit, but exact verification should use integer formulas. Rounding sqrt(2) and calculating large real powers can lose the tiny difference between x^2 and2*y^2 that defines the Pell identity. Cancellation becomes extreme as coordinates grow, even though the integer recurrence remains exact.

Record D,N,seed/family information where independently known, inclusive bounds and the exact returned strings. A machine-size integer overflow in a verifier can invalidate its conclusion without affecting the solver. Confirm that the verifier's arithmetic type can represent the squared coordinates, not merely the original coordinate length.

13 10 Additional questions

Explain why swapping x and y does not preserve x^2-2*y^2=7 in general. Compare the two orbits with a maximum of three rows and explain which completion claim remains possible. Move the upper bound to13 and then to12.999: the pair(13,9) is included in the first and excluded in the second. These tests distinguish exact endpoint semantics from approximate display formatting.

13 11 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

13 12 Usage

```
pos int x, y
x^2 - 61*y^2 = 3
```

The smallest positive solution is (8,1). Other solutions begin (335159612,42912791) and (27925945172,3575550889).

```
pos int x, y
x^2 - 61*y^2 = 2
```

This has no integer solutions: squares modulo 4 are 0 or 1, and x^2-61*y^2 cannot be 2 modulo 4. The engine reports this modular proof.

```
pos int x, y
x^2 - 2*y^2 = 7
```

This example needs multiple unit-orbits. Its first solutions are (3,1), (5,3), (13,9), and (27,19). The engine merges the families in increasing positive x order and removes exact duplicates.

CLI:

```
equation_solver.exe --eq "pos int x,y; x^2-61*y^2=3" --max-solutions 5
```

13 13 Scope

- One equation x^2-D*y^2=N with two variables restricted to strictly positive integers, through declarations or positive bounds.

- Positive integer D; positive, negative or zero integer N. Coefficients use the existing checked signed-64-bit polynomial parser. Write coefficients as integer literals, not decimal approximations.

- Rearranged/scaled equivalents and either variable declaration order, plus existing implicit multiplication notation such as 61y^2.

- General quadratic forms with cross terms or shifted variables, extra equations/inequalities, and negative/zero variable branches are not added in this specialised path. Unsupported models retain the existing algorithms; their solutions/completeness must not be inferred from this Pell shortcut.

- N=1 retains the fast standard Pell algorithm, including large D such as 1621. N=-1 has a direct continued-fraction route and an exact impossibility check when the negative Pell equation is unsolvable.

- For nonsquare D, N=0 has no strictly positive integer solution.

- For square D and nonzero N, the positive solution set is finite and is enumerated using an exact finite scan bound from the factor identity.

- For square D and N=0, the engine generates the line x=sqrt(D)*y directly.

All returned coordinates are exact arbitrary-precision decimal strings. Every returned pair is checked by exact integer arithmetic; residual 0 is an exact Pell identity, not a floating-point tolerance test. No external big-integer library is required.

13 14 General N algorithm and completeness

1. Necessary modular-residue checks reject proven impossibilities. Passing these checks is not treated as a proof that a solution exists.

2. Manageable bounded ranges are scanned exactly. Large bounds instead use exact unit-orbits when their reduced families can be established.

3. For unbounded nonsquare D and nonzero N, the positive fundamental unit epsilon=u+v*sqrt(D), u^2-D*v^2=1, comes from continued fractions.

4. Every solution alpha=x+y*sqrt(D)>0 can be reduced by a power of epsilon into sqrt(|N|/epsilon) <= alpha <= sqrt(|N|*epsilon). This gives finite representative bounds:

```
N>0: 2D*y^2 <= N*(u-1), allowing both signs of reduced y.
```

```
N<0: 2D*y^2 <= |N|*(u+1), allowing both signs of reduced x.
```

The engine scans nonnegative coordinates within those bounds, then reflects the opposite orientation through epsilon. Zero-coordinate representatives are included, so imprimitive/square-N families are not discarded.

5. Each family advances by

```
x_next = u*x + D*v*y
y_next = v*x + u*y
```

The streams are merged by exact coordinate comparisons. All matching stream heads advance together, avoiding duplicate output. Explicit bounds remain in force; a family is exhausted when a growing coordinate exceeds an upper bound.

The code does not assume that one starting solution generates every family. If a resource guard prevents complete representative discovery, it does not emit later recurrence powers that might skip smaller undiscovered solutions. Any returned direct-scan prefix remains exact and ordered; the run is labelled incomplete. A completed empty representative search proves no positive solutions, but an interrupted/limited empty search does not.

Numeric decimal/scientific bounds are rounded inward exactly, even beyond binary128's significand precision. Other bound expressions use the existing parser's evaluated arithmetic. Input bound values still need to fit that parser's finite range; automatically generated output is not restricted to double's exponent range.

13 15 Limits and reporting

All existing time/cancellation/solution limits remain active. Added guards:

- 1,000,000 rows for exact direct or reduced-representative scans.

- 10,000 reduced representatives.

- 100,000 continued-fraction or orbit-merge steps.

- 10,000 decimal digits per generated coordinate.

A hard equation can reach these guards. The engine explicitly reports incomplete search in that case; support for the form does not guarantee that every arbitrary D,N instance will finish within the selected budget.

For nonsquare D and nonzero N, an existing positive solution generates an infinite unbounded family. The engine returns a prefix up to the requested limit, or completes enumeration within explicit upper bounds when possible.

JSON/DLL output:

- precision = exact_integer

- search_arithmetic = exact_integer_pell

- exact_integer_solutions = true for this exact path

- positive_integer_search_complete = true only for a completed positive-domain exclusion or enumeration, not a time/solution/resource stop

The existing search_complete field keeps its interval-certification meaning; it is not repurposed to claim that all real points on a Pell curve were found. C ABI version 1 and result access functions remain unchanged.

The existing desktop displays/exports integer strings without losing digits. Its generic report footer retains numerical-search wording; for these runs, values are exact, and the Pell stop reason/positive-domain flag state the actual completeness result.

13 16 Exact bounds independent of floating precision

```
pos int x,y
x^2-2*y^2=1
bound x=[1e499,1e500]
```

This returns the positive Pell points whose x is in the exact inclusive interval [10^499,10^500]. It works in ordinary double mode; no extended entitlement or floating-point approximation to the bound is required. Scientific literals mean exact decimal powers here. Full decimal endpoints are accepted too.

Model-text bounds, the Variables and Bounds dialog, project save/reopen, CLI --bound/--bounds and the unchanged C ABI options preserve these literals. Original sign declarations continue to intersect bounds. Lower endpoints are rounded upward and upper endpoints downward by the exact integer routines. Comparison of literal endpoints is exact even when floating representations would be indistinguishable. A lower endpoint greater than the upper is rejected.

The beyond-floating-range capability is deliberately limited to recognised exact two-variable integer quadratic, positive Pell and positive weighted-square paths. Unsupported numerical, real, mixed-integer, modulo or higher-dimensional models must not silently receive truncated or infinite bounds: oversized bounds are rejected with a specific explanation. A bound override replaces the model-text bound according to the existing precedence rules. Normal finite real bounds and constant-expression bounds retain their previous numerical semantics.

Arbitrary-magnitude integer bounds require simple decimal/scientific literals. Expressions such as 10^500 or (1e500) are not a new arbitrary-precision expression language. Equation coefficients still have their existing parser/recogniser limits; this specialised path concerns bounds, not unbounded-size input coefficients. The literal has a 10000-character guard and a magnitude limit of 10000 digits, with exponent range +/-10000. Existing per-method time, step, factorisation, solution-count and generated-coordinate limits remain.

A bound limits magnitude, not how many rows should be displayed. For only the smallest nontrivial positive Pell pair, use pos int x,y and Maximum solutions=1. For every pair in a bounded domain, choose a sufficiently large row cap and read integer_search_complete. int x,y also permits zero and negative branches. Solution-cap stops now say enumeration incomplete without implying that an explicit upper bound was absent.

13 17 Desktop large integer workflow

13 18 Pell equations

```
pos int x,y
x^2-2*y^2=1
bound x=[1,1000]
```

The positive pairs in this x interval are (3,2), (17,12), (99,70) and (577,408). The pos int declarations exclude the trivial y=0 branch. Use int if negative and zero coordinates are intended. The generalised form x^2-D*y^2=N is handled for supported integer inputs through exact Pell/quadratic methods.

For only the smallest nontrivial positive pair, set Maximum solutions to 1 with the intended positive domain, and inspect the reason. A bounded result list is not a complete symbolic description of every member of an infinite Pell family.

13 19 Large literal bounds

```
pos int x,y
x^2-2*y^2=1
bound x=[1e499,1e500]
```

Choose Examples > Large Integer Values for this model. The interval endpoints are exact powers of ten on recognised exact two-variable integer routes. Ordinary double mode is sufficient; binary128 is not needed merely to represent the returned integer digits. Both dialog bounds and .pdeq projects preserve supported large bound text.

Use simple decimal or scientific literals such as 1e500, rather than 10^500 or (1e500), for arbitrary-magnitude bound endpoints. This facility is limited to recognised exact quadratic, positive Pell and positive weighted-square routes. It does not make every real or mixed-integer model accept enormous bounds. Unsupported cases are rejected instead of silently replacing the bounds by infinity.

Input and computation guards remain, including a 10000-character literal guard, 10000 magnitude digits, exponent bounds and per-method time, count and factorisation limits. Large equation coefficients have separate parser/recogniser constraints; large-bound support is not an unrestricted arbitrary-precision expression language.

Chapter 14 General Quadratic Equations in Integers

14 1 Read a conic as a lattice problem

Consider A*x^2+B*x*y+C*y^2+D*x+E*y+F=0 with all six coefficients integers and both unknowns restricted to integers. Over the reals this is a conic or a degenerate quadratic set. Over the integers the task is to find lattice points on that set. A smooth real ellipse can have no integer point, and an unbounded hyperbola can have a sparse infinite arithmetic family.

The sign of B^2-4*A*C is a useful structural clue, not a complete solver decision by itself. A negative discriminant suggests a definite quadratic part; a positive one suggests an indefinite part; zero raises parabolic or degenerate possibilities. Linear terms and constant offsets still affect existence, boundedness and recovery. Multiplying an equation by a nonzero integer changes its presentation without changing the original root set.

Begin with x^2+x*y+y^2=7. Completing a square gives (2*x+y)^2+3*y^2=28. Therefore |y|<=3, leaving only seven possible y values to test. This turns an apparent two-dimensional search into a transparent finite enumeration. It also illustrates why a computed exact reduction must retain the congruence 2*x+y having the same parity as y.

14 2 Derive all twelve pairs

For y=0 the equation requires x^2=7, so there is no integer x. For y=1, x^2+x-6=0 gives x=2 or -3. For y=-1, x^2-x-6=0 gives x=3 or -2. For y=2 the roots are 1 and -3; for y=-2 they are 3 and -1. For y=3 the roots are -1 and -2; for y=-3 they are 1 and 2. Values |y|>=4 are impossible by the completed-square bound.

There are twelve ordered integer pairs. Each should be checked in the original equation, not only the transformed ellipse. Using pos int instead of int keeps (2,1) and (1,2); it does not keep all twelve. A symmetry argument explains the count but does not replace domain filtering.

14 3 Reduce an indefinite quadratic to Pell form

The equation x^2+2*x*y-30*y^2=1 becomes (x+y)^2-31*y^2=1 by setting u=x+y. Its fundamental positive u,y pair is (1520,273), giving x=1247. Direct substitution in the original mixed-term equation verifies the recovered coordinate. Sign choices for u do not translate into merely changing the sign of x, because x=u-y.

For y=0 the equation gives x=1 or -1. Those zero-coordinate branches must be retained under unrestricted int declarations. With a finite y bound, Pell recurrences can identify the permitted branches exactly. Without a bound an infinite family cannot be replaced by a finite list and labelled complete. A solution cap is a computational request, not a mathematical boundary.

This recovery issue is common in general quadratic reductions. A transformed integer point can fail parity or divisibility when mapped back. Exact methods retain the required congruences, test integrality and substitute into the original polynomial. A real-variable change of coordinates is not automatically a bijection of integer lattices.

14 4 Factor a bilinear form without losing a line

For x*y+2*x+3*y=6, adding six to both sides gives (x+3)*(y+2)=12. Every signed divisor d of 12 yields x=d-3 and y=12/d-2. If positive x,y are required, the admissible factors are further restricted. At d=4, x=1,y=1; at d=6, x=3,y=0 and positivity rejects it. This shows why exact factor enumeration must be followed by the original domain tests.

Now change the right-hand side to -6. The transformed product is zero, so x=-3 or y=-2. There are two complete lines of integer solutions and their intersection must not be double-counted. Dividing by x+3 would erase an entire branch. Finite bounds make enumeration possible; an unrestricted finite scan does not describe every point on those lines.

This is the same modelling principle encountered in polynomial factor examples, but the arithmetic route can use it directly. A fast answer is justified by a finite divisor argument when the product is nonzero, while a zero product calls for a different completeness statement. Both forms should appear in an acceptance study.

14 5 Parabolic cases and scan bounds

The integer parabola y=x^2 has one point for each integer x. A bound -3<=x<=3 selects seven points: (-3,9), (-2,4), (-1,1), (0,0), (1,1), (2,4), (3,9). It does not require a separate y bound because the equation determines y for every bounded x. A bound on y instead selects x through a square-root restriction; whether the implemented route proves and uses that restriction must be established rather than assumed.

Without a finite scan interval, some parabolic/degenerate cases use an explicitly limited window. The window is a search convention, not a newly discovered theorem about the user's domain. Read the stopping reason and do not describe an arbitrary sample window as the complete integer solution set. If an application needs every point, formulate a finite justified interval or derive a symbolic parameter family independently.

14 6 Necessary congruence tests

Reducing a polynomial modulo a small integer can prove impossibility cheaply. Since an integer square is 0 or 1 modulo four, x^2+y^2=3 is impossible over integers. For x^2+2*y^2=5, modulo two forces x odd; modulo four then forces y even. Modulo eight finishes this case: x^2+2*y^2 cannot have residue five, so there is no integer solution. Necessary conditions remove candidates; passing them does not establish existence.

A complete finite proof can combine such tests with a bound. For x^2+y^2=25 in nonnegative integers, x lies from zero through five. Testing squares gives (0,5), (3,4), (4,3), (5,0). Restricting both coordinates positive leaves two. In signed integers, sign reflections and axis points yield twelve. State ordered versus unordered and sign conventions before counting.

14 7 Practice and independent checks

1. Solve x^2+x*y+y^2=1 in integers. The six pairs are (1,0), (-1,0), (0,1), (0,-1), (1,-1), (-1,1). The completed-square bound limits |y| to one.

2. Count y=x^2 with -2<=x<=2. There are five pairs; the repeated y value at opposite x does not make the ordered vectors identical.

3. Solve (x+1)*(y+1)=6 in positive integers. Positive factors greater than one give (1,2) and (2,1). The factors 1 and 6 would produce a zero coordinate and are rejected.

4. Describe (x+1)*(y+1)=0 without bounds. It is the union of x=-1 and y=-1. Their crossing is one point in both descriptions, not two distinct solutions.

5. Explain why x^2+y^2=3 has no integer solution. Modulo four the left side can be 0,1 or 2, while the right side is 3.

6. Recover x from u=x+y for u=1520,y=273. The result is 1247. Replacing x by u would solve a different equation.

7. Does an exact returned point imply every conic lattice point was returned? No; exact verification and enumeration completeness are independent fields of evidence.

8. Why are coefficient gcd and exact division useful? They simplify the identity without rounding, while a division with a remainder is not an integer-coefficient equivalence of the recognised form.

14 8 A conic comparison project

Prepare three models representing an ellipse, hyperbola and parabola. For each, derive a small independently complete finite reference set. Compare unrestricted, positive and bounded declarations. Then alter a coefficient to

make one model impossible modulo four. Retain the model, project settings, raw JSON and independent enumeration.

Report which reductions are bijections on integer pairs and which need divisibility recovery. This distinction is a more valuable research result than a timing table alone. It explains why two algebraically similar forms can require different computational paths without implying different mathematics.

14 9 Coordinate transformations and complete finite references

14 10 Translate the centre without losing integer information

The circle (x-2)^2+(y+1)^2=5 has centre(2,-1), but its lattice problem is equivalent to u^2+v^2=5 with u=x-2,v=y+1. Because the shifts are integers, the transformation is a bijection on integer pairs. The eight signed(u,v) choices are(±1,±2) and(±2,±1), giving eight distinct original pairs.

```
int x,y
(x-2)^2+(y+1)^2=5
```

For example u=1,v=2 gives x=3,y=1, while u=-2,v=-1 gives x=0,y=-2. Both satisfy the shifted original equation. This is a useful Desktop test of expanded and factored forms: expand to x^2+y^2-4*x+2*y=0 and verify that it returns the same labelled pairs. Keep the domain declarations identical during the comparison.

A half-integer shift behaves differently. Completing a square in x^2+x+y^2=2 introduces x+1/2, which is not an integer coordinate. Multiplying by four yields(2*x+1)^2+(2*y)^2=9. The first transformed coordinate must be odd and the second even. Enumerating all integer points on the radius-three circle and ignoring those parity requirements would create invalid recovery candidates.

This example illustrates why a general conic recogniser stores exact congruences alongside its transformed equation. The square-completion formula alone is necessary but not sufficient for integer equivalence. The same concern appears in discriminant-based recovery: a perfect-square discriminant must also satisfy divisibility by2*A before an integer x is accepted.

14 11 A reproducible small reference enumerator

For a finite x interval, treat the original quadratic as a quadratic or linear equation in y at each integer x. If the y^2 coefficient is nonzero, compute its integer discriminant and test whether it is a nonnegative perfect square. Then check both numerator signs and divisibility of the root formula. If that coefficient vanishes, handle the linear or identically zero cases separately rather than dividing by zero.

The identically zero case can leave every bounded y admissible at a particular x. This is a degenerate slice, not a failure of the quadratic formula. A reference checker must include it when claiming completeness. Similarly, an equation that reduces to a constant nonzero residual has no roots on that slice, but this local result does not classify other x values.

An independent brute-force rectangular scan is simpler for small bounds and can test such edge cases without reproducing the production algorithm. Use it as a correctness reference, not a performance competitor for enormous domains. A small exhaustive reference and a large exact-substitution reference answer different validation questions.

14 12 Explain limits instead of hiding them

A general quadratic solver can combine reductions, factorisation, Pell recurrences and finite scans. The chosen path depends on structure. A failure to finish one structure does not imply the program should silently switch to an arbitrary bound and claim completeness. Report the actual stopping condition and retain exact accepted pairs even when coverage remains incomplete.

For research comparisons classify test problems by definite, indefinite, bilinear and degenerate structure. Record the amount of exact recovery and factorisation work when available. A single average runtime over unrelated conics hides the mechanisms that make some enormous coordinates easy and some small coefficients difficult.

14 13 A bounded integer parabola and an affine product

14 14 Two routes can describe the same finite set

The parabola y=x^2 on -3<=x<=3 has seven integer points. It is a function of x, yet both x and y are solver unknowns. The finite x interval supplies enumeration scope; the equation recovers y exactly. The largest y is9 even without an explicit y bound.

```
int x,y
y=x^2
bound x=[-3,3]
```

The reference set is(-3,9),(-2,4),(-1,1),(0,0),(1,1),(2,4),(3,9). Removing the bound changes the question to an infinite family. If a numerical route returns only a limited scan window, its finite rows are examples rather than a complete parameterisation.

Now solve x*y+2*x+3*y=6 over positive integers. Factoring gives(x+3)*(y+2)=12. Since x+3>=4 and y+2>=3, the only positive factor choice is4*3, giving(1,1). The divisor pair6*2 gives y=0 and is rejected. The pair3*4 gives x=0 and is likewise rejected. This supplies an exact positive reference without scanning a huge rectangle.

```
pos int x,y
x*y+2*x+3*y=6
```

Expand or regroup the equation and compare the returned pair and selected reason. Equivalent polynomial forms should represent the same mathematics, while recogniser resources and unsupported extra conditions can affect which route is selected. Document fallback explicitly instead of assuming every rearrangement is handled by one shortcut.

14 15 A domain change that changes the count

Replace pos int by nonneg int in the product example. The factors must now satisfy x+3>=3,y+2>=2. The admissible choices are3*4,4*3 and6*2, giving(0,2),(1,1),(3,0). The additional points are exactly the boundary cases excluded by strict positivity. This is a particularly clear demonstration of why positive and nonnegative are not interchangeable declarations.

A condition such as x<=y can further filter those pairs, but adding separate conditions can alter specialised eligibility. The mathematical reference still follows from the original factors; the software should explain its selected path. Use this as a domain-contract test rather than reporting a changed runtime as an unexplained regression.

14 16 Verification task

For each of the three models above, keep a complete hand-derived reference list, verify every returned pair and test a row cap smaller than the list where possible. Record whether integer_search_complete or the applicable positive-domain flag reports completion. A zero residual confirms the identity of a point; the reference enumeration explains why the complete finite set has its stated size.

14 17 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

14 18 Inputs and integer domains

Use int x,y to include negative, zero and positive integers, or pos int x,y for strictly positive integers. Nonnegative/negative integer declarations work as before. Variable names are arbitrary. Expanded and factored polynomial expressions are recognised, including equivalent equations with terms on either side. Coefficients are exact integer literals (also integer-valued scientific notation). Exact constant division is accepted when every coefficient divides without a remainder. Computed constants without retained exact literal data, noninteger coefficients, additional equations and separate conditions are not handled by this specialised route; they retain existing solver behaviour.

Example:

```
int x,y
x^2+x*y+y^2=7
```

All 12 signed integer solutions are returned when the solution limit allows it.

```
int x,y
x^2+2*x*y-30*y^2=1
bound y=[-1000000,1000000]
```

This is (x+y)^2-31*y^2=1; the exact Pell reduction finds all ten integer solutions in the stated y interval, including (1,0) and (-1,0).

```
int x,y
y=x^2
bound x=[-100,100]
```

One finite coordinate interval suffices to enumerate every integer solution, even if the other variable has no explicit bound.

14 19 Methods and correctness

* Exact coefficient normalisation and necessary modular tests may prove absence.

* Bilinear forms reduce to (B*x+E)*(B*y+D)=D*E-B*F. Nonzero products use certified factorisation and signed divisor enumeration. Zero products produce lines, enumerated within available bounds or explicitly reported as a limited search.

* For A nonzero and h=4*A*C-B^2 nonzero, set U=2*A*x+B*y+D, k=2*A*E-B*D, V=h*y+k and K=h*D^2+k^2-4*A*h*F. The original equation implies h*U^2+V^2=K. Recovery requires exact congruences and verifies the original integer equation; both signs and zero-coordinate branches are included.

* Positive-definite forms (h>0) use the existing exact Cornacchia/HMW weighted sum-of-squares engine after sign normalisation. This can enumerate all signed integer points without user bounds. Difficult factorisations remain incomplete.

* Square-discriminant indefinite forms with K nonzero reduce to (V-s*U)*(V+s*U)=K, where s^2=B^2-4*A*C. Signed divisors give a finite, complete candidate set. Variables are swapped first when needed.

* Nonsquare indefinite forms reduce to the existing generalised Pell solver when transformed D and N fit its existing signed-64-bit coefficient limits. Transformed sign branches and congruences are checked exactly. A finite original y interval (or swapped x interval) provides a safe transformed magnitude upper bound. Complete underlying Pell enumeration establishes completeness of the recovered points; an interrupted or limited Pell search does not. Unbounded families return a limited verified set, not a symbolic parametrisation or a promise of the smallest solutions in original x order.

* A genuine finite interval of at most 100001 integer coordinates selects a direct exact slice search first. At each coordinate the remaining linear or quadratic equation is solved using exact division and integer square roots. Larger intervals may use algebraic reductions; otherwise at most 2000001 slices are attempted. Parabolic and other degenerate forms use this route.

Every returned coordinate pair is verified with arbitrary-precision integer arithmetic against the exact polynomial. Tolerance and merge settings cannot alter that acceptance test. Pairs are deduplicated and returned in declared variable order, sorted lexicographically among the pairs actually discovered. Discovery timestamps need not increase in this sorted order.

14 20 Completeness and limits

New JSON field integer_search_complete describes integer completeness within the declared domains and explicit bounds. It also mirrors completion of the existing positive Pell/sum-of-squares methods. exact_integer_solutions means returned values are verified exactly; it does not itself mean the search is complete. Precision is exact_integer; the general route's arithmetic label is exact_integer_quadratic. Older positive-only routes retain their arithmetic labels and positive_integer_search_complete field. The general signed route uses integer_search_complete instead.

search_complete retains its existing meaning of interval-certified *real* root coverage. An exact integer enumeration does not establish real-root coverage of a curve. The textual stopping reason explains integer completion. Existing JSON status and C ABI version 1 are unchanged.

This is a staged general-quadratic solver, not an unrestricted symbolic classification of every conic. In particular, no complete symbolic family output is provided for all unbounded parabolic/degenerate cases. Without a finite scan interval or a completed algebraic reduction, a limited scan is explicitly incomplete. For a missing lower bound it starts at -10000, or 20000 below an existing upper bound; a missing upper bound is set to 20000 above the lower endpoint. These sample windows do not become mathematical bounds.

Time, cancellation, factorisation/certification, inherited Pell resource limits, a 100000 stored-solution limit, a 1000000-divisor limit and the slice cap may interrupt enumeration. Underlying sum-of-squares limits remain in force. A solution limit marks results incomplete if enumeration was truncated. A partial search returning no points is never a proof of absence. Supply useful finite bounds and increase the requested solution count when complete lists matter.

14 21 Boost adoption

The general quadratic route uses header-only Boost.Multiprecision cpp_int. Pell's previous custom decimal-limb implementation has also been replaced by a small compatible wrapper around cpp_int. Decimal output and nonnegative arithmetic semantics are preserved. The 10000-digit guard uses a fast bit-length check followed, only at the threshold, by an exact cached comparison with 10^10000; it avoids repeated decimal conversions.

These measure the arithmetic workload, not a promised whole-solver or Windows speedup. The benchmark source and raw CSV are included. Fixed-width arithmetic in other fast paths remains unchanged; there is no blanket Boost replacement of floating-point numerical algorithms. No new runtime DLL is introduced.

Chapter 15 Weighted Sums of Squares

15 1 Start with a finite positive representation problem

The equation A*x^2+B*y^2=C with positive A,B,C defines a bounded ellipse. Positive integer declarations require x>=1 and y>=1. Necessary upper bounds are x<=floor(sqrt(C/A)) and y<=floor(sqrt(C/B)); including positivity can sharpen them by reserving at least B or A for the other term. This finiteness makes a complete representation list a meaningful target. Here floor denotes a mathematical bound. Evaluate it before entering the model; floor itself is a Compute-only function, not an ordinary-equation expression call.

For 2*x^2+3*y^2=35, positivity gives y<=3. If y=1 then x^2=16 and x=4. If y=2, x^2=23/2 is not an integer square. If y=3, x^2=4 and x=2. Thus the positive ordered pairs are (4,1) and (2,3). The fact that the coefficients differ means exchanging coordinates is not generally a symmetry.

15 2 Separate primitive and nonprimitive representations

A representation is primitive when gcd(x,y)=1. The equation x^2+y^2=20 has positive pairs (2,4) and (4,2), both with gcd two. An algorithm that searches only coprime coordinates would incorrectly report no positive solutions. Dividing by the common square factor gives u^2+v^2=5 and recovering x=2u,y=2v restores the original pairs.

This explains the square-divisor stage in a complete weighted-square method. If g divides both coordinates, then g^2 divides C. Enumerating admissible square divisors separates primitive subproblems without discarding the nonprimitive solutions. When coefficients share factors with C, additional exact divisibility substitutions may be necessary before applying the primitive algorithm.

A common factor in coefficients and target can sometimes be cancelled immediately. The equation 6*x^2+9*y^2=105 reduces to 2*x^2+3*y^2=35. But replacing a large C by a rounded floating-point number before this cancellation can destroy the identity. Retained exact literals and integer arithmetic are essential parts of the method, not just an output-format choice.

15 3 Use modular obstructions before factorisation

Squares modulo three are 0 or 1. Therefore x^2+y^2=3 forces both squares to be 0 modulo three only if their sum were0, but 3 is0 modulo3; that necessary condition alone allows coordinates divisible by3 and does not finish the proof. Modulo four gives the decisive impossibility because two squares cannot sum to3. The example illustrates the value of choosing a modulus appropriate to the equation rather than assuming every small modulus proves what is needed.

For x^2+61*y^2=C, reduction modulo61 gives x^2=C mod61. If C is not a quadratic residue modulo61, no integer representation exists. If it is a residue, there may still be no representation with the requested signs and bounds. Passing one congruence is a prerequisite, not a constructive solution.

The implementation uses factorisation and modular square-root techniques to avoid scanning every integer up to sqrt(C). That is why a target with many decimal digits can be easy when its factorisation is tractable. A shorter target with difficult factor structure can be harder. Runtime should not be predicted from decimal digit count alone.

15 4 Verify a large representation without a floating square root

The supplied example x^2+61*y^2=35677541036108645912385710 has positive pair x=1234567,y=764772987169. An exact integer substitution verifies the identity regardless of the output's visual length. A double-precision evaluation is not a suitable exact equality test for the large right-hand side.

To check every returned pair, parse decimal strings as integers, compute both squared contributions and compare their exact sum with C. Also check positivity and each bound. To prove completeness independently for a small example, enumerate y and test whether (C-B*y^2)/A is a nonnegative perfect square after exact divisibility. For very large targets this reference scan may be impractical, so use structurally independent number-theory checks or a carefully chosen test with known factors.

The case x^2+y^2=5e40 illustrates coordinate size without changing the mathematical type: (10^20,2*10^20) and its exchange are examples. An exported coordinate can exceed 64-bit integer capacity. Preserve the strings and choose an arbitrary-size integer type in any verifier. The Desktop's arithmetic selection does not turn the exact route into an approximate double computation.

15 5 Signs zero coordinates and variable labels

The positive specialised route excludes zero and negative coordinates. General signed quadratics provide a broader path when int is used. For x^2+y^2=25, the nonnegative points include axis cases and positive (3,4),(4,3). Signed reflections add the four combinations of signs for each nonzero ordered pair, while an axis zero does not have two distinct signs.

When A=B, exchanging x,y preserves the identity, but the returned vectors remain ordered if the variables are labelled. If an application counts unlabelled pairs, state an independent convention such as x<=y and determine whether adding that condition changes recogniser eligibility. Mathematical filtering and specialised route selection are separate concerns; use the returned reason to understand which computation actually occurred.

A lower bound can exclude only part of a symmetric family. For x^2+y^2=25 with positive coordinates and x>=4, only (4,3) remains. Increasing the row cap does not restore (3,4), because the explicit domain excludes it. Conversely a cap of one on the unrestricted positive set can truncate two valid rows without proving the excluded row is absent.

15 6 Precision residuals and stopping limits

Exact verification of each returned representation does not mean factorisation always completes. Time, retry, divisor-combination, stored-row and digit guards can interrupt the search. An incomplete result with no points must be described as no points returned under the limits, not no representations exist.

The JSON precision exact_integer and exact_integer_solutions flag identify the nature of accepted coordinates. positive_integer_search_complete or the applicable general integer completion flag identifies domain coverage. The real search_complete field has another meaning and is not the integer completeness test for an ellipse. A complete finite lattice list does not enumerate the infinite continuum of real ellipse points.

15 7 Practice problems with reasoning

1. Find positive solutions of x^2+y^2=65. Testing squares up to64 gives (1,8),(4,7),(7,4),(8,1). No zero coordinate works because65 is not a square.

2. Which pairs from Problem 1 satisfy x<=y? They are (1,8),(4,7). This is an unordered-counting convention, not the full ordered set.

3. Solve 2*x^2+3*y^2=35 with x>=3. Only (4,1) remains.

4. Explain the common-square factor in x^2+y^2=20. Both positive representations have gcd2, corresponding to the primitive representations of5 after division by4.

5. Decide whether x^2+y^2=7 is possible in integers. Modulo4 the right side is3 and the left side cannot be3, so it is impossible.

6. Explain why replacing 5e40 by a double and then converting back to an integer is unsafe. The decimal integer may not be represented exactly; later exact arithmetic would verify the rounded target rather than the original literal.

7. What does a factorisation timeout prove about existence? Nothing by itself; it proves that the chosen computational limits were reached.

8. Why retain the original equation when a reduction is available? Recovery and scale factors must satisfy the original coefficients, signs and bounds, not merely a transformed necessary relation.

15 8 A representation catalogue exercise

Choose ten small positive A,B,C triples, including common coefficient factors, impossible modular cases and nonprimitive pairs. Derive complete lists by independent exact enumeration. Compare them with Desktop results using a row cap larger than the reference count. Then include one larger tractable target and record exact substitution without attempting an unrealistic exhaustive reference scan.

Organise the report by mathematical structure and factorisation needs. Identify which evidence establishes a pair and which establishes the whole list. This approach carries into later numerical chapters: every method needs checks tailored to the kind of object it returns.

15 9 Design complete representation tests

15 10 Common factors in coefficients and target

The equation6*x^2+9*y^2=105 has the same positive solution set as2*x^2+3*y^2=35 after division by three. Now consider2*x^2+2*y^2=10, which reduces to x^2+y^2=5. These examples should finish with the same coordinate pairs as their primitive coefficient forms, not rescaled coordinates. Coefficient cancellation and coordinate substitution are different operations.

By contrast x^2+y^2=20 is reduced through a coordinate common factor: x=2*u,y=2*v gives u^2+v^2=5. Here the recovered coordinates must be multiplied by two. A verifier that confuses these reductions can return points on the reduced ellipse instead of the original one. Exact substitution into the original identity catches the mistake immediately.

15 11 A small table with independently complete counts

Equation	Domain	Reference pairs
x^2+y^2=5	Positive ordered	(1,2), (2,1)
x^2+y^2=20	Positive ordered	(2,4), (4,2)
x^2+y^2=25	Nonnegative ordered	(0,5), (3,4), (4,3), (5,0)
2*x^2+3*y^2=35	Positive ordered	(2,3), (4,1)
x^2+y^2=3	Signed integers	No pairs, by modulo four

Derive every row before using it as a software benchmark. For the first row, x can only be1 or2. For the second, x ranges1 through 4 and only4 and16 leave square complements. For25 the endpoint zero cases must be tested explicitly. For the weighted row, enumerate y=1,2,3 as shown earlier. These finite arguments are completeness evidence; observed numerical output alone is not their source.

Set Maximum solutions above the expected count, then deliberately reduce it below the count. Compare the stopping reasons and completion flags. A well-designed test verifies that the program recognises truncation, not only that a large cap returns the correct rows. Repeat with a lower bound that removes one reference pair and verify that this domain filtering can still be complete.

15 12 Large targets and factorisation evidence

The production method accepts certified factors rather than treating every probable prime as proven for complete enumeration. This matters when a no-representation conclusion depends on exhausting prime-power root combinations. An incomplete factorisation must remain an incomplete computation even if all explored branches fail.

For a teaching comparison, choose a target with a deliberately known factorisation. Construct it from a known pair and coefficients, then keep the pair as a guaranteed existence witness. That witness does not identify every representation, but it prevents a timeout from being misreported as an empty complete set. Add a separate small case with a complete reference list to test enumeration.

A large exact target can still fall outside the front-end coefficient/literal scope. The bound extension is not a promise to parse arbitrary symbolic coefficient expressions. When writing a model generator, preserve supported exact literals and diagnose unsupported input before comparing algorithmic runtime.

15 13 Recover a known large representation from its structure

15 14 Construct a target before asking for its representations

Choose x0=12345,y0=678 and coefficients A=2,B=3. Define C=2*x0^2+3*y0^2=306177102. The pair is guaranteed to exist by construction, but the equation can have additional representations. A solver should return only pairs it verifies, while a complete-list claim requires exhausting the relevant number-theory search.

```
pos int x,y
2*x^2+3*y^2=306177102
```

A useful independent test first confirms the planted pair exactly, then scans the finite y interval for this moderate target using integer divisibility and perfect-square tests. This separates an existence witness from an exhaustive reference. It is not valid to claim that the planted pair is unique merely because it generated C.

The same idea scales to x0=10^20,y0=2*10^20,A=B=1, giving C=5e40. The coordinate construction remains exact, but a naive scan to2*10^20 is impractical. For such tests use independent arithmetic certificates and known factor structure rather than pretending a brute-force comparison was executed.

15 15 Bounds can isolate a known representation

For x^2+y^2=65, restricting x to[4,4] selects(4,7) over positive integers. A singleton bound is a mathematical condition, not an initial estimate. Changing the interval to[4.1,6.9] admits x=5 or6, neither of which leaves a square complement, so the positive set is empty. Exact inward rounding is directly testable on this small reference.

Run the same cases using model bounds and dialog overrides. The completed report should show which bounds were used. Save a project before comparing with another machine; saving only equation text loses dialog overrides and can make apparently identical models ask different questions.

15 16 A representation is not automatically a physical design

Weighted squares can model an idealised energy, distance or norm constraint. Integer coordinates may represent manufacturing counts or permitted discrete dimensions. Exact satisfaction then verifies the arithmetic design

model, but physical tolerances, losses and feasibility can impose additional requirements. State those separately rather than calling an exact integer identity a certified engineering outcome.

For number-theory teaching, distinguish ordered pairs, sign variants, primitive representations and bounds. For engineering teaching, distinguish those arithmetic conventions from labels and units. The same two-row table can be interpreted incorrectly if the reader does not know which meaning applies.

15 17 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

15 18 User example

```
pos int x,y
x^2 + 61*y^2 = 35677541036108645912385710
bound x = [1,1e30]
```
The complete positive solution set within these bounds is:

```
x = 1234567
y = 764772987169
```
The engine verifies the original equation with exact integer arithmetic. The right-hand side is preserved from its literal digits, never rounded through double. The final native test took about 0.002 seconds on this environment; this is not a Windows performance measurement or a runtime guarantee.

15 19 Supported scope

The exact path recognises one equation a*x^2+b*y^2=N with a,b positive and x,y restricted to strictly positive integers, by declarations or positive bounds. Rearrangements, common scaling factors, expanded integer arithmetic, either variable declaration order, implicit multiplication and integer-valued decimal or scientific numeric literals are supported. Bounds are respected.

Examples:

```
pos int x,y
2*x^2 + 3*y^2 = 35
```
This returns (x,y)=(2,3) and (4,1).

```
pos int x,y
x^2 + y^2 = 20
```
This returns (2,4) and (4,2), including the common factor of 2. The algorithm is not restricted to coprime x,y.

```
pos int x,y
x^2 + y^2 = 5e40
```
There are 42 positive ordered pairs, including (10^20,2*10^20). Coordinates larger than fixed-width integers are returned as exact decimal strings.

Additional equations/inequalities, noninteger coefficients, unrestricted negative/zero variable branches, mixed-sign coefficients and general quadratic forms with nonzero cross/linear terms are not added to this exact path. Those models retain the existing algorithms. Existing generalised Pell support and polynomial-root corrections are preserved.

15 20 Method

This is a Cornacchia/Hardy-Muskat-Williams baseline with exact preprocessing, not the newer subset-sum research algorithm.

1. Parse the quadratic identity using exact integers. Apply positive-domain, exact-minimum, coefficient-gcd and modular impossibility checks.

2. Factor the normalised target. Factorisation uses trial division and Pollard rho. Prime factors within the unsigned-64 range use deterministic Miller-Rabin bases. Larger probable primes require a fully factored n-1 Lucas primality proof before acceptance. Probable primality alone is not enough for complete enumeration or a no-solution result.

3. Primes shared by a coefficient and the target force divisibility of the other variable. Substitute and divide exact common factors until a,b,N are pairwise coprime. Save the coordinate scale factors; this retains all original solutions, rather than discarding non-coprime coefficient cases.

4. Enumerate every square divisor g^2 of the transformed target. For each, solve the primitive equation at target N/g^2. This recovers nonprimitive solutions after rescaling.

5. Compute the required prime-power modular square roots using Tonelli-Shanks, exact Hensel lifting and the 2-adic cases. Combine all roots by CRT.

6. Apply the Euclidean descent to the congruence a*t^2=-b (mod N/g^2), then check the remaining quotient for an exact square. Handle coordinate interchange explicitly for equal normalised coefficients.

7. Rescale, verify the full original integer identity, apply bounds and remove duplicates. Results are ordered lexicographically in declared-variable order. If search ends early, only the known exact results are returned; an interrupted prefix need not contain the globally smallest solutions.

The method can be fast for a large decimal target with tractable factors. Difficult factorisations and many prime-factor/root combinations remain real limitations; decimal digit count alone does not predict difficulty.

References:

- Hardy, Muskat and Williams (1990), A deterministic algorithm for solving n=fu^2+gv^2 in coprime integers u and v, Mathematics of Computation 55, 327-343. https://people.math.carleton.ca/~williams/papers/pdf/165.pdf

- Cornacchia's Euclidean/modular-root method and its refinements.

- Boost.Multiprecision 1.83 headers, with upstream copyright/license notices under licenses/BOOST_COPYRIGHT.txt and licenses/BOOST_LICENSE_1_0.txt.

15 21 Limits and reporting

Time and cancellation are checked during factorisation, primality verification, modular arithmetic, root combinations and square-divisor enumeration. Guards:

- 10,000,000 counted arithmetic/search operations.

- 65,536 square-divisor subproblems and 65,536 modular roots per subproblem.

- 100,000 stored solution pairs.

- Factorisation recursion and Pollard-rho retry limits.

- 10,000 characters/digits in the exact-literal parser.

Hard cases report enumeration incomplete when a guard is reached. Failure to factor a target is never converted into a proof of no solutions. The solution limit controls the number of returned rows; the engine may enumerate more candidates first to order them and determine completeness.

JSON/DLL fields:

```
precision: exact_integer
search_arithmetic: exact_integer_cornacchia_hmw
```

```
exact_integer_solutions: true
positive_integer_search_complete: true or false
```

Residual 0 denotes an exactly verified identity. A true positive-domain flag means the complete positive integer set within the effective bounds has been returned, or the set has been proved empty. It is false after solution, time or resource truncation. The existing search_complete field retains its separate interval-certification meaning; an ellipse has infinitely many real points.

Numeric decimal/scientific bound literals are rounded inward exactly, including notation such as 1e30 and nonintegral endpoints. Nonliteral bound expressions retain existing parser arithmetic and inward integer rounding semantics.

The existing desktop launcher displays/exports solution strings without losing digits and shows exact_integer in its summary. Its legacy report footer still uses generic numerical-search wording; for exact runs use the stopping reason and positive-domain flag to assess completeness. No GUI source change is required to operate the new engine.

Chapter 16 Initial Value Ordinary Differential Equations

16 1 Derivatives describe change and initial data select a branch

An initial-value problem specifies a derivative law and enough values at one independent coordinate to determine the intended trajectory. For y'=-2*y every C*exp(-2*t) satisfies the differential equation. The condition y(0)=3 selects y=3*exp(-2*t), so y(1)=3*exp(-2), approximately 0.4060058497. Without the initial value the problem is a family, not a single numerical curve.

```
ode
y'=-2*y
initial t=0
initial y=3
interval t=[0,1]
method=RK45
output points=11
```

Verify both the derivative law and the initial value. Then compare every reported sample with the analytic reference. Checking only the final endpoint can miss an error in the interior or a wrong output-coordinate interpretation. The requested output is a function sampled across an interval, not eleven unrelated algebraic roots.

16 2 Derive a first-order system from a second-order model

For a mass-spring oscillator let q be displacement and v=q'. The equation q''+q=0 becomes q'=v and v'=-q. With q(0)=1,v(0)=0 the exact solution is q=cos(t),v=-sin(t). The direct second-order input and the explicit first-order system represent the same initial-value problem.

```
ode
q'=v
v'=-q
initial t=0
initial q=1
initial v=0
interval t=[0,6.283185307179586]
output points=101
```

The mechanical energy E=(q^2+v^2)/2 is constant in this nondimensional model. Numerical energy drift offers an independent diagnostic, though this general integrator does not promise exact symplectic conservation. Compare the returned trajectory with both cos/sin references and E=1/2. Agreement at a handful of visually smooth points is weaker evidence than a quantified maximum error and invariant check.

In a physical oscillator m*q''+c*q'+k*q=F(t), units matter. Position and velocity have different dimensions and may require different absolute tolerance scales. Converting to first order clarifies the state vector and the initial-data count; it does not remove those scales. The supported direct-order conversion performs that bookkeeping automatically for suitable equations.

16 3 Work through fixed-step RK4 on a transparent example

Classical RK4 uses four slope evaluations to form one state update. For y'=y at y(0)=1 with h=0.1, k1=1, k2=1.05, k3=1.0525 and k4=1.10525. The update is y1=1+h*(k1+2*k2+2*k3+k4)/6=1.105170833333333. The exact exp(0.1) is approximately 1.105170185988091, so the one-step error is about 6.47e-7.

This arithmetic makes the distinction between local slope samples and reported output visible. The four internal slopes are not four returned solution rows. Halving h reduces the fixed-step global error by roughly a factor of16 for a smooth problem in the fourth-order regime, until roundoff or another limitation dominates. It is a convergence observation, not a guaranteed bound for every model.

```
ode
y'=y
initial t=0
initial y=1
interval t=[0,1]
method=RK4
step=0.1
```

Run h=0.1,0.05,0.025 and compute the endpoint error against exp(1). Keep the same model and precision. If a reporting directive is used, distinguish decimation from a change in integration. In this mode rtol and atol do not create adaptive control; the fixed step itself must be assessed.

16 4 Understand adaptive error scales

An adaptive integrator compares an estimated local error with a scale built from absolute and relative tolerances. Absolute tolerance matters near zero; relative tolerance scales with the state magnitude. The numerical estimate does not represent measurement uncertainty and is not a certified global error accumulated over the whole interval.

For a component with characteristic value 1000000, rtol=1e-7 gives a different absolute scale than for a component near 1e-8. A single absolute tolerance can be unsuitable for a mixed-unit system. Component directives let you specify a smaller or larger floor consistent with the variable's scale. Choose them deliberately rather than making every number as small as possible.

A smooth solution may require small internal steps in a rapid transient and larger ones later. Requested output points are evaluated through continuous interpolation rather than forcing every accepted step to end at a sample. Thus changing from 101 to1001 output points should mainly change output density, not transform the intended accuracy target. Internal step limits and resource caps can still affect completion.

16 5 Use independent comparisons rather than a picture alone

A useful study reports maximum sample error against a known reference, number of accepted/rejected steps, derivative evaluations and completion status. When no analytic reference exists, compare tighter tolerances, smaller fixed steps, conservation laws or a physically independent relation. A discrepancy should lead to investigation of domains and scale before simply increasing the budget.

Backward integration is mathematically permitted for supported problems but can be unstable in the sense of sensitivity to perturbations. Exponential decay forward becomes growth backward. Numerical stability of a chosen integrator does not remove that conditioning. Use meaningful backward initial data and assess amplification over the entire requested interval.

16 6 Practice and worked answers

1. Solve y'=3*y,y(0)=2 analytically. The solution is2*exp(3*t), and its derivative and initial value verify the expression.

2. Convert y''=-4*y to first order. Set v=y'; then y'=v,v'=-4*y. Two initial values are needed.

3. For y'=y,h=0.1, explain why k2 differs from k1. It samples a predicted midpoint state, not the original state again.

4. What should a fourth-order step-halving ratio approach? About16 for global errors in a smooth asymptotic regime, provided the same quantity and interval are compared.

5. If a state crosses zero, can relative tolerance alone provide a nonzero scale? No; a positive absolute tolerance supplies the floor.

6. Does an output step of0.01 force RK45 internal steps of0.01? No; use max step to constrain internal step magnitude.

7. A run returns one row at the initial point and a failure status. Is the IVP solved? No. The row is valid initial data, not a completed trajectory.

8. Why compare derivative states as well as position? A position-only match can hide phase or velocity error, and physical invariants can depend on both.

16 7 Suggested independent laboratory

Use exponential growth for exact endpoint checks, an oscillator for phase and energy checks and a forced equation for a nonhomogeneous reference. For each model retain the project, trajectory JSON and report. Compare methods only after matching initial data, interval and accuracy targets. Runtime alone is not a fair comparison when one method returns a much less accurate curve.

16 8 A complete forced motion model

16 9 Use an exact polynomial trajectory

A particle with acceleration a=2 and initial position y(0)=1,velocity y'(0)=3 has y=1+3*t+t^2 and y'=3+2*t. The second-order IVP is smooth and has a simple reference at every sample. At t=2 the position is11 and velocity7.

```
ode
y''=2
initial t=0
initial y=1
initial y'=3
interval t=[0,2]
method=RK45
output points=21
```

Check both columns at the midpoint and endpoint, not just position. A direct second-order listing tests the automatic derivative-chain conversion, while the equivalent first-order system tests explicitly named state handling. Their physical trajectories should agree within numerical criteria even if variable labels differ.

For fixed RK4 this polynomial problem has very small truncation effects and can be nearly exact apart from floating-point operations. That makes it useful for checking state conversion and output indexing, but a poor sole demonstration of fourth-order convergence. Use the exponential example for a nonzero asymptotic error study. A test suite should contain both implementation checks and genuine accuracy challenges.

16 10 A coupled conservation check

The two-state system a'=-k*a,b'=k*a with a(0)=1,b(0)=0 conserves a+b=1. Its exact functions are a=exp(-k*t),b=1-exp(-k*t). This verifies that the solver integrates a coupled state vector and gives a simple invariant independent of the individual exponential values.

Both quantities may be interpreted as nondimensional amounts in a closed transfer model. If adding an external source or another loss, the conservation relation changes. Do not reuse a previous invariant after changing the equations. Similarly, negative numerical samples need investigation in an amount model; the general integrator does not automatically enforce positivity merely because the intended physical quantity is nonnegative.

16 11 Initial data and interval direction

The initial independent coordinate must be the first endpoint. For a backward run from 2 to0, supply the state at2, not the original state at0. In the polynomial example those data are y=11,y'=7. Integrating back should recover1 and3. This is a useful consistency check, though round-trip agreement alone does not certify accuracy for unstable problems.

16 12 Report more than one error measure

For an analytic trajectory, calculate maximum absolute error over the reported samples and, when meaningful, a relative error away from zero. For an invariant, report its maximum deviation separately. State the sample grid used. A single endpoint error can be misleading when oscillatory errors cancel or when symmetry forces a chosen value to zero.

Record the selected method,precision,tolerances,time and status with the errors. If a graceful partial IVP stops early, restrict comparisons to returned coordinates and do not claim coverage of the requested full interval. If the Desktop is force-stopped, there is no new partial result to analyse.

16 13 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

16 14 Basic input and higher-order equations

```
ode
y'' + 0.5*y' + 4*y = 0
initial t = 0
initial y = 1
initial y' = 0
interval t = [0,20]
method = RK45
output step = 0.05
rtol = 1e-8
atol = 1e-11
```

The interval names the independent variable, which may also be x or s. Its initial value must equal the first interval endpoint in the selected arithmetic. Backward integration is supported. Endpoints must be finite, distinct and have finite difference. Statements may use newlines or semicolons; normal comments and US/EU notation work. In EU mode, use decimal commas and semicolon separators inside lists/function calls. The CLI accepts the same model with --file or --eq.

Primes must immediately follow a state name. First-order systems integrate all states together. Direct orders 1..8 are converted to first-order chains, with a maximum of 64 total expanded states. For order m, supply initial values through derivative m-1. Columns use readable names such as y, y', y''. Named constants may depend on other constants; one constant definition per statement is supported:

```
constant rate = 0.7
population' = rate*population*(1-population/1000)
```

Equations must be linear in their own highest derivative: y''=-sin(y) and (1+t)*y''+y'=0 are supported, while (y'')^2, sin(y'') and 1/y'' are rejected. A variable highest-derivative coefficient must remain defined and nonzero. Couplings through lower derivatives are allowed, e.g. x''=-x+y', y''=-y. Implicitly coupled highest derivatives must first be isolated into separate explicit equations. This is not a DAE or implicit-branch solver. Names beginning __ode_ are reserved for internal conversion. pi is reserved as a built-in.

Expressions use the existing real function evaluator and its domain checks. Integer modulo, algebraic integer declarations and separate algebraic bounds are not ODE features. Initial values for undeclared states, missing derivatives, unknown variables and duplicate/cyclic definitions are rejected.

16 15 Integration methods

Fixed-step RK4

```
method = RK4
```

```
step = 0.2
```
This uses classical fourth-order Runge-Kutta with the specified positive step magnitude, and shortens the final step if necessary to reach the endpoint. A step directive alone selects RK4 when method is absent. It has NO adaptive error control or local-error estimate. rtol/atol in a reused model do not control RK4 accuracy. Use step-halving comparisons and physical invariants to assess it.

Default RK4 output saves each step. An output step may decimate the trajectory only when it is an integer multiple of step; this never changes integration. Alternatively, sample times can request arbitrary intermediate values using the cubic Hermite continuous extension. output points and adaptive initial/min/max step controls are rejected in fixed RK4 mode. max steps and time budgets apply.

A true endpoint derivative is evaluated for continuous interpolation and reused at the next step. A successful N-step RK4 run therefore uses 4*N+1 RHS vector evaluations in this supported mode. The classical RK4 state update itself is unchanged. A nonfinite stage or endpoint stops the run without a smaller-step retry.

Adaptive RK45 (default)

```
method = RK45
rtol = 1e-8
atol = 1e-11
```
This uses the Dormand-Prince 5(4) pair with fifth-order propagation and an embedded local error estimate. A quartic continuous extension evaluates requested output points between accepted internal steps. Reporting times no longer force internal steps to end at every row. With the same model/controls, changing output density leaves the internal step sequence unchanged, except when output storage limits or terminal events stop a run. Values are approximate interpolants, not exact solutions or rigorously certified enclosures.

16 16 Output grids and continuous queries

Choose ONE reporting specification:

```
output points = 201
```
or

```
output step = 0.05
```
or

```
sample times = [0.013,0.333,0.777]
```
Endpoints are included automatically. Sample times must be strictly ordered in the integration direction and lie inside the interval. At most 9999 requested rows are supported, additionally limited by 500000 display cells. A terminal event or an interrupted internal step may append an extra terminal/partial row. RK4 retains its integer-multiple output-step rule, but explicit sample times may be unaligned with its fixed grid. Adaptive output step is a reporting interval; use max step, rather than output step, to restrict internal step magnitudes.

For machine-readable reusable continuous output, add:

```
save dense = true
```
Dense interpolation is always used internally; saving is optional and defaults to false. JSON dense_segments contains t0, t1, start, end, and coefficient rows c[0], c[1], ... for each accepted segment. For time q within that segment set u=(q-t0)/(t1-t0), and evaluate each state as

```
start + u*(c[0] + u*(c[1] + u*(c[2] + ...)))
```
Use start/end fields directly at exact endpoints to preserve the accepted values. RK45 segments are quartic; RK4 segments are cubic Hermite; SDIRK2 saves separate cubic Hermite segments for its two half steps. Saved segments never extend beyond a terminal event: the final segment is truncated and reparameterised to that time. Never extrapolate these polynomials outside their stated interval.

Saved output is capped at 10000 segments and 500000 coefficient cells. Reaching that cap stops with resource_limit, rather than silently discarding the continuous record. For long stiff runs leave save dense off and request only needed samples.

16 17 Controls, status and precision

Control	Default	Meaning
rtol	1e-7	Relative local-error scale for adaptive methods
atol	1e-10	Absolute scale, override with e.g. atol y'=1e-12
max steps	100000	Attempted macrosteps, maximum 10000000
initial step	automatic	Initial adaptive step magnitude
max step	interval length	Maximum adaptive internal step, 0 means automatic
min step	floating resolution	Adaptive minimum magnitude, 0 adds no user minimum
newton iterations	12	SDIRK2 stage iteration cap, allowed 2..50
event tolerance	precision-dependent	Absolute event time-bracket tolerance
save dense	false	Include continuous polynomial records in JSON

atol must be positive. Adaptive rtol must be between 100 machine epsilons and 1. Double is the default; --extend performs the entire integration and interpolation in binary128 under the existing licence rules. Default ODE tolerances are unchanged in either precision. Set tighter controls explicitly. --quad-search is redundant for ODE mode. The plot itself uses double precision.

Solver Settings controls the time budget and precision. Algebraic residual tolerance, maximum solutions, restarts and root-merging are not ODE controls. Algebraic bounds and IBEX/certification options are rejected in ODE mode. --check validates syntax and settings without evaluating initial derivatives or integrating. The integration budget excludes parsing/licensing and final formatting. It may slightly overrun inside an individual operation. Aborted attempted trials count among rejected steps, preserving attempted=accepted+rejected.

Status 0 means completed integration, intentional terminal stopping, or valid syntax checking; 2 means a limited/failed integration with finite partial data; 1 means invalid input/settings. Existing licence status remains unchanged. Partial numerical statuses include step_limit, time_limit, cancelled, step_size_failure, nonfinite_derivative, newton_failure, dense_output_failure, event_failure and resource_limit. A final accepted state between requested output times is appended on interruption. The Desktop force-Stop button terminates the worker and cannot retrieve unfinished data; it retains the previous labelled snapshot. Use a time/step budget for graceful partial results.

16 18 Applying the method in the Desktop

16 19 Enter a direct higher order IVP

```
ode
y''+0.5*y'+4*y=0
initial t=0
initial y=1
initial y'=0
interval t=[0,20]
method=RK45
rtol=1e-8
atol=1e-11
output step=0.05
```

Start the model with ode. The interval names the independent variable; it may be t, x or another supported name. The initial independent value must equal the first endpoint. An order-m state requires initial values through derivative order m-1. The Desktop returns columns for the independent variable, state and lower derivatives.

Orders 1 through 8 are supported, up to 64 expanded first-order states. Equations must be affine in their own highest derivative. Coupled highest derivatives must be isolated first. This is not a general differential-algebraic or implicit-branch solver. Backward integration is supported with reversed endpoints and the corresponding first-endpoint initial data.

16 20 Choose a method

Method	Use	Required interpretation
RK45	Default adaptive method for many smooth nonstiff IVPs	rtol and atol control estimated local error
RK4	Fixed-step textbook calculations	Set method=RK4 and step=h; no adaptive error estimate
SDIRK2	Explicitly selected stiff integration	Implicit Newton solves; adaptive second order; no automatic stiffness switching

A step directive without a method selects RK4 on ordinary IVPs. RK4 shortens the final integration step to reach the endpoint. Its rtol and atol values do not provide adaptive accuracy control. Compare h and h/2 results when assessing a fixed-step answer.

RK45 defaults to rtol=1e-7 and atol=1e-10. Use a component absolute tolerance such as atol y'=1e-12 when derivative units/scales differ. SDIRK2 accepts newton iterations, default 12, within its allowed range. Smaller tolerances may increase work substantially and do not certify global error.

16 21 Reporting and internal steps

Choose one reporting specification: output step, output points or sample times. Endpoints are included. Adaptive output step changes where values are reported; it does not force the internal integrator to take that step. Use max step when limiting internal step magnitude matters.

```
sample times=[0.013,0.333,0.777]
```

Sample times must be ordered in the integration direction and lie within the interval. RK4 output step must be an integer multiple of its fixed step, but explicit sample times can use continuous interpolation at unaligned times. At most 9999 requested output rows are allowed, also subject to the 500000 display-cell limit.

Useful controls include initial step, min step, max step and max steps. The default max steps is 100000 attempted macrosteps. Check convergence by tightening tolerances or reducing steps, rather than reading more printed digits as greater accuracy. Binary128 reduces roundoff; it does not remove modelling error or instability.

16 Initial value jobs inside a workflow

```
workflow
ode exponential_decay
    y' = -y
    initial t = 0
    initial y = 1
    interval t = [0,1]
    output points = 3
end
point = at(exponential_decay.result.trajectory, 3)
report at(point.values, 2)
```

Expected y(1) = exp(-1).

To study decay rates, replace the coefficient with @rate and run the ODE job inside a loop. Compare each accepted endpoint with exp(-rate).

Chapter 17 Stiff Equations Events and Continuous Output

17 1 Separate a fast transient from a slow target

Consider y'=-k*(y-cos(t))-sin(t) with initial y(0)=1. The exact solution y=cos(t) is obtained by substitution. A perturbation e=y-cos(t) satisfies e'=-k*e, so numerical or initial perturbations decay on timescale1/k. For k=1000000 this is far faster than the slowly varying cos(t) target.

This is a clear stiffness example: an explicit method can be constrained by the fast stable mode even when the desired trajectory looks smooth. A stiff method aims to handle that stability burden efficiently. It does not make a coarse approximation automatically accurate, and it cannot remove a truly unstable physical mode.

```
ode
y'=-1000000*(y-cos(t))-sin(t)
initial t=0
initial y=1
interval t=[0,5]
method=SDIRK2
rtol=1e-7
atol=1e-10
output step=0.01
```

17 2 Why an implicit stage needs a nonlinear solve

An implicit Runge-Kutta stage contains its own unknown state inside the derivative evaluation. Even when the original differential equation is simple, the stage can require solving an algebraic system. SDIRK2 uses damped Newton iteration and linear solves within each attempted step. A failure to converge can reject the step and cause a retry at a smaller size.

For a scalar linear decay, implicit equations can be solved analytically, which helps explain their stability. For nonlinear coupled states, analytic expression Jacobians or finite-difference fallbacks guide Newton updates. The report's Jacobian, linear-solve and nonlinear-failure counts describe this work. They are not extra output samples or new solution branches.

The implemented method is a two-stage second-order L-stable SDIRK scheme with step-doubling error estimation and continuous interpolation checks. It is neither BDF nor Radau and does not automatically detect stiffness or switch methods. Method names should identify the actual algorithm because accuracy, storage and failure properties depend on it.

17 3 Events ask for a crossing not an arbitrary sample

A falling body released from height10 with acceleration-9.81 has y(t)=10-9.81*t^2/2. The impact time is sqrt(20/9.81), approximately 1.427843. If output is sampled every0.1, the true crossing normally lies between rows. Event location uses the continuous interpolant to bracket and refine that crossing rather than merely stopping at the first negative sample.

```
ode
y''=-9.81
initial t=0
initial y=10
initial y'=0
interval t=[0,5]
event ground=y
event ground terminal=true
event ground direction=falling
output step=0.1
```

A successful terminal event is intentional completion of an event-controlled calculation. It need not reach the requested endpoint five. Check both integration_successful and integration_complete when interpreting JSON. The events table records the event name, full state and whether it was terminal.

17 4 Direction and initial zeros require care

If a projectile begins at ground level and moves upward, event ground=y is already zero initially. A falling-direction filter is intended to distinguish the later downward crossing from departure. Directions are defined with respect to increasing independent-variable time, including backward integrations; changing integration direction does not redefine the physical meaning of rising.

An event that just touches zero without changing sign can be missed. For y=(t-1)^2, the event function y has a tangent zero at t=1. Endpoint sign-change tests alone need not bracket it. Multiple oscillations inside one accepted segment can similarly hide several roots. Smaller max step can improve bracketing, while denser displayed samples alone do not change the internal crossing tests.

An event definition is not a state-reset rule. A bouncing-ball impact needs a new initial state or a separate explicitly managed restart; the Desktop does not automatically apply a restitution law. Known discontinuities should be split into separate problems with mathematically justified data, not treated as smooth crossings merely because a solver accepted a step.

17 5 Continuous output belongs to a segment

Interpolation coefficients describe the approximate solution only between their stored segment endpoints. For u=(q-t0)/(t1-t0), a saved polynomial has the documented nested form start+u*(c0+u*(c1+...)). Exact stored endpoints should be used directly at t0 and t1. Extrapolating beyond the segment can produce plausible but unsupported values.

The degree depends on method: RK45 has quartic continuous output, RK4 cubic Hermite, SDIRK2 separate cubic pieces for its accepted half steps, and regular singular series can have more coefficients. A reader that assumes every segment has four rows will mishandle a valid later method without any ABI change. Read dimensions from the data.

Save dense=true is optional. It is useful for reusable queries or external analysis, but consumes storage and can reach coefficient/segment limits. Ordinary Desktop curves use returned samples and straight visual joins; a more densely sampled curve does not display the saved polynomial exactly. Distinguish numerical interpolation from graphical rendering.

17 6 Phase portraits show relationships between states

For y''+0.1*y'+y=0 with y(0)=1,y'(0)=0, choose y horizontally and y' vertically. Damping makes the phase trajectory spiral towards the origin. In the undamped case the energy ellipse is closed in exact arithmetic. These pictures support understanding but do not certify invariants, stability or absence of numerical error.

The green marker identifies the first returned point and the red marker the last, not stable and unstable equilibria. A terminal event changes the red endpoint. A time-limited partial IVP can also end early. Read the status before interpreting the shape as a complete long-time behaviour.

17 7 Exercises and explanations

1. Derive the perturbation equation for the forced stiff example. Subtracting the derivative of cos(t) gives e'=-k*e.

2. Change the initial value to2. The exact solution becomes cos(t)+exp(-k*t). This creates a visible fast initial transient and is a stronger step-resolution test.

3. Does implicit stability guarantee the transient is accurately resolved? No. Tolerances and interpolation checks still determine accuracy.

4. Compute the falling body's impact velocity. y'=-9.81*t gives approximately-14.00714 at impact; use the same analytic event time for the reference.

5. Explain why a terminal event can have successful=true and complete=false. The event goal was reached intentionally before the full requested interval endpoint.

6. Can a tangent event be guaranteed by finer output points? No. Output density and internal event bracketing are different controls.

7. Why keep event CSV and trajectory JSON? The event list and sampled curve carry related but different information, including terminal flags and interpolation diagnostics.

8. What does a nonfinite event function indicate? The event definition failed along evaluated data; it is not a valid root merely because its sign appears to change around a pole.

17 8 A fair method comparison

Compare RK45 and SDIRK2 on the same forced problem for several k values and both initial conditions1 and2. Measure analytic error, accepted/rejected steps, derivative evaluations and elapsed time. The slowly varying exact solution and the fast transient make it possible to explain why work changes. Retain failed cases rather than excluding them from a favourable timing table.

17 9 A continuous output and event verification study

17 10 Separate interpolation error from endpoint error

A method can accept accurate endpoints while its interpolating curve has larger error between them. To test this, request sample times inside accepted segments and compare against a known solution. The exact oscillator and forced cos(t) examples provide references at arbitrary coordinates, so the experiment need not depend on another numerical solver.

For fixed RK4, compare values at aligned step endpoints with unaligned sample times such as0.013,0.333,0.777. The state update is fourth order, while the cubic Hermite continuous extension has its own error behaviour. Reducing the fixed step should improve both, but the ratios need not match exactly outside the asymptotic regime. Report where the maximum error occurred rather than only its size.

For SDIRK2 the implementation compares coarse and refined interpolation at internal quarter/midpoint locations as well as checking endpoint differences. This additional work prevents a fast initial transient from being represented by a deceptively smooth but poorly resolved curve. It is still an estimate rather than a certified bound at every point.

17 11 Quantify a phase portrait without relying on appearance

For y''+y=0,y(0)=1,y'(0)=0, every exact state lies on y^2+(y')^2=1. Export the trajectory and calculate the maximum deviation from one. Also compare phase at selected times using the known cos/sin pair. A curve can stay near the circle while having substantial phase error, so energy and phase checks complement one another.

For damping coefficient0.1, the mechanical energy decreases continuously because E'=-0.1*(y')^2<=0. Numerical samples that show a small isolated energy increase may reflect discretisation error; persistent or large growth needs investigation. The general solver does not certify a monotonic discrete energy law automatically.

17 12 Use an event with an analytic time

For y'=1,y(0)=-0.3 and event zero=y, the exact crossing is t=0.3. The state is linear, making this a transparent test of event direction, duplicate suppression and terminal stopping. A rising filter should accept it in increasing time. A falling filter should not accept that crossing as a falling event.

```
ode
y'=1
initial t=0
initial y=-0.3
interval t=[0,1]
event zero=y
event zero terminal=true
event zero direction=rising
output points=11
```

Then run backward with consistent initial data and examine the same direction convention. Do not redefine rising as merely increasing along the sequence of returned rows. The convention is tied to increasing independent-variable time.

17 13 Distinguish numerical limits from host interruption

A graceful IVP limit can return earlier accepted samples and a specific status. A force-stopped Desktop worker cannot deliver its final partial result. Compare these outcomes in a controlled example before building a workflow that relies on partial data. Keep the prior completed snapshot labelled and do not export it under the interrupted run's name.

17 14 Compare a launch event and a terminal event

17 15 An event at the starting point can have a direction

For y''=-9.81 with y(0)=0 and y'(0)=10, the projectile returns to ground at t=20/9.81, approximately 2.038736. A falling ground event should describe that return rather than the upward departure. The initial event function is zero, so the direction filter and initial-zero handling matter.

The analytic trajectory is y=10*t-9.81*t^2/2. Its derivative at the initial point is10, while at the return it is-10. These signs explain why rising and falling are physically different events. At the apex,velocity zero defines a different event with time10/9.81, approximately 1.019368.

Define one nonterminal apex event and one terminal ground event in a model, then inspect their order. The first should record maximum height and allow integration to continue; the second should stop the returned trajectory. A denser sample grid is useful for the picture but should not be the source of the analytic reference times.

17 16 Coincident or repeated crossings

Two event expressions can vanish at the same physical coordinate, yet floating-point brackets can distinguish nearly coincident times. The implementation retains coincident records according to its time agreement rules and deduplicates near repeated endpoints. Do not infer a universal symbolic equality test from a pair of similar printed times.

For periodic motion, several crossings may occur. A max step sufficiently small to bracket each relevant sign change can help, but it is not certified isolation of every event. Grazing zeros, discontinuities and poles require additional analysis. Label event counts as those located under the chosen computation rather than an unconditional count over the continuum.

17 17 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

17 18 Stiff SDIRK2 implementation

```
method = SDIRK2
rtol = 1e-7
atol = 1e-10
newton iterations = 12
```

This is the two-stage, second-order Alexander singly diagonally implicit Runge-Kutta method, gamma=1-1/sqrt(2), which is L-stable. It is an explicit method selection, not automatic stiffness detection/switching. It is NOT Radau or BDF.

Each attempted macrostep computes one full step and two half steps. The two-half solution is retained without Richardson extrapolation, preserving stiff damping. The endpoint difference divided by 3 estimates local error. Coarse-versus-refined cubic Hermite interpolants are additionally compared at quarter, midpoint and three-quarter positions; the maximum scaled discrepancy controls the step too. This extra check matters because a stiff equation can have accurate endpoints but a poorly resolved continuous curve. The reported local_error_norm includes both checks for SDIRK2. The integration remains order two; tighter tolerances can require many steps. No global-error certificate is implied.

Implicit stages use damped Newton iterations with partial-pivot dense linear solves. Smooth expressions use reverse automatic-differentiation Jacobians; finite differences are used when that derivative path is unavailable. Failed Newton solves reject and reduce the step. Repeated failure at the minimum step returns newton_failure with valid earlier data. A scale-aware default first step helps resolve fast initial transients; an explicit initial step is honoured and may be rejected. Second-order stiff accuracy and cubic interpolation do not make this a universal solver for nonsmooth, singular or nonunique systems.

Robertson kinetics is supplied in example 50, and a fast forced decay with known solution cos(t) in example 51. For unstable backward problems, implicit stability does not remove the mathematical sensitivity to perturbations.

17 19 Events

```
event ground = y
event ground terminal = true
event ground direction = falling
event tolerance = 1e-10
```

An event expression is a real-valued root function. Options may be declared before or after its expression. Up to 64 events are supported. Defaults are nonterminal and direction=any. Directions any/0, rising/1 and falling/-1 refer to increasing independent-variable time, including when integration runs backwards. Constant parameters and available lower derivative labels may appear in event expressions.

Brackets use endpoint sign changes on accepted continuous segments. Bisection locates crossings with the continuous interpolant. Event tolerance is an absolute time-bracket tolerance, default 1e-10 in double or 1e-24 in binary128, subject to floating-point time resolution. It does not certify the physical root error or force the event expression's residual to zero. Use suitable time units and tighter event tolerance for very fast dynamics.

Nonterminal events are logged and integration continues. At a terminal event the returned trajectory stops at the event state, not the requested endpoint. The status is event_terminated and integration_successful=true; integration_complete is true only if the requested endpoint was also reached. Successful terminal stopping returns CLI/DLL status 0. All events located before that earliest terminal time in the same segment are retained. Coincident events at that time are retained when their computed times agree; floating-point/bracket resolution can distinguish nearly coincident roots. Exact initial zeros are reported once; directional initial zeros use an analytic directional

derivative where available, with a scale-aware finite-probe fallback to filter them. A launch-from-ground falling event therefore need not stop an upward launch.

Grazing/tangent roots without a sign change, or multiple crossings inside one accepted segment, CAN BE MISSED. Smaller max step can help bracket them; denser reporting alone cannot. Constant-zero event functions are reported once at the initial state. Event functions must remain finite across trial accepted segments; a nonfinite function causes event_failure. This is not certified event isolation or a mechanism for automatically crossing/resetting discontinuities.

Event records are time-ordered in the integration direction and contain the name, time, complete time/state values, residual, configured direction filter and terminal flag. Near repeated endpoints are deduplicated. Event storage is capped at 10000 rows and 500000 display cells; exceeding it returns resource_limit.

17 20 DLL / JSON / builds

The C ABI remains version 1 with ten exports. Pass an ode model to eq_solve and branch on problem_type=ode_ivp / schema=equation_solver.ode_result.v1. ODE solutions is an empty array; use trajectory. New method is sdirk2 alongside classical_rk4 and dormand_prince_rk45. New fields include integration_successful, terminal_event, events, dense_output, dense_interpolant, dense_saved, dense_segments, event_evaluations, newton_iterations, jacobian_evaluations, finite_difference_jacobians, linear_solves and nonlinear_failures.

Trajectory sample values remain decimal strings. local_error_norm is the accepted step's scaled estimator (including interpolation comparison for SDIRK2), not a global or per-sample error bound; null for the initial sample and every RK4 sample. search_complete and global_error_certified remain false. Existing field ownership, thread independence and licensing rules are unchanged.

17 21 Applying the method in the Desktop

17 22 Stop on an event

```
ode
y''=-9.81
initial t=0
initial y=10
initial y'=0
interval t=[0,5]
event ground=y
event ground terminal=true
event ground direction=falling
output step=0.05
```

The falling body reaches y=0 at about t=1.42784. Solve, open ODE events and inspect the ground row and Terminal column. A terminal event intentionally stops the trajectory. Use File > Export ODE events CSV to save event records separately from the trajectory.

An event expression equals zero at a crossing. Terminal defaults to false and direction defaults to any. Allowed directions are rising, falling and any; they refer to increasing independent-variable time, including backward integrations. event tolerance controls the root-time bracket, not a rigorous physical-event error bound.

Event detection uses sign changes on accepted interpolation segments. Tangencies and several crossings within one segment can be missed. Reduce max step when appropriate; denser output alone does not improve event bracketing. Event expressions must remain finite. The solver does not automatically reset states at discontinuities or impacts.

17 23 Curves and phase portraits

Open ODE / BVP / integral plot. Choose time or the independent coordinate in Horizontal and a state in Vertical for an ordinary curve. Choose y horizontally and y' vertically for a phase portrait of a second-order equation. Green marks the first returned sample and red the last.

The curve joins returned samples; output density changes appearance, not numerical integration. Values outside display range require table/export inspection. Version 3 adds PNG and comparison-table CSV controls in the relevant field and convergence/sweep panels. Plot arithmetic is display precision; returned numerical strings preserve engine precision.

17 24 Saving continuous output

Add save dense=true if you need the interpolation coefficients in a full JSON export. Dense output is used internally whether or not it is saved. Saved segments have their own intervals; external readers must not extrapolate them. Saving is capped at 10000 segments and 500000 coefficient cells, and reaching the cap stops the run rather than discarding the requested record silently.

RK45 uses quartic interpolation, RK4 uses cubic Hermite and SDIRK2 uses piecewise cubic Hermite across half steps. A supported singular-series route can use higher-degree coefficient arrays. For ordinary Desktop plotting, selecting output points is usually sufficient and saving dense coefficients is unnecessary.

Chapter 18 Supported Singular Initial Value Problems

18 1 A zero leading coefficient does not decide existence

At x=0 the equation x*y''+y'+x*y=0 cannot be divided safely by x. Yet the regular solution with y(0)=1,y'(0)=0 is well defined. The singularity is in the explicit derivative representation; the mathematical branch can still have finite derivatives. Deciding that requires compatibility, not an arbitrary replacement x=epsilon.

Expand y=sum a_n*x^n. The constant coefficient of the equation gives a1=0. For n>=1, the coefficient of x^n is (n+1)^2*a_(n+1)+a_(n-1)=0. Starting at a0=1 produces a2=-1/4,a4=1/64,a6=-1/2304 and zero odd terms. Thus y''(0)=2*a2=-1/2. At x=1 the regular value is about 0.7651976865579666.

The recurrence is an independent mathematical explanation of the endpoint data. It also shows why merely setting y''(0)=0 would choose an incorrect local curvature. A successful numerical continuation must reproduce the original initial row and the correct series, not only avoid an exception at the first stage.

18 2 Compatibility is necessary but can leave freedom

For P(x)*y''+Q(x)*y'+R(x)*y=F(x) with P(s)=0 and finite derivatives, evaluating at s gives Q(s)*y'(s)+R(s)*y(s)=F(s). This is a compatibility condition. It can reject data immediately, but passing it does not always determine every coefficient of a regular continuation.

For x*y'=y and y(0)=0, every y=C*x is a classical solution. The initial value alone leaves the slope free. A solver must report ambiguity or request additional mathematically appropriate information; choosing a small nonzero x and an arbitrary y there would hide the missing condition.

For x*y'=2*y, branches C*x^2 can be joined with different constants on the two sides while retaining C1 regularity. A continuation criterion that considers only one side's analytic formula may therefore claim uniqueness too broadly. The implemented checks distinguish the supported classical smoothness requirements and conservatively reject unsupported branch questions.

18 3 An interior crossing must match the incoming solution

The expression (t^2-1)/(t-1) has a removable hole at t=1. Away from the hole it equals t+1. With y(0)=0, integration gives y=t+t^2/2, reaching1.5 at t=1 and4 at t=2. Matching the incoming data identifies this regular continuation without introducing a user-chosen restart.

In contrast y'=1/(t-1) integrates to ln|t-1|+C on each side. It diverges near1, so no finite regular curve crosses. A finite sample at0.99 or1.01 does not establish a finite limit. Numerical values can look unexceptional until sufficiently close to a true barrier.

Initial data for a Bessel branch started away from the singularity must be accurate enough to identify that branch. Rounding an exact regular value to a few decimal places can introduce a component incompatible with the desired finite continuation. Input uncertainty should not be silently erased to force a preferred analytic answer.

18 4 Scope is structural rather than universal

The supported recogniser handles scalar linear first- and second-order IVPs with a simple affine leading coefficient and bounded polynomial/rational coefficient structure. This includes some explicit divided forms and removable affine-denominator holes. It does not cover every expression that has a removable mathematical limit.

For example sin(x)/x has a well-known finite limit, but lies outside this polynomial-linear continuation grammar. Singular coupled systems, nonlinear state-dependent singularities, multiple or irregular singular points, logarithmic

branches and arbitrary fractional powers require different analysis. A clear unsupported diagnostic is preferable to silently assuming the same method applies.

The regular BVP solver has a separate scope. A successful Bessel IVP at zero does not imply automatic singular BVP endpoint support. Likewise ordinary domain checks remain necessary for unrecognised functions; the singularity screen does not prove there are no hidden poles anywhere in an arbitrary expression.

18 5 Interpret the series method honestly

When selected, regular_singular_series is the actual method across the supported requested interval. The report retains the requested RK4/RK45/SDIRK2 method separately. Output spacing and events can still be used, but the values are not obtained by pretending ordinary RK stages were defined at the singular point.

The implementation constructs a bounded rational recurrence and selects a degree from 16,32,48 or64 using tail and roundoff estimates. Component atol governs those absolute checks. A longer interval can exceed a safe truncation/cancellation budget even though a local regular solution exists. singular_series_limit then describes the numerical method's bounded capability, not mathematical nonexistence.

Binary128 can improve summation in a suitable case but does not remove recogniser or series-degree limits. The estimated tail is not a certified global bound, and finite-rank compatibility checks are not exact existence theorems for rounded experimental data. Keep those distinctions in captions and conclusions.

18 6 Exercises with mathematical answers

1. Derive the x^4 Bessel coefficient from a2=-1/4. The n=3 recurrence gives16*a4+a2=0, hence a4=1/64.

2. If y(0)=2 on the same regular branch, what is y''(0)? Linearity gives y''(0)=-1. The regular solution is twice the unit-normalised one.

3. Is x*y'=y,y(0)=0 determined? No, because C remains free in y=C*x.

4. For the same equation starting at x=-1,y=-1, what regular linear branch fits? y=x. A crossing still needs the implemented smoothness/continuation checks, not just this candidate formula.

5. What is the removable-hole model's value at t=1? It is1.5 by the regular antiderivative and continuity.

6. Why does y'=1/(t-1) fail to cross regularly? Its logarithmic primitive diverges at1.

7. Can successful syntax checking establish regularity? No. It validates grammar/settings; compatibility and series safety require numerical/structural analysis during solving.

8. Why retain requested_method in an export? It records user intent while method records the computation actually used, avoiding misleading claims about RK stepping.

18 7 A singularity classification laboratory

Prepare four cases: determined regular Bessel data, an ambiguous first-order family, a genuine divergent pole and a mathematically removable but unsupported expression. Predict the classifications before running. Compare original implicit and supported divided forms where applicable. Save reports even when no curve is returned.

For an interior example, approach the singularity from both directions with consistent data and check the same regular function. Then perturb the data deliberately and explain whether failure is due to incompatibility, ambiguity or a method limit. This is a study of mathematical information, not a competition to find an epsilon that avoids an error message.

18 8 A tolerance study of a regular series

18 9 The local expansion has a finite numerical budget

The Bessel coefficients alternate and decrease rapidly near zero. On a longer interval, more terms and cancellation can matter. The implemented degree cap means that even a mathematically entire function can exceed this particular safe representation budget for a requested interval and tolerance. A series-limit diagnostic is therefore not equivalent to a finite radius of existence of the mathematical solution.

For a controlled experiment, run the unit Bessel initial data on[0,1] with several component atol values, retain the selected term count and compare with an independent high-precision series. Repeat in eligible binary128 mode if available. Distinguish a tighter numerical check from an independently certified error bound. Report any rejected request rather than retaining only successful outputs.

The original initial point is important evidence. An epsilon-start workaround would return a different first coordinate or require user-calculated nearby values. Check that x=0,y=1,y'=0 remain exactly represented in the initial row while the later trajectory follows the regular branch. This is part of the input/output contract, not just a plotting preference.

18 10 Compare equivalent expressions carefully

For supported syntax, the implicit equation x*y''+y'+x*y=0 and the divided expression y''=-y-y'/x describe the same nonsingular relation away from zero. At zero the explicit form has a domain hole. The recogniser can analyse supported rational structure to recover a compatible regular formulation, but it does not erase arbitrary domain exclusions in unrelated expressions.

Equivalent mathematics can exceed structural resource limits when written in an unnecessarily complicated form. A rejection should identify unsupported analysis rather than silently choosing a branch. Simplify with a justified algebraic transformation, retain the original model for verification and state which form was actually solved.

18 11 A failure record is useful teaching evidence

Keep four saved projects with determined, ambiguous, incompatible and unsupported cases. In each report state the intended class of solution, required smoothness and which data determine free coefficients. This makes the failure modes part of the mathematics lesson rather than treating every diagnostic as a software inconvenience to suppress.

18 12 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

18 13 The original Bessel input now works unchanged

```
ode
x*y'' + y' + x*y = 0
initial x = 0
initial y = 1
initial y' = 0
interval x = [0, 1]
step = 0.1
```

There are eleven output rows including the original x=0, y=1, y'=0. At x=1, y is approximately 0.7651976865579666, and y' approximately -0.4400505857449335 in double arithmetic. Extended precision preserves more digits. The user need not calculate a nearby starting point or alter the equation.

For this equation the engine derives the regular expansion

```
y = 1 - x^2/4 + x^4/64 - x^6/2304 + ...
```

and y''(0)=-1/2. It does not substitute an arbitrary epsilon, treat 0/0 as zero, or simply replace the highest derivative at a single stage. The same recognised regular solution is evaluated across the requested interval using a checked power-series representation centred at the singular point.

18 14 Supported equations

One scalar first- or second-order linear equation, with a simple affine leading coefficient, is supported after exact polynomial/rational algebra:

```
P(x)*y'  + R(x)*y = F(x)
P(x)*y'' + Q(x)*y' + R(x)*y = F(x)
```

P(x)=a*(x-s), a nonzero. Q,R,F are polynomials of degree at most eight. Coefficients may be rational/decimal/scientific numbers or evaluated model constants. Highest derivatives and all state terms must be linear. The variable need not be named x. Supported time-only affine denominators can be cleared, so explicit divided forms and removable rational holes can also be recognised:

```
ode
y'' = -y-y'/x
initial x=0
initial y=1
initial y'=0
interval x=[0,1]
step=.1
ode
y' = (t^2-1)/(t-1)
initial t=0
initial y=0
interval t=[0,2]
step=.1
```

The latter continues across t=1 and returns y=t+t^2/2. Common time factors in implicit equations are supported as well. Some equivalent but more complicated rational forms can exceed the recogniser's degree/resource limits and are rejected. Named constants are evaluated at the selected engine precision before recognition; written numeric literals are handled as exact decimal rational values in the structural algebra. Domain exclusions in unsupported expressions are not erased.

The singular point may be at the initial endpoint, final endpoint, or in the interior, with forward or reverse integration. For an interior/final singularity, the initial state must agree with a supported regular continuation from its nonsingular starting point. Sufficiently precise initial data matter: rounding a Bessel value to four digits does not supply the exact regular Bessel branch. Example 63 supplies precise data and crosses x=0.5 without a user-created restart.

18 15 Compatibility and branch decisions

At a second-order singular point s, finite classical derivatives require

```
Q(s)*y'(s) + R(s)*y(s) = F(s).
```

The power-series recurrence determines higher derivatives when it is nondegenerate. For Bessel, the next coefficient gives y''(0)=-y(0)/2. The engine derives affine series coefficients and solves a small rank-checked system containing the compatibility equations and the original initial data. It does not prescribe free coefficients as zero to manufacture a solution.

- Compatible, determined regular continuation: apply the checked series.

- Incompatible data / no supported finite regular branch: singular_incompatible.

- Unspecified regular coefficient or classical continuation freedom: singular_ambiguous.

- Fractional branches needing additional smoothness analysis, unsupported equations, or resource caps: singular_unsupported.

For example, x*y'=y with y(0)=0 admits y=C*x; it is reported as ambiguous. The same equation starting at x=-1, y=-1 can be continued as y=x with matching C1 regularity. In contrast x*y'=2*y can be joined across zero with different coefficients on each side while retaining C1 regularity, so this automatic continuation is rejected as ambiguous. Analogous C2 checks apply at order two. Fractional branches between available initial derivatives and the required classical smoothness are conservatively unsupported, not asserted to be uniquely solved or to have multiple classical solutions.

A true pole such as y'=1/(x-s) has no finite regular continuation across s. The engine detects the obstruction before attempting to cross it. This can give an earlier and more informative failure than the previous minimum-step or nonfinite-stage diagnostic. It does not prove nonexistence of arbitrary weak, discontinuous, complex or nonclassical solutions.

18 16 Actual method and error checks

When this feature is applied, the actual method is regular_singular_series. The requested RK4/RK45/SDIRK2 method is retained in the report and JSON separately. The power series replaces ordinary RK/implicit stages on this supported interval; the engine does not claim that these are classical RK4 steps.

A requested step=.1 still gives .1-sized segments and the normal reporting grid, including the original start and exact requested endpoint. Adaptive method controls continue to bound/report segments, but the solution values come from the regular series. Output points, output step, sample times, phase plots and event handling remain available. Saved dense polynomials represent the series on each segment, not an RK4 cubic or RK45 quartic. Each saved segment may therefore have more coefficient rows than previous methods. Clients must read the array dimensions.

The exact rational recurrence is constructed through degree 64. A degree 16, 32, 48 or 64 representation is selected only when the final 16-term block and a conservative coefficient-formation/cancellation roundoff estimate both fit component absolute tolerance budgets over the requested interval. Series values must also reproduce the original initial state within their numerical budget. In this route, component atol governs the series checks; loosening rtol does not relax the absolute summation/cancellation requirement. Binary128 can help with tight tolerances or cancellation; it does not remove the finite series cap.

These are numerical truncation and floating-point estimates, not a certified global error bound. The recogniser and compatibility checks also use finite precision for data fitting and rank tests. This is not a general symbolic limit, existence, or uniqueness prover. As elsewhere, global_error_certified remains false.

An interval too long for safe summation returns singular_series_limit, without falling back to an unsafe epsilon or silently returning a poor approximation. Shorten the interval or use extended precision. This first implementation screens the entire requested interval before integration. Thus a later singular barrier or unsafe series radius can cause an early failure even if an event might have stopped sooner. In that case request the shorter interval explicitly.

18 17 Scope, exclusions and resource limits

This feature is for scalar IVPs. It does not add singular BVP boundary conditions, singular coupled-system continuation, arbitrary nonlinear singularities, state-dependent singular surfaces, multiple/irregular singular points, logarithmic or fractional-power solution branches, or continuation through genuine divergence. The regular BVP solver remains available with its previous scope.

Affine/quadratic time denominators and polynomial zero factors of leading coefficients are screened for known barriers. This includes supported factors inside some otherwise unsupported expressions and coupled/higher-order models; these produce singular_unsupported rather than being silently stepped over. This is not an exhaustive

detector for poles of arbitrary functions, such as tan, or all implicit domain restrictions. Ordinary domain/stage checks still apply to unrecognised models; no claim to detect every singularity is made.

Examples such as sin(x)/x may have a removable limit but are outside the current polynomial linear continuation grammar. Their detected hole is reported as unsupported, not silently approximated. Higher-degree leading coefficients, higher-order singular ODEs and more general symbolic cancellation can be added as separate validated extensions.

Recognition is bounded by 4096 AST nodes, degree eight numerator terms, a small term count, and 32768-bit rational numerator/denominator resource guards. Initial compatibility is checked with scaled floating rank/residual tolerances; it is not an exact theorem about decimal-rounded physical data. The usual time/cancellation, step, output and event limits remain. Saved dense-output limits now count actual coefficient cells, including the larger series arrays (at most 500000 cells).

18 18 Desktop / JSON / DLL

The Desktop has two built-in examples under Examples > Differential Equations: automatic Bessel start and removable interior pole. Help > Singular ODE Points Guide opens this document. The usual table, ODE / BVP / phase plot, reports and CSV/JSON exports apply. A failed differential solve opens the Run report; a single initial row is labelled as initial data with incomplete integration, not success.

The existing ODE schema and C ABI v1 remain. Additional fields are:

- method: regular_singular_series when applied.

- requested_method: the originally selected classical_rk4, dormand_prince_rk45 or sdirk2.

- regularization_applied: boolean.

- regularization: detected, center, series_terms, series_segments, scaled_tail_estimate, scaled_roundoff_estimate, compatibility_checked.

- dense_interpolant: regular_singular_power_series when applied.

- adaptive_error_control: false for this series route; this is not an embedded RK controller.

For this method, trajectory local_error_norm contains the shared scaled series truncation/roundoff estimate, not an embedded RK step estimator. The first row remains null. Failed regularisation leaves regularization_applied=false and uses a specific singular_* status with CLI/C ABI status 2. Successful integration or terminal event stopping retains status 0. Invalid syntax/settings use status 1. Syntax checking alone does not analyse regularity or compatibility.

The original initial row is retained. Numerical failures retain the existing IVP partial-trajectory contract, so an early structural rejection may contain only that row. Inspect integration_successful, integration_complete, status and reason. All ten exports and ownership rules are unchanged; free every result once with eq_result_free. No new runtime DLL, Python or SciPy dependency is introduced.

18 19 Applying the method in the Desktop

18 20 Bessel start at zero

```
ode
x*y''+y'+x*y=0
initial x=0
initial y=1
initial y'=0
interval x=[0,1]
step=0.1
```

Use Examples > Differential Equations > ODE: automatic Bessel start at zero. The original row x=0, y=1, y'=0 is retained. The expected regular value is y(1) approximately 0.7651976865579666. The expansion begins 1-x^2/4+x^4/64, giving y''(0)=-1/2.

For this supported structure the actual method is regular_singular_series. The engine checks a regular expansion across the requested interval rather than evaluating undefined RK stages or moving the start to an arbitrary nearby number. Run report and JSON distinguish the requested method from the actual method.

18 21 Interior removable holes

```
ode
y'=(t^2-1)/(t-1)
initial t=0
initial y=0
interval t=[0,2]
step=0.1
```

This supported hole at t=1 has the regular continuation y=t+t^2/2. Crossing an interior singularity requires compatibility with the original data and the solution approaching it. A finite nearby value alone is insufficient.

Supported automatic continuation is limited to scalar linear first- or second-order IVPs with a simple affine leading coefficient and bounded polynomial/rational structure. It is not a universal limit calculator. Singular BVPs, nonlinear singularities, coupled singular systems and arbitrary logarithmic or fractional branches remain outside this feature.

18 22 Understand the diagnostic

Status	Meaning and next action
singular_incompatible	Data do not fit the supported finite regular branch; check the model and initial data
singular_ambiguous	Conditions leave continuation freedom; supply mathematically justified information
singular_unsupported	Structure or branch analysis is outside the implemented class
singular_series_limit	The bounded series cannot meet its truncation/roundoff checks; shorten the interval or assess extended precision

For x*y'=y with y(0)=0, every y=C*x is allowed; the application must not silently choose C. A genuine pole cannot be fixed by an arbitrary epsilon. Series checks are numerical tail and roundoff estimates, not certified existence, uniqueness or global error bounds. F4 does not perform this compatibility analysis.

Chapter 19 Regular Two Point Boundary Value Problems

19 1 Boundary conditions select an entire curve

For y''=-2, integration gives y=-x^2+C1*x+C2. Applying y(0)=0 gives C2=0; y(1)=0 gives C1=1. Therefore y=x*(1-x) and y'=1-2*x. The midpoint0.25 is a convenient reference, while endpoint values alone cannot verify the interior because every admissible trial curve can be made to match them.

A BVP solver determines all mesh states together. It is not simply an IVP with a missing slope filled arbitrarily. Local Newton iteration needs a starting curve, and adaptive collocation refines intervals whose continuous defect is too large. The boundary residual and differential defect are separate quantities and both should be inspected.

19 2 Solve the logarithmic example independently

The requested problem x^2*y''+x*y'=1 on [1,1.4] with y(1)=0 and y(1.4)=ln(1.4)/2 is regular because x never vanishes. Observe that (x*y')'=x*y''+y', so division by x gives (x*y')'=1/x. Integrating yields x*y'=ln(x)+C1 and y=(ln(x))^2/2+C1*ln(x)+C2.

At x=1 both logarithms vanish, so C2=0. Let L=ln(1.4). The final condition gives L^2/2+C1*L=L/2, hence C1=(1-L)/2. The exact expression is y=(ln(x))^2/2+(1-L)*ln(x)/2. It satisfies both endpoint data and the differential equation by differentiation.

This derivation avoids a common mistake: treating x^2*y''+x*y' as the derivative of x^2*y'. That derivative is x^2*y''+2*x*y', a different expression. An apparently small algebraic slip changes the BVP. Always differentiate the proposed integrating factor or reduced form explicitly.

19 3 Derive the Robin example

The equation y''-2*x*y'-2*y=-4*x can be written as (y'-2*x*y)'=-4*x. Integrating once gives y'-2*x*y=-2*x^2+C. At x=1 the boundary condition2*y(1)-y'(1)=1 implies y'(1)-2*y(1)=-1, hence C=1.

The function y=x is a particular solution of that first-order equation, and the homogeneous solution is K*exp(x^2). Therefore y=x+K*exp(x^2). At zero, y(0)=K while y'(0)=1, so y(0)-y'(0)=0 gives K=1. The final answer is y=x+exp(x^2), not y=x. The latter satisfies the differential equation but fails the left boundary condition.

This example is a strong acceptance test because checking only the ODE would accept the wrong particular solution. A BVP answer must satisfy every boundary residual as well as the differential equation. At the right endpoint,2*y-y'=1 follows directly from the once-integrated relation.

19 4 Guesses can select different nonlinear branches

For Bratu's equation y''=-exp(y),y(0)=y(1)=0, the zero starting guess can converge to a small positive branch while a sufficiently large positive guess can converge to a large branch. The supplied example uses guess y=0; trying3*sin(pi*x) illustrates branch dependence. Guesses are not additional equations and do not establish completeness.

An analytic family makes this more concrete: y(x)=2*ln(cosh(theta/2)/cosh(theta*(x-1/2))) satisfies the zero endpoints. Substitution gives the parameter condition2*theta^2/cosh(theta/2)^2=1 for this coefficient. It can have two positive theta values, leading to different peaks. The scalar branch equation provides independent evidence of multiplicity beyond observing two numerical curves.

For systems, supply independent guesses for derivative components when helpful. A guess for y' is not automatically obtained by differentiating guess y. Unknown eigenparameters are not added merely by typing an undeclared constant name; the current BVP scope does not solve such parameter problems.

19 5 Mesh accuracy is separate from output density

An output step of0.01 controls the displayed sampling. It does not force a uniform collocation mesh of100 intervals. The adaptive mesh is chosen from defect checks, and the final curve is sampled through its interpolant. Increasing output rows can make a picture smoother without improving the underlying solution.

The defect is the difference between the interpolant derivative and the ODE's vector field. Its scale depends on atol and rtol. Boundary tolerance checks endpoint residuals directly. A small defect is useful numerical evidence but is not a certified global solution error, especially for sensitive BVPs where small perturbations can change the curve strongly.

Limits on nodes, unknowns, sparse entries, refinements and time can stop a valid mathematical problem. Failed BVP calculations return no converged trajectory. Report residual diagnostics as the last verified numerical evidence, not as an accepted final curve. Distinguish singular Jacobian from a demonstrated proof of no mathematical solution.

19 6 Practice and answers

1. Solve y''=0,y(0)=2,y(1)=5. The linear solution is y=2+3*x.

2. Replace the final value in Problem 1 by y'(1)=5. Then y=2+5*x; endpoint derivative data change the slope rather than the final value directly.

3. Verify y=x alone in the Robin example. It satisfies the ODE, but y(0)-y'(0)=-1, so it is rejected.

4. Evaluate the logarithmic solution at x=1. It is zero because ln(1)=0, confirming C2=0.

5. Why are two conditions not always independent? Repeating the same endpoint relation twice adds no information and can make the Newton system rank deficient.

6. Does a denser reported curve certify more accuracy? No; mesh/defect convergence must be checked separately.

7. Does a guess that finds one Bratu branch exclude the other? No. It demonstrates a local convergence outcome.

8. Does the Bessel IVP feature solve a singular BVP at zero? No. The regular two-point BVP and the supported scalar singular IVP are distinct capabilities.

19 7 A boundary condition study

Use the exact quadratic, logarithmic and Robin examples to compare sampled errors against independent formulas. Tighten defect and boundary tolerances separately and record mesh changes. Then study Bratu with two guesses and a stiff boundary layer with increasing max nodes. Preserve unsuccessful attempts with their reasons; they reveal the difference between mathematical existence and numerical convergence.

19 8 A first order BVP with endpoint coupling

19 9 Count states rather than equation lines alone

A first-order equation y'=0 has constant solutions. The condition boundary left y=2 fixes y=2 across the interval, so one scalar condition matches one state. Adding boundary right y=3 makes the system overdetermined and inconsistent. The number of required conditions comes from total differential order after conversion, not from how many printed equations fit on a screen.

For a second-order equation y''=0, two independent conditions determine the linear function. Coupled endpoint conditions such as y(0)-y(1)=0 and y'(0)=1 are inconsistent, because a linear function with equal endpoint values has zero slope. A correct condition count does not establish compatibility. This distinction should be visible in both the mathematical derivation and the diagnostic interpretation.

19 10 Boundary layers need local mesh work

The equation y''=10000*y with y(0)=1,y(1)=0 has solution sinh(100*(1-x))/sinh(100). Most variation lies near the left endpoint on a scale about 0.01. A uniform coarse output grid can miss the shape even when the adaptive collocation mesh resolves it internally. Conversely, a dense output request cannot force a poor internal solution to become accurate.

Use a stable independent expression when checking this function numerically. Direct sinh(100) is finite in double, but larger coefficients can overflow; an exponential ratio form can avoid unnecessarily large intermediates. Reference evaluation itself needs numerical care before it can judge the solver.

19 11 Report local convergence honestly

A successful Newton iteration is one local outcome for a selected guess and mesh. It does not enumerate all BVP solutions. A singular-Jacobian failure can arise from redundant boundary conditions, a poor iterate or a structurally ill-posed problem. Separate those possibilities with small analytic examples before making a general claim about existence.

For screenshots include the guess, mesh limits and stopping reason where they matter. A pair of visually different Bratu curves should be accompanied by their input guesses and residual checks, not merely labelled first and second solution. A failed request should not show an old retained curve as if it were its result.

19 12 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

19 13 First example

```
bvp
y'' = -2
interval x = [0, 1]
boundary left y = 0
boundary right y = 0
output step = 0.1
```

The solution is y=x*(1-x); y(0.5)=0.25. The table includes x, y and y'. Use bvp, not ode. There are no initial-value directives or fixed RK steps. output step sets reporting spacing; it does not fix the collocation mesh. The initial and final endpoints are included even if the spacing does not divide the interval. Use output points instead to request a fixed count including endpoints.

19 14 Equations and boundary conditions

Explicit first-order systems and direct equations up to order eight are accepted. Each highest derivative must enter affinely, just as for direct higher-order IVPs. A system may mix orders. After conversion there may be 1..16 first-order states. Exactly that many scalar boundary conditions are required. Highest derivatives of different unknowns must already be isolated into separate equations.

Here left means the first interval endpoint a, right means the last endpoint b. The interval must have finite a < b. Its independent variable can be x, t, etc.

```
boundary left y = 1                 # Dirichlet
boundary right y' = 0               # Neumann
boundary left y + 2*y' = 3          # Robin
boundary left(y) - right(y) = 0     # Coupled endpoint condition
boundary left(y') - right(y') = 0   # Periodic derivative condition
boundary left(y^2) + right(y) = 2   # Nonlinear endpoint condition
```

These lines illustrate syntax; select the appropriate independent conditions, not every line at once. In shorthand, the whole expression between left/right and '=' is evaluated at that endpoint. General left(expression) and right(expression) can appear on either side of '=' and include constants, functions, the independent variable, states and lower derivatives. Bare state variables outside left/right expressions are rejected. Endpoint operators cannot be nested. Duplicate/dependent conditions lead to a singular-Jacobian failure; having the correct count alone does not establish independence.

Example of coupled second-order equations:

```
bvp
u'' = v
v'' = 0
interval x = [0, 1]
boundary left u = 0
boundary right u = 1
boundary left v = 0
boundary right v = 0
```

The solution is u=x, v=0. Four scalar conditions match total order four. A fourth-order clamped-beam example is supplied in examples/58_bvp_clamped_beam.txt.

19 15 Initial guesses and multiple solutions

A BVP finds one local solution from a guess for the solution across the interval. This is not an initial condition. Guesses may be expressions in the independent variable and constants, but may not depend on unknown state values. Missing component guesses default to zero, including derivative components. A derivative guess is independent; it is not automatically differentiated from the base guess.

```
bvp
y'' = -exp(y)
interval x = [0, 1]
boundary left y = 0
boundary right y = 0
guess y = 0
rtol = 1e-6
output step = 0.01
```

This is Bratu's equation. guess y=0 selects the small branch (peak about 0.14054). Use guess y=3*sin(pi*x) for the large branch (peak about 4.09147). An optional guess y'=3*pi*cos(pi*x) can help convergence. The engine does not search or enumerate all branches and does not establish existence or uniqueness. Some problems have no solution; others have many. Newton failure does not prove that no solution exists. Try a better guess, a different initial mesh or continuation by manually changing a constant through a sequence of nearby problems.

Constants use constant k=... / const k=... and can depend on other constants. They are fixed input values, not unknown parameters. Parameter/eigenvalue fitting with extra boundary conditions is not included in this supported mode.

19 16 Numerical method and accuracy

The engine implements fourth-order Hermite-Simpson / Lobatto IIIA collocation: it solves for all nodal states together. On each interval the midpoint state is (y_left+y_right)/2-h*(f_right-f_left)/8 and the integral condition uses Simpson weights 1,4,1. Damped Newton uses local analytic expression Jacobians when available, with finite-difference fallback otherwise. A sparse, partially pivoted elimination handles the local block structure and the endpoint border. There is no external numerical library or new runtime DLL dependency.

A continuous C1 cubic Hermite interpolant connects each pair of mesh nodes. At five interior locations per interval (including off-collocation points) the engine checks the defect p'(x)-f(x,p(x)) with component scale atol + rtol*abs(f). Intervals with scaled defect >1 are split in two or three, and Newton solves the new mesh. Boundary conditions have an independent absolute boundary tolerance, in the units/scaling of the expressions as written. Rescale badly scaled boundary equations if necessary.

Success requires both the sampled continuous defect <=1 and the maximum absolute boundary residual <= boundary tolerance. These are residual diagnostics, not a certified bound on global solution error. Ill-conditioning, resonance, rapid oscillations, discontinuities or features between residual probes can make an accurate residual misleading. Compare tighter tolerances and denser initial meshes for demanding work. Output density alone does not improve the solve mesh.

Reference context: SciPy solve_bvp documentation (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_bvp.html) describes the same broad family of fourth-order collocation and residual control. Patodia's implementation and simplified limits are its own; this is not a port of SciPy, bvp4c, BDF, Radau or a certified interval BVP algorithm.

19 17 Controls (US notation)

Directive	Default	Allowed / meaning
method	collocation	Only collocation for BVPs
rtol	1e-6	Relative continuous-defect scale, 100*machine epsilon .. 1
atol	1e-8	Positive absolute continuous-defect scale
boundary tolerance	1e-8	Absolute boundary residual; >=100*machine epsilon
mesh points	11	Initial uniform nodes, >=3 and <=max nodes
max nodes	1001	Adaptive node cap, at most 4000
newton iterations	25	Iterations per mesh, 1..100
max refinements	16	Refinement rounds, 1..30
output points	201	Sample rows including endpoints, 2..9999
output step	unset	Positive spacing; alternative to output points

At most 6000 nodal unknowns, 500000 sparse matrix entries and 500000 output cells are allowed. Node limits can stop refinement even when the problem is solvable. Use the Desktop solver time budget or CLI --seconds; zero means unlimited. The soft budget covers numerical work, not parsing, licensing or formatting. Cancellation and time limits return explicit status without a solution curve. Desktop Stop kills the worker, preserving the prior completed run snapshot.

Use Compute > Precision or --extend for binary128. Licensing is unchanged. EU style uses decimal commas and semicolons inside brackets/function arguments, for example interval x=[0;1] and output step=0,1.

19 18 Desktop and exports

Examples > Boundary Value Problems opens three editable built-in examples. The Solutions table shows the independent variable, states and derivative states. The ODE / BVP / phase plot supports coordinate-versus-state and state-versus-state views. CSV/JSON and run report export retain the existing exact decimal strings. The Run report

includes Newton/mesh information, maximum scaled defect and all boundary residuals. Help > Boundary Value Problems Guide opens this document. Projects retain the BVP model and solver settings in the usual .pdeq format.

If a BVP fails, no converged trajectory is returned; the report opens directly. Failed IVP solves also open the report, making the previous initial-row-only behaviour clearer. An initial row is initial data, not proof of integration.

19 19 CLI / DLL contract

CLI usage: equation_solver.exe --file examples/54_bvp_dirichlet.txt --json --seconds 20. Syntax checking: add --check; this checks syntax/names/counts, not solvability, Jacobian rank, domains along the unknown solution, or boundary compatibility.

BVP JSON uses schema=equation_solver.bvp_result.v1, problem_type=ode_bvp, method=lobatto_collocation_4. variables includes the independent variable first and readable lower derivative labels. trajectory[].values contains corresponding decimal strings. solutions is empty (it is the algebraic array). Do not look for IVP integration_complete or interpret mesh nodes as IVP steps.

bvp_successful=true, status bvp_converged, and CLI/C ABI status 0 mean the numerical residual criteria were met. Numerical nonconvergence returns status 2, bvp_successful=false and empty trajectory. Syntax/configuration errors use status 1 and the existing error contract. Statuses include nonlinear_failure, singular_jacobian, nonfinite_equation, nonfinite_guess, mesh_limit, mesh_resolution_failure, resource_limit, time_limit and cancelled.

Diagnostic fields: mesh_nodes, mesh_refinements, newton_iterations, jacobian_evaluations, finite_difference_jacobians, derivative_evaluations, linear_solves, rtol, atol, boundary_tolerance, maximum_scaled_defect, maximum_boundary_residual, boundary_residuals, interval, precision and seconds. Residual fields can be null/empty when the solve stops before verification. On failure after refinement, residual fields describe the last verified mesh, which may precede the final attempted mesh_nodes. They do not label an unconverged iterate as a solution. search_complete, global_error_certified and uniqueness_established remain false. C ABI version 1 and all ten exports are unchanged; use eq_result_free once per returned result. See tests/bvp_dll.py.

19 20 Scope and singular endpoints

This stage supports regular two-point, finite-interval ODE BVPs. It does not add DAEs, unknown eigenparameters, interior/multipoint conditions, integral constraints, PDEs, integral equations, continuation automation, singular mass matrices or automatic treatment of regular-singular endpoints.

In particular, x*y''+y'+x*y=0 on [0,1] still has an undefined explicit derivative at x=0. Its regular Bessel solution exists, but a singular-endpoint formulation is a separate numerical capability. This BVP release detects the nonfinite equation and returns no curve; it does not silently move the endpoint, invent a limit, or claim to have solved the exact singular model. For a nonzero left endpoint, use boundary data appropriate to that different finite interval.

19 21 Applying the method in the Desktop

19 22 Choose bvp for endpoint conditions

```
bvp
y''=-2
interval x=[0,1]
boundary left y=0
boundary right y=0
output step=0.1
```

The solution is y=x*(1-x), so y(0.5)=0.25. Choose the built-in BVP: fixed endpoint temperatures example to begin. Unlike an IVP, a BVP supplies constraints at endpoints and finds a whole curve. Do not add initial directives or an RK4 step.

The finite interval must have a<b. Left means the first endpoint and right the final endpoint. Supply as many independent scalar boundary conditions as the total first-order state count. Direct orders up to eight and at most 16 expanded states are supported. Mixed endpoint conditions can be linear or nonlinear within the supported equation grammar.

19 23 Robin and coupled conditions

```
bvp
y''-2*x*y'-2*y=-4*x
interval x=[0,1]
boundary left y-y'=0
boundary right 2*y-y'=1
output points=101
```

This requested Robin problem has y=x+exp(x^2). The shorthand evaluates its entire expression at the named endpoint. For coupled endpoint conditions use forms such as boundary left(y)-right(y)=0. Use independent conditions; a correct count alone does not prevent a singular Jacobian.

```
bvp
x^2*y''+x*y'=1
interval x=[1,1.4]
boundary left y=0
boundary right y=ln(1.4)/2
output points=101
```

For this second requested problem, write L=ln(1.4). The analytic result is y=(ln(x))^2/2+(1-L)*ln(x)/2. Its differential coefficient is regular on this interval.

19 24 Guesses mesh controls and branches

A guess is a starting curve for the numerical iteration, not a boundary condition. Use guess y=... and, when useful, independent derivative-component guesses. Missing guesses default to zero. They cannot depend on unknown states. Different guesses can converge to different branches.

```
bvp
y''=-exp(y)
interval x=[0,1]
boundary left y=0
boundary right y=0
guess y=0
output step=0.01
```

For this Bratu example the small branch has peak about 0.14054. Changing the guess to 3*sin(pi*x) can find a large branch with peak about 4.09147. This does not enumerate every solution or establish uniqueness.

The adaptive collocation solver uses defect and boundary residual criteria. Defaults include rtol=1e-6, atol=1e-8, boundary tolerance=1e-8, mesh points=11, max nodes=1001, newton iterations=25 and max refinements=16. Output points or output step changes reporting rather than fixing the collocation mesh. Tightening tolerances can require more nodes and time.

A converged result reports bvp_converged and bvp_successful=true in JSON. Failure returns no converged trajectory; read Run report for singular_jacobian, nonlinear_failure, mesh_limit or other reasons. Global error and uniqueness are not certified. Singular endpoints, unknown eigenparameters, multipoint conditions and DAEs are not included. Singular IVP support does not make singular BVPs supported.

19 Boundary value jobs inside a workflow

```
workflow
bvp parabola
    y'' = -2
    interval x = [0,1]
    boundary left y = 0
    boundary right y = 0
    output step = 0.5
end
point = at(parabola.result.trajectory, 2)
report at(point.values, 2)
```

Expected y(0.5) = 0.25.

Sweep a load or boundary value by substituting a loop variable in the model. Retain the boundary residual and convergence evidence for each case.

Chapter 20 Laplace and Poisson Equations on Rectangles

20 1 From a differential operator to an interior balance

For a smooth field u(x,y), centred differences approximate u_xx by (u_E-2*u_P+u_W)/hx^2 and u_yy by (u_N-2*u_P+u_S)/hy^2. Laplace's equation sets their sum to zero. On a square grid with hx=hy=h, rearrangement gives u_P=(u_E+u_W+u_N+u_S)/4. The four neighbours are axial, not diagonal.

For Poisson's equation u_xx+u_yy=f, the corresponding formula is u_P=(u_E+u_W+u_N+u_S-h^2*f_P)/4. The sign of the source contribution matters. A code or handwritten calculation using the opposite sign solves the opposite forcing problem. Substituting a manufactured solution is an efficient way to detect that error.

When hx and hy differ, the neighbour weights are not all one quarter. The denominator is2/hx^2+2/hy^2, and the numerator is(u_E+u_W)/hx^2+(u_N+u_S)/hy^2-f_P. This is why equal node counts do not imply equal physical spacing on a nonsquare rectangle. Record both domain lengths and grid counts.

20 2 A one-unknown problem can be solved by hand

Use a unit square with grid x=3 and grid y=3. There is one interior node at(0.5,0.5). Prescribe boundary u=x on all sides: left0,right1,bottomx,topx. The corner data agree and the exact solution is u=x. The interior neighbour values are0,1,0.5,0.5, giving average0.5.

```
pde
type=poisson
u_xx+u_yy=0
domain x=[0,1]
domain y=[0,1]
boundary left u=0
boundary right u=1
boundary bottom u=x
boundary top u=x
grid x=3
grid y=3
refine=false
```

This deliberately tiny fixed-grid model has nine returned nodes. Every boundary and interior value is independently checkable. With refine=false the values solve that requested discrete system. With automatic refinement enabled, returned coordinates remain the same but values come from finer internal solutions sampled there.

20 3 A four-unknown system explains iteration

Take four interior unknowns in a2 by2 interior block with surrounding values. Each equation couples one unknown to its interior neighbours and to known boundary contributions. Updating one value from the four-neighbour average while leaving others at old estimates gives an iteration, not an independent closed-form solution of the entire coupled system.

Starting from all zeros, a first sweep propagates boundary influence only according to the update order and previous guesses. Later sweeps change those values again. A number such as2.44 after one sweep may be an intermediate estimate rather than the converged value at that coordinate. To reproduce any particular list, the grid, boundary ordering, initial guesses, iteration rule and stopping criterion must be specified.

The Desktop's Poisson method solves the coupled discrete system using its documented iterative method, not an arbitrary fixed count of manual sweeps. Compare a reference list only after matching node coordinates and the exact boundary interpretation. A sequence of five values without coordinates does not identify a unique row or section of the field.

20 4 Interpreting the supplied boundary data

The supplied corrected boundary samples were top all0,right all0,left15.45,29.39,40.45,47.56,50 and bottom50,47.56,40.45,29.39,15.45. Their interpretation requires the original figure: whether endpoints are included, which direction each list is read and how corner nodes are shared. These values must not be silently treated as five interior nodes if the figure uses five boundary nodes.

The values resemble rounded samples of a sinusoidal profile under one possible coordinate convention, but resemblance does not establish the intended continuous boundary function. If a boundary is reconstructed by a sine or polynomial, the reconstruction is a modelling choice. It can differ between nodes and change a refined-grid problem. Label any such choice explicitly.

The current PDE grammar accepts expressions on sides rather than a direct arbitrary boundary-array import. For a discrete comparison from a figure, establish the coordinate map and compatible boundary expression, or formulate the finite-difference interior equations as a separate algebraic model with the given samples. Those are different levels of modelling and should not be conflated.

The later approximate five-value list11.78,22.66,31.86,39.05,44.54 cannot be asserted as the exact output of an unspecified grid merely from those boundary numbers. Preserve the corrected31.86 and the stated approximate status. A trustworthy screenshot must show which section those values describe and the model that produced them.

20 5 Manufacture a smooth Poisson reference

Let u=sin(pi*x)*sin(pi*y) on the unit square. Differentiating twice gives u_xx+u_yy=-2*pi^2*u, and all four sides are zero. This generates a known source and compatible boundary values without needing an external reference table.

At the centre u=1. The centred discrete operator applied to the sampled sine has an eigenvalue involving sin(pi*h/2)^2 rather than exactly pi^2. Consequently the discrete solution can differ systematically from the continuous sine even if the linear solve residual is tiny. The discrepancy is discretisation error, not a failed iterative solve.

For equal spacing h, the ratio between the discrete sine amplitude and the continuum amplitude is pi^2*h^2/(4*sin(pi*h/2)^2). It approaches one as h tends to zero. This provides an unusually strong independent check: the sampled sine is an eigenvector of the discrete operator, so the expected discrete amplitude can be derived, not merely guessed from a smooth picture.

20 6 Grid refinement and error ratios

For a raw fixed-grid study, set refine=false and run 17, 33 and 65 nodes per direction. Subdivisions are node counts minus one, so spacing halves. For a smooth second-order stencil, error often scales as C*h^2. Coarse error is then approximately 4/3 of the coarse/fine difference, whereas fine error is approximately 1/3. Version 3 automatic refinement assesses the finer solution.

One comparison does not prove second-order behaviour. Use another level to see whether difference ratios approach four. Boundary nonsmoothness, rounded data, off-grid poles and iterative errors can spoil the expected trend. A relative discrete linear tolerance and a continuum error estimate measure different quantities; tighten the iterative solve enough that it does not dominate the mesh study.

Version 3 automatically compares successive finer solutions against an absolute-plus-relative target. Returned coordinates remain those requested, but returned values come from the finest computed grid. accuracy_not_met leaves field_successful=false and may retain computed samples for inspection. The estimates are not certified continuum bounds; see Chapters 26 and 32.

20 7 Corner compatibility and physical interpretation

If left u=50 and top u=0 are both stated as constants, the upper-left corner receives inconsistent values. A continuous Dirichlet formulation must address that conflict. The engine checks agreement within its boundary tolerance; it does not invent an unannounced averaged corner value to make the input acceptable.

In steady heat conduction with constant conductivity and no source, Laplace's equation describes an ideal equilibrium temperature field. The same mathematics can represent other potentials under different assumptions. Units and constitutive assumptions determine what u means physically. A field satisfying the equations does not by itself establish material safety, boundary heat-transfer laws or experimental agreement.

20 8 Practice and worked checks

1. What is the sole interior value of a3 by3 square grid with all boundary values10? It is10; the constant field is exactly harmonic.

2. For hx=hy=h and f=4, write the interior equation. It is u_P=(sum of four neighbours-4*h^2)/4.

3. Why is grid x=33 not32 cells plus33 interior nodes? It counts33 total nodes,32 subdivisions and31 interior nodes on that axis.

4. Can one manual sweep from zeros be compared with a converged solve? Only as an intermediate algorithm state, not as the final field.

5. If all boundary values are zero and f=0, what field is expected? The zero field, with a uniqueness argument for the standard finite Dirichlet problem.

6. Why can the centre differ from 1 in the manufactured sine example? The numerical scheme solves a discrete operator, whose sine eigenvalue differs from the continuum eigenvalue.

7. Does a small discrete residual certify continuum accuracy? No. It checks the finite system; refinement or analytic comparison investigates the discretisation.

8. How should an unexplained five-number reference be treated? As incomplete comparison information until its coordinates, grid and boundary convention are known.

20 9 A finite difference teaching experiment

Construct the one-node example first, then a small four-interior-node system, then the manufactured sine field. For each, predict a value, a symmetry and a residual check before running. Use the heatmap section slider to identify rows by coordinate. Export numeric values for the calculations rather than reading approximations from colours.

20 10 Construct a compatible nonzero boundary field

20 11 Use a bilinear harmonic function

The field u=x*y on the unit square has u_xx=u_yy=0, so it is harmonic. Its boundary expressions are left0,righty,bottom0,topx. All four corners agree. The centred second differences of a bilinear function vanish exactly in exact arithmetic, making it a strong finite-difference indexing check at every grid node.

```
pde
type=poisson
u_xx+u_yy=0
domain x=[0,1]
domain y=[0,1]
boundary left u=0
boundary right u=y
boundary bottom u=0
boundary top u=x
grid x=9
grid y=9
refine=false
```

At(0.5,0.5),u=0.25; at(0.25,0.75),u=0.1875. These values do not depend on interpreting a colour scale. Compare them with the returned rows using coordinates, then select the matching y section in the Desktop. The field should increase along x for positive fixed y.

Changing the domain to a rectangle while preserving u=x*y still gives a harmonic field, but the right/top boundary expressions must be evaluated at the new side coordinates. For example right u=2*y when xmax=2. Typing right u=y after that domain change would prescribe another problem, even though the formula resembles the old one.

20 12 A discrete maximum principle as a diagnostic

For the standard five-point Laplace system with prescribed boundary values and positive neighbour weights, an interior value is a convex average of neighbours. A strict interior maximum above every boundary value would propagate through equal neighbours and contradict connected boundary data. Thus the discrete solution stays within the boundary range for this setting.

Use this as a check on a harmonic field, not as a theorem for every forced PDE or every numerical scheme. Poisson forcing can create interior extrema outside the boundary range. Heat discretisation with rough data has different positivity considerations. State the operator and assumptions whenever applying a qualitative principle.

20 13 Compare a section rather than an unexplained row number

The exported field stores x fastest, then y. A table row number can change when node counts change, whereas a coordinate such as y=0.5 has a clear meaning when present on both grids. Compare common physical coordinates and record the section label in screenshots. This avoids attributing a mesh-order mismatch to a numerical error.

20 14 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

20 15 1. Poisson and Laplace on a rectangle

```
pde
type = poisson
u_xx + u_yy = -2*pi^2*sin(pi*x)*sin(pi*y)
domain x = [0, 1]
domain y = [0, 1]
boundary left u = 0
boundary right u = 0
boundary bottom u = 0
boundary top u = 0
grid x = 33
grid y = 33
refine=false
```

Exact solution: sin(pi*x)*sin(pi*y). Use a zero right side for Laplace. The unknown is named u; coordinates must be x,y. Supported operator:

```
a*(u_xx + u_yy) = f(x,y), a constant and nonzero.
```

Equivalent affine rearrangements and constant scaling are accepted. The isotropic second-derivative coefficient must be a nonzero constant. Mixed partials, anisotropy, reaction terms and nonlinear steady PDEs are outside this class. Sources and boundaries may use x,y and constants. Dirichlet, outward-normal Neumann and supported nonnegative Robin conditions are now available. Left/right denote minimum/maximum x; bottom/top denote minimum/maximum y. Dirichlet corner data must agree.

The current implementation uses conservative rectangular finite volumes, natural normal/Robin boundary contributions and sparse preconditioned conjugate gradients. Away from boundaries its centred Laplacian reduces to the familiar five-point balance. No dense spatial PDE matrix is required.

grid x and grid y count nodes including endpoints. The Version 3 defaults are 21 per spatial direction for steady fields. Requested output nodes and internal refinement budgets are distinct. The spacing on a requested axis is domain length divided by node count minus one.

20 16 Controls, refinement and numerical meaning

```
refine = true
linear tolerance = 1e-10
boundary tolerance = 1e-9
max iterations = 10000
```

refine=true is the default. Version 3 computes successively finer internal solutions and checks estimated error against absolute tolerance+relative tolerance*abs(value) at returned nodes/components. Requested output coordinates remain fixed. PDE/eigen default targets are each 1e-3; integral-equation defaults are each 1e-9. Actual internal work appears in the convergence table.

For second-order PDE convergence, a finer-grid difference divided by 3 estimates the finer error; first-order upwind spatial convergence uses divisor 1. Integral equations instead compare fourth-order Richardson extrapolants with divisor 15. These estimates assume the expected smooth convergence regime and can fail near singularities, layers or roundoff.

linear tolerance controls the relevant discrete linear residual, not continuum error. Defaults are 1e-11 in double and 1e-25 in binary128. max iterations limits iterative work separately on each grid. Field boundary tolerance defaults to 1e-8 for compatibility checks. ODE BVP boundary tolerance has separate absolute-residual semantics. Discrete boundary residuals are not independent continuous-error certificates.

maximum_discrete_residual is an absolute residual of the finite-difference or quadrature equations (heat/wave are normalised to their isolated time derivative). It does not estimate the error in the original continuous equation. Sources/kernels are sampled; their smoothness is not proved. Numerical success means the selected discrete computation finished, not that a specified number of digits or all mathematical solutions have been established.

With refine=false, the requested fixed grid is solved and accuracy_target_met is null. Automatic target failure reports accuracy_not_met with field_successful=false; computed samples may remain available for inspection. Other failures may have no field. All refinement levels share the numerical time and cancellation budget.

Requested field output is capped at 10000 nodes. Defaults for internal max grid are PDE 801, eigen 1025, Fredholm 513 and Volterra 4097; default max nodes is 250000, allowed up to 500000. Requested PDE spatial axes permit 3..401 nodes, with at least 5 for time PDE x; time counts permit 2..10001. Integral output permits 3..513. Syntax validation does not establish numerical convergence or compatibility.

20 17 Desktop, exports and DLL

The bundled V3_ files demonstrate the new numerical classes. Open them directly if no individual menu item exists. Field maps now include contours and transient playback; convergence/eigenpairs/sweeps and Parameter sweep panels provide comparisons and CSV/PNG export. Display plots use limited precision while numeric exports retain engine strings.

PDE output row order is x fastest, then y or t. JSON uses schema equation_solver.field_result.v1, problem_type pde or integral_equation, grid_shape=[nx,ny] (integral ny=1), variables, trajectory[].values, empty solutions, field_successful, method, field_type, status, reason, precision, iterations, residuals, refinement fields and optional wave courant_number. global_error_certified and search_complete are false.

CLI/C ABI status 0 indicates accepted numerical computation or a syntax check; input/configuration and numerical-failure statuses must be distinguished. Read status and the problem-specific success/accuracy flags. accuracy_not_met can retain field samples; a quadrature candidate remains unaccepted when value is null. C ABI v1 ownership is unchanged: free each handle once with eq_result_free.

20 18 Deliberate boundaries of this supported mode

Version 3 adds Neumann/Robin PDE conditions, eigenproblems and linear transport while retaining a bounded scope. Arbitrary geometry, periodic boundaries, nonlinear/coupled PDE systems, multidimensional transient PDEs, moving boundaries and first-kind or singular-kernel integral equations are excluded. Standalone quadrature has its own wider endpoint/infinity support; it does not extend integral-equation kernel scope.

20 19 Applying the method in the Desktop

20 20 Poisson and Laplace on a rectangle

```
pde
type=poisson
u_xx+u_yy=-2*pi^2*sin(pi*x)*sin(pi*y)
domain x=[0,1]
domain y=[0,1]
boundary left u=0
boundary right u=0
boundary bottom u=0
boundary top u=0
grid x=33
grid y=33
refine=false
```

This manufactured example has u=sin(pi*x)*sin(pi*y). For Laplace, set the right-hand side to zero and enter the required boundaries. Use u as the unknown and x,y as coordinates. The operator is a*(u_xx+u_yy)=f(x,y) with a constant nonzero a. Prescribed-value boundaries may use coordinates and constants; the values at each shared corner must agree.

Left/right are the minimum/maximum x sides and bottom/top the minimum/maximum y sides. grid x and grid y count nodes including endpoints. On a unit interval 33 nodes give spacing 1/32. Supplied boundary values must correspond to these coordinates; a short list of measured values is not automatically a boundary function or an accepted interpolation directive. If converting a figure, establish coordinates, point order, corner values and a justified boundary expression first.

The steady solver uses conservative rectangular finite volumes and sparse preconditioned CG. The class remains isotropic and linear on a rectangle. Dirichlet, normal and supported Robin data are accepted; mixed derivatives, arbitrary geometry, nonlinearities and periodic conditions are outside scope. Chapter 26 develops the outward-normal signs.

20 Poisson jobs inside a workflow

```
workflow
pde poisson_square
    type = poisson
    u_xx+u_yy = -2*pi^2*sin(pi*x)*sin(pi*y)
    domain x = [0,1]
    domain y = [0,1]
    boundary left u = 0
    boundary right u = 0
    boundary bottom u = 0
    boundary top u = 0
    grid x = 33
    grid y = 33
end
report poisson_square.status
```

Expected status solution_computed. The centre value u(0.5,0.5) is near 1 on this grid.

Use @n for both grid directives and repeat with increasing n to compare the field near the centre. Retain the full field from each accepted job.

20 Related field studies

The same workflow pde block structure applies to supported time-dependent and mixed-boundary models in Chapters 21, 26 and 28. Retain their service-specific boundary, grid and time directives.

Chapter 21 Heat and Wave Equations in One Space Dimension

21 1 Time evolution adds a second numerical resolution

A heat or wave calculation has a spatial mesh and a time mesh. Refining only one can leave the other as the dominant error. The Desktop's grid x counts spatial nodes and grid t counts time levels, including both endpoints. On [0,1] with 41 spatial nodes, dx=1/40; on [0,1] with 101 time levels, dt=1/100.

For heat conduction u_t=a*u_xx, a has units length squared per time. For the wave equation u_tt=c^2*u_xx, c has units length per time. The same numerical coefficient1 in a nondimensional model has a different meaning in the two equations. Do not compare time steps across them without identifying scales.

21 2 Derive the heat equation reference

On the unit interval with zero endpoint values, u(x,t)=exp(-pi^2*t)*sin(pi*x) satisfies u_t=u_xx. At t=0 its profile is sin(pi*x); both endpoints vanish at every time. The amplitude decays while the spatial shape remains unchanged. At x=0.5,t=0.1, the value is exp(-pi^2/10), approximately 0.3727078389.

This reference tests every part of the model: initial profile, endpoint conditions, diffusion sign, spatial coordinate and time coordinate. A negative diffusivity would produce growth and falls outside the positive-diffusivity scope. A wrong time interval can still give a smooth decaying plot but a quantitatively different endpoint amplitude.

The centred semidiscrete sine has decay rate4*sin(pi*dx/2)^2/dx^2. Crank-Nicolson then updates its amplitude by a rational factor involving dt and that discrete rate. This separates spatial eigenvalue error from temporal propagation error. For carefully chosen smooth data, an independent discrete reference can be derived before taking the continuum limit.

21 3 Why implicit heat stepping still needs resolution

Crank-Nicolson averages the spatial derivative contribution at the old and new time levels. It solves a tridiagonal system at each step. Its favourable stability properties do not guarantee positivity or absence of oscillation for rough data, dominant advection or very large time increments.

A sharp initial profile contains rapidly decaying high-frequency modes. A stable but poorly resolved step can retain oscillatory numerical components rather than representing the physical smoothing accurately. Refining time and space and checking extrema can reveal this. Binary128 reduces roundoff but cannot replace those resolution studies.

The supported heat form can include variable diffusion a(x,t), advection b(x,t), reaction c(x,t) and source f(x,t), all linear in u. The product a(x)*u_xx is not automatically interpreted as the conservative derivative (a*u_x)_x; the latter adds a derivative-of-coefficient term. Use the equation that matches the physical balance actually intended.

21 4 Wave motion needs both position and velocity

The wave model u_tt=u_xx with initial sin(pi*x), initial velocity zero and zero endpoint displacements has solution cos(pi*t)*sin(pi*x). At t=0.5 the continuum profile is zero, though the velocity is not. At t=1 the displacement profile is the negative of its initial profile. Heat decay and wave sign reversal should not be interpreted through the same intuition.

Specifying only initial displacement leaves a family of wave solutions. For example a sin(pi*t)*sin(pi*x) contribution vanishes initially but changes initial velocity. This is why the second-order time equation requires both initial u and initial u_t. Boundary time derivatives must also be compatible with initial velocity at the corners.

A travelling-wave or d'Alembert construction can provide other references, but endpoint reflections must be incorporated. A free infinite-string formula is not automatically the solution of a finite string with fixed ends. State the domain and boundary mechanism as carefully as the PDE itself.

21 5 Derive the Courant condition's practical meaning

For the explicit centred wave scheme, the Courant number is c*dt/dx. The implemented stability check requires this to be at most one. With c=1,dx=0.025,dt=0.01 the ratio is0.4. Doubling spatial nodes from 41 to81 halves dx and doubles the ratio to0.8. Refining once more without changing dt raises it to1.6 and violates the check.

Increasing grid t reduces dt because more time levels fit the same interval. Shortening the physical interval can also change dt for a fixed count, but it changes the requested problem. Prefer a mesh change that preserves the intended interval when testing numerical convergence.

The Courant restriction is a stability property of this particular scheme and constant-speed model. Meeting it does not certify a small phase error or continuum accuracy. A stable coarse solution can have the wrong propagation speed numerically, causing increasing phase mismatch over many cycles.

21 6 Boundary compatibility is part of the model

For heat, initial u must agree with the endpoint boundary values at the first time. For wave, initial velocity must also agree with the time derivative of the displacement boundary functions. A constant zero endpoint displacement has zero time derivative; an initial velocity that is nonzero there conflicts with it.

The engine's necessary corner checks do not prove every higher-order smoothness condition. Rough but finite functions can still yield reduced convergence or physical irregularities. If a problem includes a deliberate sudden boundary change, state its interpretation and determine whether the current smooth finite-input method is appropriate rather than loosening compatibility tolerance until it passes.

21 7 Read a space time field correctly

A transient heatmap has t as its second coordinate. Select a returned time for the spatial section u(x,t). Version 3 also provides Animate/Pause playback of computed frames. Neither slider motion nor playback performs another integration or inserts new numerical time steps.

The colour scale is computed from the returned field. Two runs can use different extrema and therefore similar colours can represent different numeric values. Compare labels or exported samples for quantitative work. Plotting uses double precision even when engine strings retain binary128 output.

PDE rows have x varying fastest, then time. If exporting to another tool, use the coordinate columns and grid shape rather than assuming arbitrary row sorting preserves mesh connectivity. A transposed reconstruction can look plausible while exchanging space and time.

21 8 Accuracy experiments and limits

For heat, compare spatial refinement with temporal refinement separately, then refine both. For wave, keep a safe Courant ratio while refining and compare both amplitude and phase against the exact sine mode. Use maximum errors across common nodes, not only the final centre value; at some times a chosen sample can vanish by symmetry and hide a substantial error elsewhere.

Version 3 automatic refinement doubles active subdivisions repeatedly and assesses estimated solution error at returned nodes. A target is accepted only when the method criterion is met. For independent raw-order experiments use refine=false and compare successive requested meshes. Acceptance remains conditional numerical evidence, not a certified global bound.

No nonlinear PDEs, variable-speed wave systems, arbitrary boundaries, finite elements or moving geometries are implied by these examples. A successful basic vibrating-string calculation should motivate a carefully scoped extension, not a claim that the software solves general structural dynamics.

21 9 Exercises with answers

1. Compute dx for grid x=21 on [0,2]. It is2/(21-1)=0.1.

2. Compute dt for grid t=101 on [0,2]. It is0.02.

3. With c=3 and those spacings, is the Courant test met? The ratio is3*0.02/0.1=0.6, so yes.

4. What happens if dx is halved at the same dt? The ratio becomes1.2 and fails the implemented criterion.

5. What is the continuum heat centre value at time0? It is1. At time0.1 it isexp(-pi^2/10).

6. What is the continuum wave centre displacement at time0.5? Zero for this sine mode. That does not mean the entire dynamic state is stationary.

7. Why are two initial functions needed for wave? The time derivative is second order and the displacement alone does not fix its initial velocity.

8. Does refine=false mean the scheme is exact? No. It means the optional refined comparison is skipped and no corresponding error estimate is reported.

9. Why not judge two fields solely by matching colours? Their scales can differ and colours are a visual encoding, not full-precision numeric evidence.

10. Explain the nonconservative variable-diffusion warning. Expanding(a*u_x)_x gives a*u_xx+a'*u_x, which differs from a*u_xx when a varies spatially.

21 10 A classroom experiment

Use the same sine profile for heat and wave. Predict monotone amplitude decay versus periodic sign changes, then compare numerical fields. Record mesh counts, dx,dt,Courant number where applicable, selected section coordinate and analytic errors. This gives students a concrete distinction between a differential law, a numerical scheme and a display.

21 11 Add a nonzero source without losing an exact reference

21 12 Manufacture a stationary heat field

Take u=x*(1-x), independent of time. Its second spatial derivative is-2 and time derivative is0. Therefore it solves u_t=u_xx+2 with zero endpoint values and initial u=x*(1-x). The source balances diffusion exactly and the continuum field remains constant in time.

```
pde
type=heat
u_t=u_xx+2
domain x=[0,1]
interval t=[0,0.1]
boundary left u=0
boundary right u=0
initial u=x*(1-x)
grid x=21
grid t=21
refine=false
```

Centred differences reproduce the second derivative of this quadratic exactly in exact arithmetic. The manufactured stationary case therefore provides a clean source-sign and initial-data test. If the source is entered as-2, the state

should not remain stationary; a smooth heatmap alone would not expose the wrong physical balance without a reference.

21 13 A nonzero wave forcing example

Let u=x*(1-x)*cos(t). Then u_tt=-x*(1-x)*cos(t) and u_xx=-2*cos(t). The forcing in u_tt=u_xx+f is f=(2-x*(1-x))*cos(t). Initial displacement is x*(1-x),initial velocity0, and both endpoint displacements remain zero. This gives a manufactured forced wave compatible with the current constant-speed scope.

A chosen function can thus generate source, initial and boundary data together. The method is valuable because every data term is linked by derivation. It is not enough to select a visually plausible forcing independently and assume a previous exact solution remains valid. Differentiate the proposed reference explicitly.

21 14 Source and reaction terms have different meanings

A forcing f(x,t) is independent of u. A reaction c(x,t)*u changes with the unknown. They can coincide for one known reference but represent different equations for perturbations. This affects stability, sensitivity and physical interpretation. Keep the full entered operator with any reported solution so those distinctions remain reviewable.

21 15 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

21 16 2. Heat, advection–diffusion and reaction–diffusion in 1D

```
pde
type = heat
u_t = u_xx
domain x = [0, 1]
interval t = [0, 0.1]
boundary left u = 0
boundary right u = 0
initial u = sin(pi*x)
grid x = 41
grid t = 101
refine=false
```

Exact solution: exp(-pi^2*t)*sin(pi*x). Supported equation:

```
u_t = a(x,t)*u_xx + b(x,t)*u_x + c(x,t)*u + f(x,t).
```

The leading u_t coefficient must be a nonzero constant; all state terms are linear, and diffusion is strictly positive at evaluated interior nodes. Initial data use x at the first time endpoint. Dirichlet, outward-normal Neumann and supported Robin conditions may use x,t and must satisfy compatibility. Periodic conditions remain excluded. a(x)*u_xx is not automatically the conservation-form derivative (a(x)*u_x)_x.

The method is Crank–Nicolson in time with centred spatial differences. Each time step solves a tridiagonal system. The method is second order for sufficiently smooth solutions. Implicit time stepping does not guarantee physical positivity, absence of oscillations, or adequate resolution. Stiff reactions, rough initial data and dominant advection can need much finer meshes; no automatic upwind or positivity-preserving discretisation is claimed. A failed tridiagonal pivot is reported, without silently changing the model.

grid t counts time levels, including both interval endpoints. Default counts are 41 spatial nodes and 201 time levels. Both spatial and temporal grids matter.

21 17 3. Wave equation in 1D

```
pde
type = wave
u_tt = u_xx
domain x = [0, 1]
interval t = [0, 1]
boundary left u = 0
boundary right u = 0
initial u = sin(pi*x)
initial u_t = 0
grid x = 41
grid t = 101
refine=false
```

Exact solution: cos(pi*t)*sin(pi*x). Supported equation:

```
u_tt = c^2*u_xx + f(x,t), c^2 a positive constant.
```

The u_tt coefficient is a nonzero constant and c^2 is a positive constant. Damping, reaction, advection, variable speed and coupled waves are excluded. Both displacement and velocity are required. Dirichlet, normal or supported homogeneous/nonhomogeneous Robin boundary data must meet initial displacement and differentiated boundary compatibility; higher smoothness is not proved by those necessary checks.

The explicit centred scheme uses a second-order initial Taylor step and obeys

```
c*dt/dx <= 1.
```

An unstable grid returns unstable_grid before time stepping. Increase grid t to reduce dt; increasing only grid x can make this condition harder to satisfy. The Courant number is included in results. This stability check applies to this constant-speed wave scheme; it is not an error bound or a generic PDE theorem.

21 18 Controls, refinement and numerical meaning

```
refine = true
linear tolerance = 1e-10
boundary tolerance = 1e-9
max iterations = 10000
```

refine=true is the default. Version 3 computes successively finer internal solutions and checks estimated error against absolute tolerance+relative tolerance*abs(value) at returned nodes/components. Requested output coordinates remain fixed. PDE/eigen default targets are each 1e-3; integral-equation defaults are each 1e-9. Actual internal work appears in the convergence table.

For second-order PDE convergence, a finer-grid difference divided by 3 estimates the finer error; first-order upwind spatial convergence uses divisor 1. Integral equations instead compare fourth-order Richardson extrapolants with divisor 15. These estimates assume the expected smooth convergence regime and can fail near singularities, layers or roundoff.

linear tolerance controls the relevant discrete linear residual, not continuum error. Defaults are 1e-11 in double and 1e-25 in binary128. max iterations limits iterative work separately on each grid. Field boundary tolerance defaults to 1e-8 for compatibility checks. ODE BVP boundary tolerance has separate absolute-residual semantics. Discrete boundary residuals are not independent continuous-error certificates.

maximum_discrete_residual is an absolute residual of the finite-difference or quadrature equations (heat/wave are normalised to their isolated time derivative). It does not estimate the error in the original continuous equation. Sources/kernels are sampled; their smoothness is not proved. Numerical success means the selected discrete computation finished, not that a specified number of digits or all mathematical solutions have been established.

With refine=false, the requested fixed grid is solved and accuracy_target_met is null. Automatic target failure reports accuracy_not_met with field_successful=false; computed samples may remain available for inspection. Other failures may have no field. All refinement levels share the numerical time and cancellation budget.

Requested field output is capped at 10000 nodes. Defaults for internal max grid are PDE 801, eigen 1025, Fredholm 513 and Volterra 4097; default max nodes is 250000, allowed up to 500000. Requested PDE spatial axes permit 3..401 nodes, with at least 5 for time PDE x; time counts permit 2..10001. Integral output permits 3..513. Syntax validation does not establish numerical convergence or compatibility.

21 19 Applying the method in the Desktop

in one space dimension

```
pde
type=heat
u_t=u_xx
domain x=[0,1]
interval t=[0,0.1]
boundary left u=0
boundary right u=0
initial u=sin(pi*x)
grid x=41
grid t=101
refine=false
```

The reference solution is exp(-pi^2*t)*sin(pi*x). The supported linear form can include variable diffusion, advection, reaction and forcing. Diffusivity must be positive at sampled interior nodes. Initial data and prescribed boundary values must agree at the initial corners. The Crank-Nicolson scheme uses centred spatial differences and tridiagonal solves. It does not automatically preserve positivity or avoid oscillation on an inadequate grid.

21 20 Wave motion in one space dimension

```
pde
type=wave
u_tt=u_xx
domain x=[0,1]
interval t=[0,1]
boundary left u=0
boundary right u=0
initial u=sin(pi*x)
initial u_t=0
grid x=41
grid t=101
refine=false
```

The reference solution is cos(pi*t)*sin(pi*x). Supported speed is constant and positive. Supply both initial position and velocity. For wave speed c, the explicit method requires c*dt/dx<=1. An unstable_grid result means increase time resolution, for example by increasing grid t. Increasing spatial resolution alone can worsen this stability ratio.

Time-dependent boundaries must meet initial position and velocity corner checks. These are necessary checks, not a proof of all smoothness requirements. Damping, variable speed and coupled wave systems are not added by this mode.

21 21 Read the heatmap and sections

Open PDE fields / animation to inspect numerical, exact or signed-error fields, contours and spatial sections. x is horizontal; the second coordinate is y for steady fields or t for transient data. Animate/Pause displays computed transient frames. Use PNG export and numeric CSV/JSON alongside the visual display.

Move the slider to select a y or t level. The dashed line identifies the selected map level and the curve below shows u(x) at that level. The label gives the selected coordinate. Values and colours use double precision; consult numeric exports for retained engine strings. A blank map after failure is not evidence that u is zero.

21 22 Refinement controls and meaning

By default, refine=true computes successively finer internal grids until the estimated absolute-plus-relative target is met or a work/grid limit intervenes. The requested table coordinates remain unchanged, with finest-grid values sampled there. The default PDE absolute and relative targets are each 1e-3. Fixed-grid runs have accuracy_target_met=null.

Use refine=false to skip the comparison; no refinement error estimate is then claimed. linear tolerance controls the discrete Poisson residual, not the continuum solution error. boundary tolerance controls corner compatibility checks. max iterations limits the Poisson iterative solve. Heat and wave use their own direct/marching operations.

Returned numerical fields are capped at 10000 nodes. max grid defaults to 801 for PDEs and max nodes to 250000 internal nodes, with an allowed internal maximum of 500000. Requested spatial axes permit up to 401 nodes and time axes up to 10001, subject to the output cap. A target failure can retain computed samples, but they remain unaccepted accuracy evidence.

21 Compare time dependent PDE runs

Use one pde job per grid and time-step combination. Substitute a numeric workflow variable in a grid or time directive, and compare accepted field snapshots at matching physical times. Keep the heat or wave model directives from this chapter inside the job.

Chapter 22 Fredholm and Volterra Integral Equations

22 1 A kernel couples values across an interval

A second-kind integral equation has the form y=f+lambda*K[y]. The forcing f is supplied and K is an integral operator defined by a kernel. In a Fredholm equation the integration interval is fixed; in a Volterra equation the upper endpoint is the current coordinate x. Both can describe accumulated influence, but their discrete systems have different structures.

The term second kind means that the unknown appears outside the integral as well as within it. First-kind inversion removes that outside term and can have very different conditioning. The current solver supports selected linear second-kind problems with finite ordinary kernels; it does not provide a general inverse-problem framework.

22 2 Manufacture a Fredholm solution by integrating it first

Choose kernel K(x,s)=x*s on [0,1], lambda=1 and desired y=1+x. Then the integral is x*integral from 0 to1 of s*(1+s) ds=x*(1/2+1/3)=5*x/6. To make the equation hold, set forcing y-integral=1+x/6. This derivation produces the built-in model without relying on an unexplained reference curve.

```
integral
type=fredholm
interval x=[0,1]
forcing=1+x/6
kernel=x*s
lambda=1
grid x=65
refine=false
```

At x=0 the kernel vanishes and y(0)=1 directly. At x=1 the analytic value is2. The discrete trapezoidal quadrature does not integrate every quadratic exactly, so a successful discrete result can differ slightly from the manufactured continuum function. That discrepancy is expected and should shrink under refinement.

22 3 Reduce the separable kernel to one scalar quantity

Let M=integral from 0 to1 of s*y(s) ds. Then y(x)=1+x/6+x*M. Multiplying by x and integrating gives M=1/2+1/18+M/3, hence M=5/6. Substitution recovers y=1+x. This independent reduction is especially useful because it explains the operator's finite-rank structure.

For a general forcing f and the same kernel, M=F+lambda*M/3 where F=integral s*f(s) ds. Thus M=F/(1-lambda/3) when lambda is not3. As lambda approaches3, small errors in F or quadrature can be amplified. At lambda=3, compatibility rather than ordinary division determines whether solutions exist or are nonunique.

This gives a mathematical context for singular_system. A singular discrete matrix can reflect a genuine operator issue or a quadrature-specific degeneracy. Its diagnostic alone is not a theorem about the continuous equation. Conversely a nonsingular discrete solve near a singular operator can have a tiny algebraic residual and a large continuum error.

22 4 Derive the Volterra exponential example

The equation y(x)=1+integral from 0 to x of y(s) ds gives y(0)=1. Differentiating for a smooth solution yields y'=y, so y=exp(x). Retaining the original value at zero is essential; differentiating without that condition would leave a free multiplicative constant.

```
integral
type=volterra
interval x=[0,1]
forcing=1
kernel=1
lambda=1
grid x=65
refine=false
```

The changing upper integration limit produces a forward-marching discrete structure. Earlier values contribute to later integrals, while the endpoint term can contain the current unknown. Solving its diagonal relation is different from treating the whole integral as an explicit sum of already known samples.

22 5 Inspect the first trapezoidal step

With spacing h and kernel1, the first step satisfies y1=1+h*(y0+y1)/2. Since y0=1, rearrangement gives y1=(1+h/2)/(1-h/2). At h=0.1 this is1.1052631579, compared with exp(0.1) approximately 1.105170186. The difference is a quadrature approximation, not a failure to solve the scalar step equation.

Repeating yields y_j=((1+h/2)/(1-h/2))^j for this constant-kernel example. This is the same rational amplification factor as the trapezoidal method for y'=y. It supplies an exact reference for the discrete scheme, while exp(x) is the continuum reference. Comparing both identifies where the error enters.

If the marching diagonal1-lambda*h*K(x_j,x_j)/2 becomes zero or numerically unsafe, the discrete step fails. Refining h can avoid that particular diagonal obstruction for a suitable smooth problem, but the correct response depends on the equation. Do not loosen a pivot test until a meaningless huge value is accepted.

22 6 Build another manufactured example

Let the Volterra kernel be x-s, choose lambda=1 and y=1. The integral of x-s from 0 to x is x^2/2, so forcing=1-x^2/2 yields an exact constant continuum solution. The kernel is smooth and vanishes on the diagonal. Its polynomial integrand is linear in s for this chosen y, so trapezoidal quadrature integrates it exactly at mesh nodes in exact arithmetic.

This differs from the exponential example, where quadrature truncation is nonzero. A constant manufactured solution can test indexing, interval limits and weights with little truncation ambiguity, while a nonpolynomial solution tests convergence. A good validation suite includes both, rather than selecting only a case the method happens to integrate exactly.

For Fredholm choose kernel1, desired y=1+x and lambda=1/4. The integral of y is3/2, so forcing=5/8+x. Integrating the equation gives a compatible scalar moment and recovers the same desired solution. Altering lambda or forgetting the interval length changes the required forcing.

22 7 Refinement and smoothness assumptions

The base scalar integral-equation discretisation is composite trapezoidal quadrature, with second-order smooth-grid error. Version 3 uses Richardson extrapolation to cancel that leading error and compares successive extrapolants. Fixed-grid derivations in this chapter explicitly use refine=false; the adaptive returned values need not equal the raw trapezoidal recurrence.

Automatic refinement returns the finer Richardson extrapolant at requested coordinates. Differences between successive extrapolants divided by 15 estimate fourth-order error under smooth asymptotic convergence. Default absolute and relative targets are each 1e-9. Near-singular operators and unresolved structure can invalidate a simple error expansion; the estimate is not a general theorem.

Requested scalar integral grids permit 3..513 nodes. Internal max grid defaults to 513 for Fredholm and 4097 for Volterra. Automatic refinement may require larger internal grids than requested output; the chosen cap can stop target acceptance. There is no current universal 257-requested-node refinement rule.

22 8 Interpret output and failures

Use Solutions for x,y samples and ODE / BVP / integral plot for the curve. Full JSON identifies field_type and field_successful; the algebraic solutions array is empty by design. File > Export results saves the returned table or full result. A small maximum_discrete_residual checks the quadrature system, not the original continuous equation at every unsampled x.

First-kind equations, nonlinear y-dependent kernels, infinite domains, principal values and singular/weakly singular kernels are excluded. Rewriting a weakly singular kernel as an ordinary expression does not make it supported. The finite sampling checks are not a proof that the kernel is smooth across the relevant rectangle or Volterra triangle.

22 9 Practice problems and worked explanations

1. Find forcing for y=1 with Fredholm kernel1,lambda=1/2 on[0,1]. The integral is1, so forcing=1/2.

2. Change that interval to[0,2]. The integral is2, so forcing must be0. Keeping1/2 would specify another equation.

3. Find forcing for y=1+x with kernel x*s,lambda=2. It is1+x-2*(5*x/6)=1-2*x/3.

4. Explain why the Volterra example has y(0)=1. The integral over a zero-length interval vanishes.

5. Compute the first trapezoidal value with h=0.2,kernel1,forcing1. It is1.1/0.9=11/9, approximately 1.2222222222.

6. Why does differentiating a Volterra equation require care? Boundary terms and differentiated kernel terms can appear, and the original endpoint relation must be retained.

7. What does a tiny discrete residual establish? That the returned values solve the discrete quadrature equations to the numerical criterion, not that continuum error is tiny.

8. Why use two independent references for the exponential example? The closed-form discrete recurrence checks implementation, while exp(x) checks continuum convergence.

9. Does starting with 129 nodes fix the final internal grid? No. Doubling once gives 257 nodes, but integral extrapolation uses a preliminary refinement and subsequent estimate-producing refinements. Inspect the actual internal-grid history rather than inferring all work from the output count.

10. Does a finite value at every grid node exclude a singular kernel between nodes? No. Smoothness must be justified over the full relevant domain.

22 10 An integral equation laboratory

Implement the constant manufactured Volterra case, the exponential case and the separable Fredholm case as separate projects. Derive each forcing and reference independently. Compare grids17,33,65,129 where applicable, record discrete residual and continuum sample error, and explain whether second-order ratios emerge. End the report by distinguishing successful discrete solution, convergence evidence and a certified continuous error claim.

22 11 A degenerate Fredholm operator and compatibility

22 12 A constant kernel can be analysed completely

Consider y(x)=f(x)+lambda*integral from 0 to1 of y(s) ds. Let M be that integral and F the integral of f. Integrating the equation gives M=F+lambda*M. For lambda not1, M=F/(1-lambda), and the unique candidate is y=f+lambda*F/(1-lambda).

At lambda=1 the relation becomes0=F. If F is nonzero there is no solution. If F=0, M is free and there is a family y=f+M consistent with the integral relation. This small example separates no solution from nonuniqueness at a singular operator using mathematics rather than interpreting a failed matrix solve as one or the other.

For f=1 and lambda=1/2, F=1,M=2 and y=2. For f=x-1/2 and lambda=1, F=0 and the family y=x-1/2+M is possible. For f=1 and lambda=1, the contradiction0=1 proves no solution. A finite quadrature system can reflect these structures, but its diagnostic must be interpreted with its own numerical tolerances.

22 13 Near singularity amplifies quadrature and data errors

With lambda=0.99, the denominator1-lambda is0.01, so an error in F can be amplified by a factor100 in M. This is a conditioning issue, not necessarily a failure of Gaussian elimination. Tightening arithmetic precision can reduce roundoff, but cannot remove a modelling error in f or an inadequately resolved quadrature.

Use the example to design an accuracy study: choose a smooth f with known integral, vary lambda toward one, refine the grid and compare against the analytic scalar reduction. Report both discrete residual and function error. A tiny residual near a sensitive operator should not be marketed as a guarantee of many correct continuum digits.

22 14 A documented singular case is a useful limit example

If the Desktop returns singular_system, retain the model and explain whether independent analysis establishes inconsistency, freedom or only an unsafe numerical system. The same message need not decide the continuous classification universally. Screenshot the report and mathematical setup rather than inventing a curve for an unsuccessful request.

22 15 Implementation controls and supported scope

The following details specify the implementation controls, supported structures and resource limits for the methods used in this chapter. Read them with the mathematical examples rather than assuming that a successful simple case extends to every related problem.

22 16 4. Fredholm integral equation of the second kind

```
integral
type = fredholm
interval x = [0, 1]
forcing = 1+x/6
kernel = x*s
lambda = 1
grid x = 65
refine=false
```

This represents

```
y(x) = forcing(x) + lambda * integral[a,b] kernel(x,s)*y(s) ds.
```

The unknown is y, integration coordinate is s, and both coordinates use the same interval [a,b]. The example has exact solution y=1+x. forcing may depend on x; kernel may depend on x,s. They cannot depend on y. lambda is an optional constant scalar, default 1. Model constants can be declared with constant k=....

Nyström discretisation with composite trapezoidal quadrature gives a bounded dense linear system, solved with scaled partial-pivoting elimination. A singular or numerically unsafe pivot returns singular_system. This is a diagnostic about the discretisation, not a proof that the continuous integral equation has no solution. Near-singular operators can amplify quadrature errors substantially.

22 17 5. Volterra integral equation of the second kind

```
integral
type = volterra
interval x = [0, 1]
forcing = 1
kernel = 1
lambda = 1
grid x = 65
refine=false
```

This represents

```
y(x) = forcing(x) + lambda * integral[a,x] kernel(x,s)*y(s) ds.
```

Exact solution: exp(x). The method uses composite trapezoidal quadrature with forward marching, including the current unknown through its diagonal kernel term. At x=a the integral is zero, so y(a)=forcing(a). A singular marching diagonal is diagnosed and usually calls for a finer grid.

Both integral-equation classes require finite smooth kernels and finite intervals. First-kind inversion, nonlinear kernels, singular/weakly singular kernels, principal values and unknown fitted parameters are excluded. Fredholm remains scalar; Volterra supports up to eight coupled linear components. Internal defaults differ: Fredholm max grid=513 and Volterra max grid=4097. Finite sampling does not establish global kernel smoothness.

22 18 Controls, refinement and numerical meaning

```
refine = true
linear tolerance = 1e-10
boundary tolerance = 1e-9
max iterations = 10000
```

refine=true is the default. Version 3 computes successively finer internal solutions and checks estimated error against absolute tolerance+relative tolerance*abs(value) at returned nodes/components. Requested output coordinates remain fixed. PDE/eigen default targets are each 1e-3; integral-equation defaults are each 1e-9. Actual internal work appears in the convergence table.

For second-order PDE convergence, a finer-grid difference divided by 3 estimates the finer error; first-order upwind spatial convergence uses divisor 1. Integral equations instead compare fourth-order Richardson extrapolants with divisor 15. These estimates assume the expected smooth convergence regime and can fail near singularities, layers or roundoff.

linear tolerance controls the relevant discrete linear residual, not continuum error. Defaults are 1e-11 in double and 1e-25 in binary128. max iterations limits iterative work separately on each grid. Field boundary tolerance defaults to 1e-8 for compatibility checks. ODE BVP boundary tolerance has separate absolute-residual semantics. Discrete boundary residuals are not independent continuous-error certificates.

maximum_discrete_residual is an absolute residual of the finite-difference or quadrature equations (heat/wave are normalised to their isolated time derivative). It does not estimate the error in the original continuous equation. Sources/kernels are sampled; their smoothness is not proved. Numerical success means the selected discrete computation finished, not that a specified number of digits or all mathematical solutions have been established.

With refine=false, the requested fixed grid is solved and accuracy_target_met is null. Automatic target failure reports accuracy_not_met with field_successful=false; computed samples may remain available for inspection. Other failures may have no field. All refinement levels share the numerical time and cancellation budget.

Requested field output is capped at 10000 nodes. Defaults for internal max grid are PDE 801, eigen 1025, Fredholm 513 and Volterra 4097; default max nodes is 250000, allowed up to 500000. Requested PDE spatial axes permit 3..401 nodes, with at least 5 for time PDE x; time counts permit 2..10001. Integral output permits 3..513. Syntax validation does not establish numerical convergence or compatibility.

22 19 Desktop, exports and DLL

The bundled V3_ files demonstrate the new numerical classes. Open them directly if no individual menu item exists. Field maps now include contours and transient playback; convergence/eigenpairs/sweeps and Parameter sweep panels provide comparisons and CSV/PNG export. Display plots use limited precision while numeric exports retain engine strings.

PDE output row order is x fastest, then y or t. JSON uses schema equation_solver.field_result.v1, problem_type pde or integral_equation, grid_shape=[nx,ny] (integral ny=1), variables, trajectory[].values, empty solutions, field_successful, method, field_type, status, reason, precision, iterations, residuals, refinement fields and optional wave courant_number. global_error_certified and search_complete are false.

CLI/C ABI status 0 indicates accepted numerical computation or a syntax check; input/configuration and numerical-failure statuses must be distinguished. Read status and the problem-specific success/accuracy flags. accuracy_not_met can retain field samples; a quadrature candidate remains unaccepted when value is null. C ABI v1 ownership is unchanged: free each handle once with eq_result_free.

22 20 Applying the method in the Desktop

22 21 Fredholm second kind

```
integral
type=fredholm
interval x=[0,1]
forcing=1+x/6
kernel=x*s
lambda=1
grid x=65
refine=false
```

This means y(x)=forcing(x)+lambda times the integral from a to b of kernel(x,s)*y(s) ds. The example has y=1+x. Use x for the output coordinate, s for the integration coordinate and y for the unknown. The forcing depends on x; the kernel may depend on x and s, but not on y. Lambda is a constant, defaulting to 1.

The method uses Nyström discretisation with composite trapezoidal quadrature and a dense linear solve. A singular_system diagnostic describes the numerical system; it is not a universal proof that the original integral equation has no solution.

22 22 Volterra second kind

```
integral
type=volterra
interval x=[0,1]
forcing=1
kernel=1
lambda=1
grid x=65
refine=false
```

The upper integration endpoint is x instead of b. This example has y=exp(x) and uses forward trapezoidal marching. At the first endpoint the integral is zero, so y(a)=forcing(a). A problematic marching diagonal is diagnosed; a finer grid may help an otherwise suitable model.

22 23 Grids results and exclusions

Both modes use finite intervals and ordinary finite kernels. Automatic refinement returns extrapolated finest estimates at the requested coordinates. Default targets are 1e-9 absolute and relative, with max refinements=6; internal grid defaults are 513 for Fredholm and 4097 for Volterra. Use refine=false to study the raw trapezoidal method.

Use Solutions and the integral curve panel, then export CSV or JSON. Read field_successful and accuracy_target_met separately from the discrete residual. The residual belongs to the unextrapolated internal solve, not a direct residual certificate for extrapolated output. An accuracy_not_met request may retain useful computed samples for diagnosis.

First-kind inversion, nonlinear integrands, infinite intervals, singular/weakly singular kernels, principal values and unknown parameters are not supported. A finite value at quadrature nodes does not exclude an off-grid pole. Choose kernels and forcing that are smooth over the full relevant domain.

22 Integral equation jobs inside a workflow

```
workflow
integral fredholm_constant
    type = fredholm
    interval x = [0,1]
    forcing = 1-x/2
    kernel = x*s
    lambda = 1
    grid x = 65
end
point = at(fredholm_constant.result.trajectory, 1)
report at(point.values, 2)
```

Expected u(x) = 1.

Repeat the job at several grid sizes to check convergence. This job solves for a function. Chapter 31 shows how to batch definite integrals.

Chapter 23 Extended Precision and Numerical Reliability

Extended precision is useful when rounding limits the refinement or verification of an equation solution. It can preserve information that ordinary double arithmetic loses and can reduce residuals substantially. A reliable numerical result still requires the correct equations, admissible domains, suitable scaling and an interpretation of conditioning. A long decimal output is the start of that assessment, rather than its conclusion.

This chapter follows the worked-example format of the preceding chapters. Ten examples examine representation, cancellation, small exponential changes, repeated roots, sensitive equations, mismatched coordinate scales, transcendental domains and solution counting. The examples distinguish independent reference calculations from historical developer-reported solver results.

Run each solver listing separately. Numerical reference values below were calculated independently; they are not new Patodia execution results. The installed executable was not run while preparing this chapter. Screenshot locations are reserved for actual runs. Use the equations to identify each benchmark. A benchmark name alone cannot establish its discretization, physical interpretation or certified difficulty.

23 1 Separate precision accuracy residual and completeness

Binary64 has 53 significant binary bits and binary128 has 113 [1]. This corresponds to roughly 16 and 34 decimal digits of significand capacity; conservative descriptions often quote about 15 to 16 useful decimal digits for double and about 33 for binary128. These are arithmetic capacities. They do not guarantee that each computed coordinate has that many correct digits.

Spacing depends on magnitude. Near 1 the next larger binary64 value is $1 + 2^{-52}$, giving a gap of about $2.220446049250313 \times 10^{-16}$. For round-to-nearest arithmetic the usual unit roundoff is half that gap, 2^{-53}. Binary128 has corresponding values 2^{-112} and 2^{-113}. Neither is a universal lower bound on residuals: an exactly representable root may evaluate to zero, while cancellation, scale and function evaluation affect other residuals.

Question	Evidence required
Was the equation satisfied?	Evaluate each original residual at the exported coordinates
How accurate are the coordinates?	Reference solution, conditioning or a validated error bound
Does the model fit reality?	Data, assumptions and uncertainty analysis
Were all admissible roots found?	Domain-specific algebra or a suitable validated completeness argument

Binary128 is a fixed precision format. Printing 36 or 37 significant digits can help preserve its stored value, but does not make it an arbitrary-precision calculation or establish 37 accurate digits. Export enough digits for rechecking and distinguish those digits from a demonstrated coordinate error bound.

23 2 Example 1 Establish a reference with the square root of two

Find the positive root of $x^2 = 2$. This gives a simple reference with a known sign restriction and a nonsingular derivative. The listing uses the same domain in both precision modes.

```
var x
x^2=2
bound x=[1,2]
```

Quantity	Reference
Positive root	1.41421356237309504880168872420969808
Derivative at the root	2√2 ≈ 2.8284271247461901

The positive root is unique because x^2 is strictly increasing on [1,2]. If xhat is an approximation and $r = xhat^2 - 2$, then $|xhat - \sqrt{2}| = |r|/(xhat + \sqrt{2})$. The residual therefore has a direct coordinate-error interpretation on this interval.

For example, xhat = 1.41421356 has residual $-6.7121264 \times 10^{-9}$ and error about $-2.37309505 \times 10^{-9}$. The relation above accounts for their different magnitudes. A 10^{-24} residual here would imply a coordinate error of roughly 3.54×10^{-25} near the root, subject to reliable residual evaluation.

This favorable behavior follows from the local derivative and domain. It should not be transferred to a repeated or nearly singular root. Use this example to check output export, precision selection and the independent checker before moving to harder systems.

23 3 Example 2 Avoid cancellation in a small difference

Calculate $y = \sqrt{1 + t} - 1$ for the exact mathematical input $t = 10^{-20}$. A scaled unknown $z = y/10^{-20}$ makes the expected coordinate near one. Compare the following direct formulation with the rationalized one in separate runs.

```
var z
z=(sqrt(1+1e-20)-1)/1e-20
bound z=[0,1]
```

Form	High precision reference
$z = (\sqrt{1+t} - 1)/t$	0.4999999999999999999875 approximately
$y = t\,z$	$4.999999999999999999875 \times 10^{-21}$ approximately

In ordinary binary64 evaluation, $1 + 10^{-20}$ rounds to 1, so the direct expression can become zero before the division. This is information loss in evaluation, even though the mathematical answer is near 0.5. The effect is not fixed by merely displaying more digits after the calculation.

Multiply numerator and denominator by $\sqrt{1+t} + 1$ to get $z = 1/[\sqrt{1+t} + 1]$. This avoids the subtraction of nearly equal numbers. At t = 0 the original quotient is undefined, while the rationalized expression extends it continuously with value 1/2. Here t is nonzero, so the two expressions are exactly equivalent.

Use this stable comparison input:

```
var z
z=1/(sqrt(1+1e-20)+1)
bound z=[0,1]
```

The unscaled difference $\sqrt{1+t} - 1$ also equals $t/[\sqrt{1+t} + 1]$. Extended precision can preserve the tiny change in the direct form, but the stable form remains preferable when available. An implementation may simplify expressions internally; record observed behavior rather than assuming every evaluator follows the same operation order.

23 4 Example 3 Evaluate small exponential and logarithmic changes

The mathematical definitions are expm1(t) = exp(t) − 1 and log1p(t) = ln(1+t). Specialized implementations avoid the loss caused by forming a value very close to 1 and then subtracting, or by rounding 1+t before taking its logarithm [2].

```
var u,v
u=(exp(1e-20)-1)/1e-20
v=log(1+1e-20)/1e-20
bound u=[0,2]
bound v=[0,2]
```

Scaled quantity	Reference expansion at $t = 10^{-20}$
expm1(t)/t	$1 + 5 \times 10^{-21} + O(t^2)$
log1p(t)/t	$1 - 5 \times 10^{-21} + O(t^2)$

The series are expm1(t)/t = 1 + t/2 + t²/6 + … and log1p(t)/t = 1 − t/2 + t²/3 − … . The scaled results are close to 1, even though naive binary64 evaluation of the displayed numerators can yield zero. The sign of the first correction also provides a useful check.

If the installed build lists expm1 and log1p as supported functions, replace the right sides by expm1(1e-20)/1e-20 and log1p(1e-20)/1e-20 in a separate run. The Python checker later in this chapter uses math.expm1 and math.log1p independently of solver support.

The real domain of log1p(t) is t > −1. Better cancellation behavior does not remove that restriction. These functions improve the evaluation of the stated expressions; they do not recover uncertainty in the input t or make every subsequent subtraction accurate.

23 5 Example 4 Relate a repeated root residual to coordinate error

Find the real root of (x − 1)² = 0 on [0,2]. Compare its residual behavior with Example 1. There is one distinct root, of multiplicity two.

```
var x
(x-1)^2=0
bound x=[0,2]
```

Candidate	Residual	Absolute coordinate error
$1 + 10^{-12}$	10^{-24}	10^{-12}
$1 + 10^{-15}$	10^{-30}	10^{-15}

Writing xhat = 1 + e gives residual r = e² exactly in real arithmetic. A residual of 10^{-24} therefore corresponds to 10^{-12} coordinate error. It does not establish 24 accurate decimal places in x. The derivative 2(x − 1) vanishes at the root, so the simple-root linear estimate is unavailable.

The usual Newton update away from the root is e_new = e/2, showing linear rather than quadratic error reduction for this repeated-root equation. This is an explanation of mathematical sensitivity and local iteration behavior; it does not assert which iteration Patodia uses internally.

An exactly representable x = 1 can yield a zero residual even in double arithmetic. Conversely, extended arithmetic can yield a much smaller nonzero residual without resolving the conditioning issue. If the equation becomes $(x - 1)^2 = \delta$, positive δ creates two roots, zero δ gives one distinct root and negative δ gives no real root.

23 6 Example 5 Expose sensitivity in nearly dependent equations

Consider a linear system with a small difference between its two equations. Its exact decimal constants define the mathematical test; they are not measured data. The root is x = y = 1.

```
var x,y
x+y=2
x+1.000000000001*y=2.000000000001
bound x=[-10,10]
bound y=[-10,10]
```

Quantity	Value or relation
Coefficient difference ε	10^{-12}
Determinant	ε
If the second right side changes by δ	Δy = δ/ε and Δx = −δ/ε

Subtract the first equation from the second to obtain $10^{-12}y = 10^{-12}$. This gives y = 1 and x = 1 uniquely. A perturbation $\delta = 10^{-16}$ in the second right side changes y by 10^{-4}. The model is sensitive because the equations nearly repeat the same information.

Binary conversion and arithmetic errors can therefore be amplified. Extended precision may reduce their numerical contribution, but it cannot remove a real uncertainty in the right sides. If the constants are measured less accurately than the small difference, many output digits would be unjustified.

A row-scaled form multiplies each residual by a nonzero factor and leaves exact roots unchanged. It changes the numerical residual size and the meaning of an absolute acceptance threshold. A sensible comparison records both original residuals and a declared dimensionless scaling, rather than describing the smallest raw residual as automatically the best answer.

23 7 Example 6 Refine a system with mismatched coordinate scales

The following example uses a Powell badly scaled system. One positive coordinate is near 10^{-5} and the other near 9.1. The equations define the benchmark used here.

```
var x,y
10000*x*y=1
exp(-x)+exp(-y)=1.0001
bound x=[0.000001,10]
bound y=[0.000001,10]
```

x	y
0.000010981593296998175	9.1061467398665240
9.1061467398665240	0.000010981593296998175

The first equation gives $xy = 10^{-4}$. Substituting $y = 10^{-4}/x$ reduces the check to $\exp(-x) + \exp(-10^{-4}/x) - 1.0001 = 0$. The listed pair and its exchanged pair were calculated independently using high precision arithmetic. Symmetry guarantees the exchanged root whenever the first is a root.

The two table rows are validated reference candidates. Symmetry and numerical refinement alone are not a proof that the box contains exactly two roots. A completeness claim would require analysis of the reduced scalar function or validated isolation over its entire domain.

For a selected small-x branch, $x = 10^{-5}u$ puts u near 1.1. A corresponding y scale can also be chosen near its reference magnitude. Transform both equations and all bounds explicitly if using such a reformulation. Do not impose a small-x bound while claiming to retain the exchanged large-x branch.

The first Jacobian row is (10000y,10000x), so its entries have very different sizes at the small-x root. Check the raw equations as well as scaled versions. A change in raw residual may reflect scaling instead of a proportional improvement in coordinates.

23 8 Example 7 Check tangent and exponential intersections

Use the tangent–exponential system below. The circle bounds both coordinates, while tangent is undefined at $x = \pm\pi/2$ within the box. Angles are in radians.

```
var x,y
tan(x)=exp(y)
x^2+y^2=9
bound x=[-3,3]
bound y=[-3,3]
```

x reference	y reference
-2.8073636809617375	-1.0576904853580578
-1.6523782020700648	2.5039261724986422
0.049766498953847147	-2.9995871875279566
1.4966590429435061	2.6000022517635304

The four listed candidates were refined independently. Because exp(y) is positive, admissible x must lie where tan(x) is positive: within this box those intervals are $[-3,-\pi/2)$ and $(0,\pi/2)$. No point at a tangent pole is admissible. A sign change across a pole does not imply an intervening root.

On a valid interval, substitution $y = \ln(\tan(x))$ gives the scalar equation $x^2 + [\ln(\tan(x))]^2 = 9$. This reduction supports independent checking. The circle and sign restrictions narrow the search, but the four numerical candidates alone do not constitute a certified count over all valid intervals.

The supplied double run reports four rows, a largest residual of about 1.89×10^{-12} and a largest listed Found seconds value of 0.0021939 s. Treat these as a developer-reported observation. A paired performance comparison requires a matching extended-mode four-row result.

For each candidate evaluate $\tan(x) - \exp(y)$ and $x^2 + y^2 - 9$ independently at full exported precision. A rounded display can obscure the circle residual or greatly alter the first residual near a pole. Record each residual separately and confirm that the input coordinate is inside the original function domain.

23 9 Example 8 Audit reported double and extended results

The attachment calls the following three-variable system a Troesch problem. Here it is treated as the stated algebraic hyperbolic-sine system. A claim that it is a particular discretization of a boundary-value problem would also require the differential equation, mesh, boundary conditions and scaling.

```
var x1,x2,x3
2*x1-x2+sinh(5*x1)=0
-x1+2*x2-x3+sinh(5*x2)=0
-x2+2*x3+sinh(5*x3)-1=0
bound x1=[0,1]
bound x2=[0,1]
bound x3=[0,1]
```

Coordinate	Independent reference
x1	0.00286634928528186874576249977396 95
x2	0.02006493562296981603470626177793 2
x3	0.13775658012451754209479661006027

The system is the gradient of $\Phi = x1^2 + x2^2 + x3^2 - x1x2 - x2x3 + [\cosh(5x1) + \cosh(5x2) + \cosh(5x3)]/5 - x3$. Its Hessian has diagonal entries $2 + 5\cosh(5xi)$ and adjacent off-diagonal entries -1. Each diagonal is at least 7 and exceeds the sum of the absolute off-diagonal entries in its row.

The symmetric Hessian is positive definite everywhere, and the quadratic part is positive definite, making Φ strictly convex and coercive. It has one minimizer and hence one stationary point over all real coordinates. The independently calculated root lies in the given box. This establishes uniqueness by structure rather than by the number of returned rows.

Source run	Reported maximum residual	Reported Found seconds
Double	$5.9674487573602164 \times 10^{-15}$	0.0000683
Extended	$3.1265873855642740 \times 10^{-25}$	0.0003754

Re-evaluating the exported coordinate strings with 90-decimal working precision gives maximum residuals about $5.97306051870808 \times 10^{-15}$ for the double row and $3.12658738553892 \times 10^{-25}$ for the extended row. The slight differences from the reported residuals are consistent with evaluating decimal exports rather than the original internal representations and arithmetic path. These checks confirm a substantial residual reduction for these two supplied vectors.

The extended vector is printed below exactly as supplied. The table of independent references above intentionally shows fewer digits. Agreement of a prefix does not establish every printed digit as accurate.

x1 = 0.00286634928528186874576253158265914922
x2 = 0.02006493562296981603470631354043573 68
x3 = 0.13775658012451754209479665432002193 9

The reported discovery-time ratio is approximately 5.50. It describes these observations, not a general slowdown factor. The attachment omits enough run settings and repetitions to prevent a controlled benchmark conclusion. The bounds printed in this chapter are explicit teaching bounds and are not known to be the bounds of those earlier runs.

23 10 Example 9 Distinguish close roots from close coefficients

A reliable close-root test should have a known answer set. The following constructed polynomial has two distinct roots separated by 2×10^{-10}. Factored input preserves the intended small offset without hiding it in a rounded expanded constant.

```
var x
(x-1)^2=1e-20
bound x=[0.999999999,1.000000001]
```

Root	Exact value
Lower	0.9999999999
Upper	1.0000000001

At x = 1 the residual is -10^{-20}, although the point is not a root and lies 10^{-10} from each root. An absolute residual criterion of 10^{-9} would not, by itself, distinguish this midpoint from either root. A tighter tolerance is useful, but root separation and duplicate handling must also be assessed.

The derivative magnitude at each root is 2×10^{-10}. If the right side changes from δ to δ + h, the root displacement is approximately h/(2√δ) for |h| much smaller than δ. This explains why small coefficient changes can matter greatly. As δ approaches zero the two roots merge into a repeated root.

The attachment also supplies a different coupled test. Its three cubic constants should not be described as three known roots of the full system.

```
var x,y
(x-1.0001)*(x-1.0002)*(x-1.0003)=y
x^2+y^2=1.0004
bound x=[-2,2]
bound y=[-2,2]
```

At x = 1.0002 the cubic equation gives y = 0, but the circle residual is $1.0002^2 - 1.0004 = 4 \times 10^{-8}$. Thus this cubic zero is not an exact solution of the coupled system. Eliminating y gives a degree-six polynomial in x; its real roots must be checked in both original equations. This chapter makes no exhaustive root-count claim for that coupled test.

23 11 Example 10 Count periodic roots within an explicit domain

Use the sine–cosine pair in the attachment, restricting both coordinates to $0 \le x < 2\pi$ and $0 \le y < 2\pi$. The finite decimal upper bounds below approximate that interval from below and retain all four reference roots.

```
var x,y
sin(x)*cos(y)=0.5
cos(x)*sin(y)=0.5
bound x=[0,6.283185307179586]
bound y=[0,6.283185307179586]
```

x exact	y exact
π/4	π/4
3π/4	7π/4

x exact	y exact
5π/4	5π/4
7π/4	3π/4

Adding the equations gives sin(x+y) = 1, so x+y = π/2 + 2kπ. Subtracting gives sin(x−y) = 0, so x−y = mπ. The domain allows sums π/2 and 5π/2. For the first sum only m = 0 is admissible; for the second sum m = −1, 0 and 1 give the other three rows. Hence the domain contains exactly four distinct roots.

The Jacobian determinant is cos(x+y)cos(x−y), which vanishes at every reference root. This is a singular system even though the domain contains finitely many isolated roots. Numerical refinement or coordinate-error estimation based on a nonsingular Jacobian must account for that fact.

Outside the restricted domain periodic copies continue indefinitely. A report of 400 solutions from a finite run cannot establish that every unrestricted root was found. Extended precision may improve a candidate residual, while an independent domain argument is still needed for completeness. The nonzero stationary points of optimization benchmarks likewise remain valid roots of their stationary equations; they should not be dismissed as false roots merely because they are not global minimizers.

23 12 Run a controlled precision comparison

Use identical input files and bounds for a within-product comparison. Record executable version, arithmetic mode, requested tolerance, computational budget, machine, thread settings and seeds if exposed. A useful test records both a same-tolerance comparison, to assess cost at one target, and a stricter extended-mode target, to assess attainable refinement. These answer different questions.

The existing book documents the following command-line form. Use it only with a build and license that support extended mode. Save the Example 8 input as ch12_sinh.txt. A tolerance is a requested numerical threshold; its precise stopping semantics must follow the installed application documentation.

```
equation_solver.exe --file ch12_sinh.txt --seconds 1 --tol 1e-9
equation_solver.exe --file ch12_sinh.txt --seconds 1 --extend --tol 1e-9
equation_solver.exe --file ch12_sinh.txt --seconds 1 --extend --tol 1e-24
```

Repeat runs rather than drawing a timing conclusion from one pair. Match corresponding roots by their coordinates and mathematical branches, not by row order. Report discovery times separately from total elapsed process time. Do not add per-row Found seconds values unless the program explicitly defines them as independent durations to be summed. Separate search, refinement, export and process-startup costs when those measurements are available.

Record for each run	Purpose
Exact equations and bounds	Fix the mathematical task
Mode tolerance budget and stopping reason	Fix the requested search and acceptance conditions
Full coordinate strings and individual residuals	Permit independent re-evaluation
Distinct admissible rows and matching rule	Avoid row-order and duplicate-count errors
Repeated discovery and elapsed timings	Describe variability and the chosen timing metric

23 13 Verify exported values independently

Set checking precision before converting decimal strings. Converting through a binary64 variable first discards digits that cannot be reconstructed by increasing precision later. The following standard-library Python checker evaluates the exported Example 8 vector using Decimal arithmetic. It checks that vector rather than calling the solver or proving a root exists.

```
from decimal import Decimal as D, localcontext

raw = [
 "0.0028663492852818687457625315826591492 2",
 "0.0200649356229698160347063135404357368",
 "0.13775658012451754209479665432002 1939",
]

def check(digits):
   with localcontext() as ctx:
      ctx.prec = digits
      a, b, c = map(D, raw)
      def sh(t):
         return (t.exp() - (-t).exp()) / D(2)
      f = [2*a-b+sh(5*a),
          -a+2*b-c+sh(5*b),
          -b+2*c+sh(5*c)-1]
      return f, max(abs(v) for v in f)

for digits in (60, 90):
   residuals, maximum = check(digits)
   print(digits, residuals, maximum)
```

Repeat at 60 and 90 working decimal digits and compare the leading residual digits. Agreement supports stability of the evaluation, but is not a formal interval bound. For this example the moderate arguments and high working precision make the direct hyperbolic-sine formula adequate; subtraction in that formula can itself require care when its argument is extremely small.

For the small-change example, an independent binary64 illustration uses Python’s math functions:

```
from math import exp, expm1, log, log1p
t = 1e-20
print((exp(t)-1)/t, expm1(t)/t)
print(log(1+t)/t, log1p(t)/t)
```

A typical binary64 result is zero for each direct expression and approximately one for each specialized expression. Function libraries and evaluator transformations affect exact output. A mathematically zero residual, a computed zero residual and a rigorously enclosed zero are distinct statements; report which one has actually been established.

23 14 Practice problems

Problem 1 Interpret a decimal export

A binary128 result is printed with 37 significant digits. Does that establish 37 accurate digits?

Problem 2 Use a nonsingular residual

For Example 1, bound the error from $|r| \le 10^{-24}$ and xhat in [1,2].

Problem 3 Rationalize a difference

Rewrite √(1+t) − 1 to avoid subtraction. What is the limiting scaled result after division by t?

Problem 4 Check small logarithms

State the real domain of log1p(t) and explain why log(1+t) can return zero at $t = 10^{-20}$.

Problem 5 Audit a repeated root

For $(x-1)^4 = 0$, what coordinate error magnitude corresponds to residual 10^{-24}?

Problem 6 Perturb nearly dependent data

In Example 5, let the second right side increase by 10^{-15}. Find the exact changes in x and y.

Problem 7 Interpret source timings

Compute the extended-to-double discovery-time ratio in Example 8. Does this establish the general overhead of extended precision?

Problem 8 Check the constructed root midpoint

In Example 9 evaluate the residual at x = 1 and explain why the midpoint is not a root.

Problem 9 Count a smaller periodic domain

Restrict Example 10 to $0 \le x < \pi$ and $0 \le y < \pi$. How many roots remain?

Problem 10 Separate refinement from discovery

If extended mode reduces each returned residual but finds the same rows, what has improved and what remains unproved?

23 15 Worked answers

Answer 1 Interpret a decimal export

No. Binary128 has 113 significant binary bits, corresponding to roughly 34 decimal digits of capacity. A longer export can preserve the stored value without proving that every printed digit is accurate. Coordinate accuracy also depends on conditioning, input conversion and convergence.

Answer 2 Use a nonsingular residual

Using $|xhat - \sqrt{2}| = |r|/(xhat + \sqrt{2})$, the denominator is at least $1 + \sqrt{2}$. Therefore the error is at most $10^{-24}/(1 + \sqrt{2}) \approx 4.142135624 \times 10^{-25}$. A tighter estimate near the root uses denominator $2\sqrt{2}$.

Answer 3 Rationalize a difference

The stable form is $t/[\sqrt{(1+t)} + 1]$ for $t \ge -1$. For $t \ne 0$, division by t gives $1/[\sqrt{(1+t)} + 1]$, whose limit as t approaches zero is 1/2. The original quotient is undefined at zero until extended by continuity.

Answer 4 Check small logarithms

The real domain is $t > -1$. Near 1, binary64 spacing is much larger than 10^{-20}, so $1 + t$ can round to 1; then log(1) is zero. log1p computes the small logarithmic change without first discarding t in that addition.

Answer 5 Audit a repeated root

If the coordinate error is e, the residual is e^4. Thus $|e| = (10^{-24})^{(1/4)} = 10^{-6}$. A very small residual is compatible with only a modest coordinate accuracy at this fourth-order root.

Answer 6 Perturb nearly dependent data

Here $\varepsilon = 10^{-12}$ and $\delta = 10^{-15}$. Hence $\Delta y = \delta/\varepsilon = 0.001$ and $\Delta x = -0.001$, giving $x = 0.999$ and $y = 1.001$. More arithmetic precision cannot remove a genuine perturbation in the data.

Answer 7 Interpret source timings

The ratio is 0.0003754/0.0000683 ≈ 5.496339678. It describes the two supplied Found seconds observations. Repeated matched runs with recorded settings and defined timing metrics are needed before generalizing.

Answer 8 Check the constructed root midpoint

The residual is $(1-1)^2 - 10^{-20} = -10^{-20}$. The midpoint is 10^{-10} away from each true root and is not one of them. A permissive absolute residual test can nevertheless accept it unless other acceptance and refinement conditions distinguish it.

Answer 9 Count a smaller periodic domain

Only $(\pi/4,\pi/4)$ remains. The other three reference roots have at least one coordinate greater than or equal to π. The sum-and-difference argument excludes any further root in the smaller domain.

Answer 10 Separate refinement from discovery

The residuals of the returned vectors have improved under the reported evaluation. Without conditioning or a reference error estimate, the same numerical reduction cannot be asserted for coordinate errors. The unchanged row set also does not prove completeness unless an independent count or validated isolation supports it.

23 16 References and a reliable reporting standard

Independent checks here address the stated equations and exported coordinates. They do not establish internal algorithms, competitor performance or unspecified benchmark conditions.

[1] NIST Digital Library of Mathematical Functions Section 3 1 Arithmetics and Error Measures
https://dlmf.nist.gov/3.1

[2] GNU C Library Manual Exponents and Logarithms
https://sourceware.org/glibc/manual/latest/html_node/Exponents-and-Logarithms.html

[3] Python Standard Library Documentation Decimal Arithmetic
https://docs.python.org/3/library/decimal.html

Preserve exact inputs, domains, arithmetic mode, tolerances, full exports and independent residuals. State the basis for accuracy and root-count claims, and report model uncertainty separately. This makes the benefit of extended precision concrete and independently reviewable.

23 17 Current edition workflow and scope

23 18 Precision in trajectories and fields

In supported ODE/BVP/field modes, --extend selects binary128 arithmetic subject to entitlement. Standalone quadrature uses binary128 guard arithmetic even in standard mode, reporting binary128_guard. Mode-specific tolerances differ from algebraic acceptance tolerances. Extra precision reduces roundoff but cannot remove discretisation, conditioning or unresolved-feature error.

Exact integer coordinates use arbitrary-size integer arithmetic on recognised routes and are not limited to binary128 digits. Conversely, huge bound support is limited to supported simple literal forms and does not extend every real calculation. Desktop plots still use double display arithmetic, while exported strings preserve the selected engine output.

Chapter 24 Automation and Integration

Automation makes an equation model repeatable: a program prepares the input, runs the solver, retains the response and verifies the returned values. Integration brings that calculation into another application. In both cases the host must preserve the model, the meaning of the result and the numerical precision. Starting a process successfully is only one part of a correct integration.

This chapter develops ten practical integration examples using the current C ABI version 1. Algebraic examples use equation_solver.result.v1; trajectory and field modes have their own schemas. The current Developer Guide and the public header define the interface.

Commands use Windows Command Prompt syntax unless stated otherwise. Python code uses the standard library and UTF-8 model text. Each labelled file is independent unless its continuation is explicitly stated. The reference root sets are calculated from the equations; the Patodia executable and Windows DLL were not run while preparing this chapter.

24 1 Choose a boundary that matches the application

The CLI executable equation_solver.exe and native equation_solver.dll share the numerical engine. The documented desktop host PatodiaEquationSolver.exe starts the CLI as a separate worker process; it does not call eq_solve directly. This affects stopping work, deployment and failure isolation. A desktop update and a worker update can therefore be separate changes.

Boundary	Suitable use	Host responsibility
CLI subprocess	Language-neutral jobs and isolated workers	Arguments streams exit codes deadlines and JSON
Native DLL	Direct calls within an authorized x64 host	C ABI declarations result lifetime and dependencies
Desktop host	Interactive editing and result inspection	Run snapshots asynchronous work and worker settings
Excel interface	Tabular model inputs and independent formulas	Version-specific add-in contract and numeric precision

Use the CLI when the application needs a strict external deadline or forced cancellation. The v1 DLL call is synchronous and has no exported cancellation handle or progress callback. A worker thread can keep an interface responsive, but terminating a thread inside native code is not a sound cancellation design. Independent DLL calls may run concurrently under the guide’s contract, with separate owned result handles.

24 2 Example 1 Run and check a reproducible model

Save the following as product_model.txt. Both unknowns are integers and their declared output order is a,b. The finite bounds make the intended root set transparent.

```
var a,b
int a,b
a+b=7
a*b=12
bound a=[0,7]
bound b=[0,7]
```

```
dist\equation_solver.exe --file product_model.txt --check --json
dist\equation_solver.exe --file product_model.txt ^
  --seconds 2 --seed 42 --max-solutions 100 --json
```

a	b	Independent verification
3	4	3 + 4 = 7 and 3 × 4 = 12
4	3	4 + 3 = 7 and 4 × 3 = 12

Eliminating b gives $a^2 - 7a + 12 = (a - 3)(a - 4)$, so exactly these two ordered pairs satisfy the model. A syntax check returns syntax_valid without searching. The subsequent solve requests a search and should be compared with this reference set rather than merely checked for a nonempty response.

Use --file explicitly. The guide notes that a nonexistent positional path can be interpreted as equation text, while --file produces a clearer file-open failure. For stdin, use --file - and send the model bytes through the process input stream. Option tokens and model statements are different interfaces; do not combine them into an arbitrary command string.

24 3 Example 2 Preserve a complete process record

Save the next two blocks consecutively as run_solver.py. The function takes an executable, model path and output folder. It creates a new run directory, records hashes and arguments, and preserves both output streams even when the worker exceeds the host deadline. Binary capture retains the original response bytes [1].

```
import hashlib
import json
import math
import subprocess
import tempfile
import time
from pathlib import Path

def run_case(exe, model_file, output_root, *,
             seconds=2, deadline=15, extend=False):
    exe = Path(exe).resolve()
    model_file = Path(model_file).resolve()
    if not exe.is_file() or not model_file.is_file():
        raise FileNotFoundError('Executable or model missing')
    if (not math.isfinite(seconds) or seconds <= 0
            or not math.isfinite(deadline) or deadline <= 0):
        raise ValueError('Use positive finite work limits')
    model = model_file.read_bytes()
    model.decode('utf-8')
    root = Path(output_root).resolve()
    root.mkdir(parents=True, exist_ok=True)
    folder = Path(tempfile.mkdtemp(prefix='run-', dir=root))
    (folder / 'model.txt').write_bytes(model)
    cmd = [str(exe), '--file', '-', '--json',
           '--seconds', str(seconds), '--seed', '42',
           '--max-solutions', '100']
    if extend:
        cmd += ['--extend', '--tol', '1e-24']
    else:
        cmd += ['--tol', '1e-9']
```

Continue in the same file. The record includes a hash of the executable and exact submitted model. The working directory is the runtime folder so the worker has a predictable local context.

```
  record = {'command': cmd, 'model_sha256':
          hashlib.sha256(model).hexdigest(),
          'exe_sha256': hashlib.sha256(
             exe.read_bytes()).hexdigest(),
          'external_deadline_seconds': deadline}
  started = time.perf_counter()
  stdout = stderr = b''
  try:
    result = subprocess.run(
      cmd, input=model, capture_output=True,
      cwd=str(exe.parent), timeout=deadline,
      check=False, shell=False)
    stdout, stderr = result.stdout, result.stderr
    record.update(outcome='completed',
              returncode=result.returncode)
  except subprocess.TimeoutExpired as exc:
    stdout = exc.output or b''
    stderr = exc.stderr or b''
    record['outcome'] = 'external_timeout'
  except OSError as exc:
    record.update(outcome='launch_error',
              diagnostic=str(exc))
  finally:
    record['process_wall_seconds'] = (
      time.perf_counter() - started)
    (folder / 'stdout.bin').write_bytes(stdout)
    (folder / 'stderr.bin').write_bytes(stderr)
    (folder / 'run.json').write_text(
      json.dumps(record, indent=2), encoding='utf-8')
  return folder, record
```

The work allowance --seconds and the subprocess timeout have different roles. The former controls solver work approximately; the latter is a host deadline that can interrupt before final JSON is produced. The standard subprocess timeout kills and waits for its immediate child, but this example is not a process-tree supervisor for optional external certification backends. Use a suitable supervisor when descendants must also be terminated.

Invoke the function from a small script in the same folder. This usage prints the unique evidence directory and the host outcome. It does not yet interpret the solver JSON.

```
from run_solver import run_case

folder, record = run_case(
  "C:/Patodia/equation_solver.exe",
  "product_model.txt", "runs",
  seconds=2, deadline=15)
print(folder)
print(record["outcome"], record.get("returncode"))
```

A missing input raises FileNotFoundError before launch. A loader or launch error is recorded as launch_error; a host interruption is external_timeout. Neither is a mathematical no-root result. This teaching wrapper reads the executable to hash it on each call; a production batch can cache a verified build hash rather than include that repeated I/O in its preparation cost.

24 4 Example 3 Decode status and preserve decimal strings

After a completed call, decode stdout as UTF-8 and parse it as one JSON document. Do not strip a diagnostic prefix until a brace happens to appear: unexpected stdout can indicate a protocol or launch problem. A nonzero exit code must be interpreted before treating the response as a successful calculation.

```
import json

if record["outcome"] != "completed":
   raise RuntimeError(record["outcome"])
if record["returncode"] not in (0, 2):
   raise RuntimeError("Inspect run.json and stderr.bin")
raw = (folder / "stdout.bin").read_bytes()
try:
   data = json.loads(raw.decode("utf-8"))
except (UnicodeError, json.JSONDecodeError) as exc:
   raise RuntimeError("Invalid solver response") from exc
if not isinstance(data, dict):
   raise RuntimeError("Expected a result object")
print(data.get("status"), data.get("reason"))
```

Exit code	Meaning in the supplied guide
0	Successful solve satisfied constant system or syntax check
1	Input option or engine error
2	No point found unsatisfied constant system or completed empty certification domain
3	Compute licence or entitlement failure

Before accepting a response in a deployed application, validate the equation_solver.result.v1 schema identifier and field layout against the matching build. The snippets here validate selected structural properties; they are not a complete schema validator. Status-specific responses can intentionally omit fields used by numerical search results.

Save the next block as result_rows.py. The variables array gives the name and order of every values array. The explicit length check avoids silently truncating a row with zip. Coordinates remain strings in the raw response and become Decimal objects only when the host needs arithmetic.

```
from decimal import Decimal

def decimal_rows(data):
  variables = data.get('variables', [])
  names = [v['name'] for v in variables]
  if len(names) != len(set(names)):
    raise ValueError('Duplicate variable names')
  rows = []
  for solution in data.get('solutions', []):
    values = solution['values']
    if len(values) != len(names):
      raise ValueError('Coordinate count mismatch')
    if any(not isinstance(v, str) for v in values):
      raise TypeError('Coordinates must be strings')
    parsed = [Decimal(v) for v in values]
    if any(not v.is_finite() for v in parsed):
      raise ValueError('Nonfinite coordinate')
    rows.append(dict(zip(names, parsed)))
  return rows
```

```
from result_rows import decimal_rows

rows = decimal_rows(data)
for row in rows:
  print(row)
```

Do not infer failure solely from an empty solutions array. syntax_valid did not search, and constant_satisfied can describe a satisfied constant system with no unknown vector. feasible=false means no point was returned, not necessarily that infeasibility was proved. Interpret reason, search_complete, scope and certification together. In the documented exact affine-factor path, search_complete remains false because that general field is reserved for the certification mechanism; the specialized reason carries additional evidence.

Coordinates, tolerance and maximum_residual are decimal strings, whereas timing fields are ordinary JSON numbers or null where documented. Converting a coordinate through float before Decimal loses digits. Decimal construction from a string preserves those digits, but subsequent arithmetic still needs an adequate decimal context. Keep the original strings for export and independent checking.

24 5 Example 4 Batch jobs without losing failures

A batch should retain every attempted case, including externally interrupted or failed runs. Use the wrapper from Example 2 and one unique directory per call. The loop below is sequential, making result ownership and resource use straightforward. Save it beside run_solver.py as batch_cases.py.

```
import json
from pathlib import Path
from run_solver import run_case

exe = "C:/Patodia/equation_solver.exe"
summary = []
for model in sorted(Path("models").glob("*.txt")):
  try:
    folder, rec = run_case(exe, model, "batch_runs")
    item = {"model": str(model), "folder": str(folder),
        "outcome": rec["outcome"],
        "returncode": rec.get("returncode")}
  except (OSError, ValueError, UnicodeError) as exc:
    item = {"model": str(model),
        "outcome": "host_input_error",
        "diagnostic": str(exc)}
  summary.append(item)
  Path("batch_summary.json").write_text(
    json.dumps(summary, indent=2), encoding="utf-8")
```

The summary is updated after each attempted model, but an abrupt machine failure can still interrupt a write. Production systems that require crash-consistent indexes should write a temporary index and atomically replace the previous one. The per-run evidence remains separate, so an invalid response should not erase earlier successful results.

This loop records process outcomes rather than asserting numerical acceptance. Apply Example 3 and model-specific verification to each completed response. A batch with exit code 2 needs interpretation of its status and domain; it should not disappear from a report as though no run occurred.

If parallelism is added, limit worker count and isolate filenames, streams and temporary backend artifacts. More workers can alter timings and resource availability. Preserve seeds and options, but do not assume a fixed seed makes every timing or discovery order identical across machines and builds.

24 6 Example 5 Generate a parameter study from validated data

Chapter 12 used supply $q = p^2$ and demand $q = A - p$. Generate three independent cases with A equal to 30, 31 and 42. Parse A as Decimal, require a finite positive value and use decimal-dot formatting. Each model includes its own bounds; a demand shift must not retain an obsolete quantity ceiling that excludes the new equilibrium.

```
from decimal import Decimal
from pathlib import Path

root = Path("models")
root.mkdir(exist_ok=True)
for i, text in enumerate(("30", "31", "42"), 1):
    A = Decimal(text)
    if not A.is_finite() or A <= 0:
        raise ValueError("Demand intercept must be positive")
    s = format(A, "f")
    model = (f"var p,q\nq=p^2\nq={s}-p\n"
             f"bound p=[0,{s}]\nbound q=[0,{s}]\n")
    (root / f"market_{i:02d}.txt").write_text(
        model, encoding="utf-8")
```

A	Positive price	Quantity
30	5	25
31	(√125 − 1)/2 ≈ 5.090169944	31 − p ≈ 25.909830056
42	6	36

The reduced equation is $p^2 + p - A = 0$. Its derivative is $2p + 1 > 0$ for $p \geq 0$, giving one positive root for $A > 0$. The formula $p = (\sqrt{(1 + 4A)} - 1)/2$ provides an independent oracle for these cases. Recheck $p^2 - q$ and $q + p - A$ from every exported row, with a precision appropriate to the requested tolerance.

Generate mathematical text from validated scalar data, with explicit multiplication and parentheses. The model language has no implicit unit system: currency, quantity and accounting period must be stated by the application. US and EU input styles have different separators; use an explicit per-call style when needed. JSON remains decimal-dot regardless of the model notation.

24 7 Example 6 Call the documented C ABI from C plus plus

Use the public equation_solver.h shipped with the matching DLL. The guide documents C ABI version 1, cdecl calls, UTF-8 model text and opaque eq_result handles. Save the following as chapter13_client.cpp in the extracted source root. The example uses the same two-root model as Example 1.

```
#include "equation_solver.h"
#include <iostream>
#include <memory>
#include <string>

int main() {
   if (eq_abi_version() != 1) return 1;
   const char* opts[] = {"--seconds", "2",
                         "--max-solutions", "100"};
   using Result = std::unique_ptr<eq_result,
                                  decltype(&eq_result_free)>;
   Result r(eq_solve(
      "var a,b; int a,b; a+b=7; a*b=12;"
      "bound a=[0,7]; bound b=[0,7]",
      4, opts), &eq_result_free);
   if (!r) return 1;
   int status = eq_result_status(r.get());
   const char* text = eq_result_json(r.get());
   std::string copied = text ? text : "";
   if (status != 0 && status != 2) {
      const char* err = eq_result_error(r.get());
      std::cerr << (err ? err : "Solver error") << "\n";
   }
   std::cout << copied << "\n";
   return status;
}
```

```
g++ -std=c++17 -Iengine chapter13_client.cpp ^
  -Lengine\bin -lequation_solver ^
  -o engine\bin\chapter13_client.exe
engine\bin\chapter13_client.exe
```

The option count is four tokens, even though there are two named switches. The model is a separate argument. DLL options reject --file, --eq and --json; obtain JSON through eq_result_json. Copy borrowed UTF-8 strings while the result is alive and release the result only through eq_result_free. The unique_ptr deleter provides deterministic release even if later C++ processing throws.

A null result means no handle could be returned, for example after allocation failure. Ordinary solve errors normally have a result handle with status and error information. Keep null-handle failure separate from status 2. Do not call free or delete on borrowed result strings, and do not retain a pointer after releasing its owner.

These commands build a client of the supplied engine. They do not rebuild the binary128 implementation with MSVC. The guide requires an x64 GCC toolchain with true __float128 and quadmath for an engine rebuild; an MSVC client needs the matching import library and header.

24 8 Example 7 Bind the DLL from Python

Save the next two blocks consecutively as dll_case.py and run it with 64-bit Python. ctypes.CDLL selects the documented C calling convention [2]. Every signature must be set before calling a function; otherwise a pointer-sized result can be misinterpreted.

```
import ctypes as C
import json
import os
import sys
from pathlib import Path
from result_rows import decimal_rows

path = Path(sys.argv[1]).resolve()
search = None
if os.name == "nt":
    search = os.add_dll_directory(str(path.parent))
lib = C.CDLL(str(path))
lib.eq_abi_version.argtypes = []
lib.eq_abi_version.restype = C.c_uint
if lib.eq_abi_version() != 1:
    raise RuntimeError("Unsupported ABI")
lib.eq_solve.argtypes = [C.c_char_p, C.c_int,
                         C.POINTER(C.c_char_p)]
lib.eq_solve.restype = C.c_void_p
for name in ("eq_result_json", "eq_result_error"):
    fn = getattr(lib, name)
    fn.argtypes = [C.c_void_p]
    fn.restype = C.c_char_p
lib.eq_result_status.argtypes = [C.c_void_p]
lib.eq_result_status.restype = C.c_int
lib.eq_result_free.argtypes = [C.c_void_p]
lib.eq_result_free.restype = None
```

Continue in the same file. The returned bytes are decoded and copied into Python objects before the result is freed. Keep result_rows.py from Example 3 in the same folder.

```
model = ("var a,b; int a,b; a+b=7; a*b=12;"
        "bound a=[0,7]; bound b=[0,7]")
opts = [b"--seconds", b"2", b"--max-solutions", b"100"]
args = (C.c_char_p * len(opts))(*opts)
handle = None
try:
    handle = lib.eq_solve(
        model.encode("utf-8"), len(opts), args)
    if not handle:
        raise MemoryError("No result handle")
    status = lib.eq_result_status(handle)
    raw = lib.eq_result_json(handle)
    if status not in (0, 2):
        err = lib.eq_result_error(handle)
        raise RuntimeError(
            err.decode("utf-8") if err else "Solver error")
    if not raw:
        raise RuntimeError("Missing JSON response")
    data = json.loads(raw.decode("utf-8"))
    print(data.get("status"), decimal_rows(data))
finally:
    if handle:
        lib.eq_result_free(handle)
    if search is not None:
        search.close()
```

```
python dll_case.py C:/Patodia/equation_solver.dll
```

The added DLL-directory handle must remain alive while dependencies may be needed. A missing transitive runtime DLL can cause loading to fail even when equation_solver.dll exists. This script is a short-lived demonstration; a reusable library wrapper should also close its loader resources if initialization fails and encapsulate result lifetime in a context manager.

The guide permits independent concurrent calls with separate results, but a result must never be freed while another thread reads it. An interactive host should move the synchronous solve off its UI thread and marshal completed data back safely. For strict cancellation use supervised process workers, since the ABI does not expose a cancellation operation.

24 9 Example 8 Keep Excel inputs and numerical evidence separate

Use the sample workbook and add-in instructions belonging to the installed release. The developer guide does not specify Excel macro names, cell addresses or an add-in calling signature, so this chapter does not invent them. The following workbook arrangement can be implemented through that documented interface or by importing the CLI result.

Worksheet	Content	Check
Inputs	a + b = 7 and ab = 12 model and settings	Keep integer declarations and bounds visible
Results	a in column A and b in column B	Associate every row with the completed run

Worksheet	Content	Check
Evidence	Model options raw JSON and build identity	Retain full coordinate strings and status

For the small integer example, place a returned pair in A2 and B2 on Results. The formulas =A2+B2-7 and =A2*B2-12 should both evaluate to zero, and =AND(A2=INT(A2),B2=INT(B2),A2>=0,A2<=7,B2>=0,B2<=7) checks integrality and bounds. These formulas assume the English Excel function and separator convention.

Excel numeric arithmetic is based on double precision and does not preserve arbitrary long decimal values [3]. Import extended coordinates into explicitly text-formatted fields when complete digits are needed. Keep an independent high-precision check outside ordinary worksheet numeric cells. Displaying more decimal places cannot restore digits already rounded during import.

A CSV opened automatically by Excel can trigger type conversion even if the export contains long decimal strings. Use an explicit text import workflow or the version-specific workbook interface. A human-readable rounded result table can coexist with a full-precision text record; never overwrite the authoritative coordinates with the display formatting.

24 10 Example 9 Preserve desktop snapshots and responsive hosts

In the documented desktop, a run snapshots the equations, settings and bound overrides before starting a worker. Editing the model does not rewrite the last completed result. An integration should preserve this distinction so a visible row cannot silently become associated with a different model.

Run product_model.txt, retain its two-root result, then change the sum equation to a + b = 8 without launching a second solve. The retained rows still belong to the completed sum-seven model. They must not be labelled as solutions of the edited model. The revised equations have integer pairs (2,6) and (6,2), illustrating why the association matters.

A .pdeq project uses the documented patodia.equations.project.v1 format and preserves model, settings and bounds. A plain text equation file does not preserve the same project state. Bound overrides can change the active mathematical problem; include them in the run evidence rather than saving only the visible equation text.

The desktop runs asynchronously, drains stdout and stderr, and decodes the final JSON. Stop terminates the worker process tree; the documented protocol does not return a partial final solution set after that interruption. Previous completed results can remain visible only with their original run label. Host process time and found_seconds are different measurements.

Desktop settings are explicit and can differ from CLI defaults. The guide records a desktop default of two seconds and 30000 starts, while the CLI engine defaults are 20 seconds and 12000 starts. For a comparison, pass the same settings deliberately instead of attributing the effect of changed defaults to an integration boundary.

24 11 Example 10 Verify an integration at its public boundary

Begin with independent mathematical references, then exercise failure and precision paths. Compare unordered sets of ordered vectors: row order may change, but swapping labelled coordinates can be a genuinely different solution. The following cases provide a compact acceptance suite for a host application.

Case	Input or action	Expected interpretation
Two real roots	$x^2 = 4$ with x in [−3,3]	Reference roots −2 and 2
Constrained root	Same model with x in [0,3]	Reference root 2

Case	Input or action	Expected interpretation
Integer pairs	Example 1	Exactly (3,4) and (4,3)
No real roots	$x^2 + 1 = 0$ on $[-3,3]$	No root; mathematical proof comes from $x^2 \geq 0$
Syntax only	Example 1 with --check	syntax_valid without a solve
Constant truth	$1 = 1$	Satisfied constant response without unknown vectors
Constant falsehood	$1 = 2$	Unsatisfied constant response
Missing file	Explicit --file for an absent path	Input failure not a no-root calculation
Forced host timeout	Deadline shorter than worker completion	External interruption not infeasibility
Extended export	$x^2 = 2$ with $x \geq 0$ and --extend	Retain decimal strings and check independently

For precision coverage, --extend requests double candidate discovery followed by binary128 polishing and final checking. --extend --quad-search requests binary128 discovery as well. Compiled capability, active entitlement and selected mode are separate facts. A capability query alone does not establish permission, and entitlement failures must remain distinct from mathematical search outcomes.

The guide also documents relative coordinate merging through --merge, a default returned-point limit through --max-solutions, and finite start budgets. Close roots can be merged under a permissive setting, and a point limit can stop a search before every reference root appears. --seconds 0 removes the time deadline but leaves other limits; it does not establish exhaustive search.

For DLL acceptance, call repeatedly, copy result strings before freeing handles and include error-result paths. For CLI acceptance, test executable paths containing spaces, invalid UTF-8 or invalid model input as applicable, empty stdout and malformed JSON. For numerical checks, substitute every candidate into the original equations rather than comparing only the host's display text.

24 12 Package and deploy the matching runtime

Keep the x64 executable, DLL, public header and appropriate import library matched to one release. Extract the complete runtime package to a normal folder. The guide identifies libquadmath-0.dll and libwinpthread-1.dll as engine dependencies, with libgcc_s_seh-1.dll imported transitively by quadmath. Verify the actual release dependencies instead of assuming one DLL file is sufficient.

All compute entry points enforce entitlement. Use the installed licence and documented options; an invalid explicit licence selection does not silently fall back to a trial. Deployment and redistribution rights depend on the applicable agreement. The integration's role is to surface entitlement failures clearly and preserve the existing activation state during updates.

For an internal service, apply bounded worker concurrency, model-size and output limits, per-job evidence folders and an external process-tree timeout policy where needed. The v1 DLL call runs in the host process, so native failure affects that host. Process isolation is useful when the application needs to contain such failures or forcibly stop a job.

Record executable hashes, runtime identity and the options actually forwarded to the engine. A GUI running beside an older equation_solver.exe can retain old numerical behavior even if the GUI file itself is newer. Test from a freshly extracted runtime folder with spaces in its path before distributing the integration.

24 13 Check numerical meaning after automation

A host should show returned rows, status, reason and precision without inventing a stronger conclusion. found_seconds records when a distinct point first passed final acceptance; improved duplicates do not reset that time, and coordinate sorting can put later-discovered rows first. Compare timestamps directly instead of summing them or treating the last displayed row as the last discovery.

Preserve original-equation residuals separately from display rounding and independent checks. Certification records can include enclosures, scope and unresolved regions. A numerical candidate with a small residual is different evidence from a certified enclosure or a complete domain proof. Store those distinctions with the run, including when the UI presents a simplified summary.

The chapter's Python process and row-decoding functions were exercised locally with simulated completed, empty, malformed, timed-out and launch-failure responses. The tests checked record retention and string precision. This is host-code validation only; live Windows CLI/DLL execution, licensing, loader behavior and workbook integration remain installation-specific checks.

24 14 Practice problems

Problem 1 Distinguish checking from solving

A run with --check returns no solution rows. Is the model infeasible?

Problem 2 Count option tokens

What count should eq_solve receive for {"--seconds","2","--extend","--tol","1e-24"}?

Problem 3 Preserve coordinate precision

Why is Decimal(float(text)) unsuitable for an extended coordinate?

Problem 4 Interpret an external timeout

The host kills a worker after 15 seconds without final JSON. What outcome should be recorded?

Problem 5 Generate a revised model

Change Example 5 to A = 56. Find the equilibrium and suitable nonnegative bounds.

Problem 6 Verify native lifetime

Can an application keep the pointer returned by eq_result_json after eq_result_free?

Problem 7 Choose discovery precision

Which documented options request binary128 search as well as polishing?

Problem 8 Retain a desktop snapshot

After changing a + b = 7 to a + b = 8, may the old rows be relabelled as current results?

Problem 9 Interpret an empty result

Why does an empty solutions array not by itself mean failure?

Problem 10 Assess an unrestricted budget

Does --seconds 0 prove the returned list is exhaustive?

24 15 Worked answers

Answer 1 Distinguish checking from solving

No. --check validates syntax and domain setup without searching. syntax_valid is the expected successful check status. A separate solve is required to request roots.

Answer 2 Count option tokens

Five. The count is the number of tokens, including option values and the standalone --extend flag. The model text is passed separately.

Answer 3 Preserve coordinate precision

float(text) rounds through binary64 first. Decimal then preserves that already rounded value rather than the original digits. Use Decimal(text), retain the source string and select an adequate arithmetic context for later calculations.

Answer 4 Interpret an external timeout

Record an external_timeout or equivalent interrupted outcome, the host deadline, available streams and elapsed time. Do not manufacture a mathematical no-solution result or a partial final solution set.

Answer 5 Generate a revised model

Elimination gives $p^2 + p - 56 = (p - 7)(p + 8)$. The admissible equilibrium is $p = 7$, $q = 49$. Bounds p and q in [0,56] include it. The negative price −8 is excluded.

Answer 6 Verify native lifetime

No. The pointer is borrowed from the result. Copy the UTF-8 content while the result is alive, then release the handle through eq_result_free. The host must not free the borrowed string independently.

Answer 7 Choose discovery precision

Use --extend --quad-search with a supporting build and entitlement. --extend alone uses double discovery with binary128 refinement and final verification.

Answer 8 Retain a desktop snapshot

No. The old rows belong to the completed sum-seven model. The edited sum-eight model has the distinct integer pairs (2,6) and (6,2); launch a new solve and associate its response with its own input snapshot.

Answer 9 Interpret an empty result

A syntax check has no roots because it did not search, and a satisfied constant system can have no unknown vectors. For a numerical search, no returned point also does not prove infeasibility. Interpret status, reason and proof scope.

Answer 10 Assess an unrestricted budget

No. It removes the time deadline, but starts, iterations, returned-point limits and other stopping controls still apply. Exhaustiveness requires an independent argument or an applicable validated completeness result over a stated domain.

24 16 References and integration record

The supplied Patodia Equation Developer Guide dated 28 October 2026 is the product-specific source for the CLI, JSON field meanings, C ABI version 1, lifetime rules, precision modes, desktop behavior and runtime dependencies. Its named source snapshot is the affine-factor update. Consult the matching equation_solver.h and examples from that release before compiling or shipping a binding.

[1] Python Standard Library Subprocess Management
https://docs.python.org/3/library/subprocess.html

[2] Python Standard Library ctypes Foreign Function Library
https://docs.python.org/3/library/ctypes.html

[3] Microsoft Learn Floating Point Arithmetic in Excel
https://learn.microsoft.com/en-us/troubleshoot/microsoft-365-apps/excel/floating-point-arithmetic-inaccurate-result

A reproducible integration record joins one exact model and option set to one identified runtime and one retained response. With ordered variables, full decimal strings, clear failure states and independent checks, automation extends the mathematical workflow of the book into applications without losing the evidence needed to assess an answer.

24 17 Current edition workflow and scope

24 18 Route every schema explicitly

Route by problem_type and schema: equation_solver.result.v1 for algebraic roots; equation_solver.ode_result.v1 for IVPs; equation_solver.bvp_result.v1 for BVPs; equation_solver.field_result.v1 for PDEs, integral equations and eigenproblems; equation_solver.sweep_result.v1 for parameter batches; equation_solver.quadrature_result.v1 for standalone quadrature. Read each success/accuracy flag and reason. C ABI v1 and the ten exports are unchanged; borrowed strings live until eq_result_free.

Commercial licences permit double-precision CLI and Desktop solves but reject DLL solving and extended mode. An eligible automatic trial can operate without a signed licence file. Test computation entitlement, not merely DLL loading or metadata. Invalid explicit licence selections must not fall back silently to a trial. See the current Developer Guide for the full deployment contract.

24 Run a workflow through the CLI or DLL

Add --extend to select extended precision where your licence and service support it. For a DLL client, pass the same workflow text as the model to eq_solve. Pass options such as --seconds and 20 as separate tokens, then release the result using eq_result_free. The DLL requires a DLL-enabled entitlement.

```
equation_solver.exe --file workflow.txt --seconds 20
equation_solver.exe --file workflow.txt --seconds 20 --json
equation_solver.exe --file workflow.txt --check
```

--check validates workflow structure and checks service models that can be checked without runtime substitution. A model with @parameters is validated when it runs. Run the workflow to verify time-cap support, runtime names and parameter values.

CLI exit code 0 means complete without unaccepted jobs, or syntax check passed. 2 means at least one unaccepted job or a resource stop. 1 indicates a syntax or execution error. In JSON, check the top-level status and the individual jobs and reports.

Chapter 25 Benchmarks Teaching and Further Exploration

Patodia Equation Solver can support three connected activities: testing numerical behavior, teaching mathematical modelling, and investigating new applications. A benchmark measures a defined computational task. A teaching activity develops a student's reasoning and verification skills. An industrial or research study asks whether a mathematical model explains useful behavior and supports a decision. The same equation may serve all three purposes, but the evidence required is different.

This chapter has three sections. Benchmarks develops a reproducible testing method using elementary reference problems and the supplied twenty-system nonlinear collection. Teaching shows how university courses and PhD projects can use equation solving without replacing derivation, experimental work or research judgement. Further Exploration identifies practical entry points across industries and develops small pilot models that can be extended with domain data.

The benchmark source is Patodia 20 Original Nonlinear Challenge Systems, Revision 2, dated 24 October 2026. Its historical validation records are reported as supplied, not presented as new runs. During preparation, all twenty constructed reference points were rechecked with the collection's double-precision evaluator, and all 31 entries in its SHA256 manifest matched. No live Patodia engine benchmark was executed for this chapter. Industry examples and teaching plans below are proposed activities rather than claims of adoption or validated industrial performance.

25 1 Benchmarks

25 1 1 Begin with a question and an independent reference

A useful benchmark asks a measurable question before execution. Can the solver recover a known finite root set? How many valid distinct candidates appear within a fixed allowance? How much does extended precision reduce independently checked residuals? What total elapsed time does an application require? These questions need different targets and should not be collapsed into one speed score.

Start with small problems whose answers can be derived independently. Such cases detect wrong variable ordering, lost constraints and overly loose acceptance before the experiment becomes difficult to interpret. Preserve exact equations, bounds, variable types and units. A change to a bound or a discrete declaration changes the mathematical question, even when the displayed equations look similar.

Class	Test and domain	Known answer
Polynomial	$x^2 = 4$ with $-3 \le x \le 3$	$x = -2$ and 2
Coupled polynomial	$x+y=7$ and $xy=12$ with x,y in $[0,7]$	(3,4) and (4,3)
Exponential	$\exp(x)=2$ with x in $[0,2]$	$x = \ln(2)$
Trigonometric	$\sin(x)=0$ with x in $[0,7]$	$x = 0, \pi$ and 2π
Integer	$a+b=7$ and $ab=12$ with nonnegative integers	(3,4) and (4,3)
Mixed integer and real	n integer; $y=n/2$; $y^2=1$; n in $[-3,3]$	$(n,y) = (-2,-1)$ and (2,1)
No real root	$x^2+1=0$ over the real numbers	None, since x^2+1 is positive
Repeated root	$(x-1)^2=0$ on $[0,2]$	One distinct root $x=1$

The mixed example can be entered exactly as follows. It tests whether the host and solver preserve integrality and declared variable order, rather than merely accepting a real relaxation.

```
var n,y
int n
y=n/2
y^2=1
bound n=[-3,3]
bound y=[-2,2]
```

If a complete reference set has N distinct admissible roots, coverage is M/N where M is the number of distinct reference roots matched by verified candidates. Matching needs a declared coordinate scale and tolerance, preferably with one-to-one assignment. An unmatched candidate should be investigated as a possible extra root, a reference error or a numerical false acceptance. Counting output rows alone does not establish coverage.

25 1 2 Understand what the twenty system collection contains

The supplied collection has twelve continuous systems and eight systems mixing continuous variables with general integers or binary choices. Three include binary variables, one of them together with general integers. Model sizes range from eight to sixteen variables. They contain polynomial, trigonometric, exponential, logarithmic, rational, radical and saturating nonlinearities.

These are constructed, dimensionless numerical challenges. Titles such as resonance mesh and feedback network describe mathematical structure; they do not establish that the equations represent calibrated equipment. A planted reference point is a verification anchor, not a full solution catalogue. Some models may have many additional roots or nonisolated branches. The archive makes no independently ranked difficulty or general superiority claim.

The following table combines the archive catalogue and its primary seed-42 validation record. D and E are the numbers of returned rows in normal and extended modes under a two-second allowance and a 100-solution cap. They are observed search counts, not certified totals. The original filenames beginning with the same two-digit identifiers locate each model.

ID	System	Vars	Discrete names	D	E
01	Cubic feedback ring	8	None	100	100
02	Oscillatory phase lattice	10	None	40	40
03	Exponential response loop	8	None	13	12
04	Logarithmic response mesh	8	None	50	50
05	Saturating feedback network	12	None	21	21
06	Rational interaction ring	10	None	21	21
07	Quartic coupled states	8	None	100	100
08	Stiff scaled response chain	10	None	100	100
09	Radical coupled amplitudes	8	None	90	85

ID	System	Vars	Discrete names	D	E
10	Exponential phase interaction	12	None	28	27
11	High-frequency resonance mesh	14	None	100	95
12	Dense polynomial-transcendental network	16	None	74	68
13	Integer stage feedback	8	n1,n2	1	1
14	Integer selected phase network	8	n1,n2	24	23
15	Binary switched exponential network	8	b1,b2	1	1
16	Integer configured saturation network	11	n1,n2,n3	87	82
17	Binary switched rational network	8	b1,b2	100	100
18	Integer selected quartic network	10	n1,n2	79	75
19	Integer and binary configured cubic network	12	n1,n2,b1,b2	85	85
20	Integer selected exponential network	11	n1,n2,n3	57	57

For the primary record, the search used seed 42, 30000 starts, 300 iterations, integer-neighbors 2 and a 100-point cap. The archive identifies a native Linux engine build; it is not a Windows GUI launch test. The reported two seconds is a computational allowance, not a guarantee of exactly two seconds of process wall time. Reaching 100 means that the configured cap was reached.

25 1 3 Reproduce a continuous reference model

Model 01 is an eight-variable cubic feedback ring. The following input is copied from the archive with descriptive comments omitted. Retain all equations and bounds when comparing runs.

```
var x1, x2, x3, x4, x5, x6, x7, x8
bound x1=[-2,4]
bound x2=[-2,4]
bound x3=[-2,4]
bound x4=[-2,4]
bound x5=[-2,4]
bound x6=[-2,4]
bound x7=[-2,4]
bound x8=[-2,4]
(x1-0.4)^3-1.3*(x1-0.4)+0.18*(x2-0.7)^2+0.12*(x2-0.7)*(x3-1.1)=0
(x2-0.7)^3-1.3*(x2-0.7)+0.18*(x3-1.1)^2+0.12*(x3-1.1)*(x4-1.4)=0
(x3-1.1)^3-1.3*(x3-1.1)+0.18*(x4-1.4)^2+0.12*(x4-1.4)*(x5-1.8)=0
(x4-1.4)^3-1.3*(x4-1.4)+0.18*(x5-1.8)^2+0.12*(x5-1.8)*(x6-0.4)=0
(x5-1.8)^3-1.3*(x5-1.8)+0.18*(x6-0.4)^2+0.12*(x6-0.4)*(x7-0.7)=0
(x6-0.4)^3-1.3*(x6-0.4)+0.18*(x7-0.7)^2+0.12*(x7-0.7)*(x8-1.1)=0
(x7-0.7)^3-1.3*(x7-0.7)+0.18*(x8-1.1)^2+0.12*(x8-1.1)*(x1-0.4)=0
(x8-1.1)^3-1.3*(x8-1.1)+0.18*(x1-0.4)^2+0.12*(x1-0.4)*(x2-0.7)=0
```

Variable order	Constructed reference vector
x1 through x8	(0.4, 0.7, 1.1, 1.4, 1.8, 0.4, 0.7, 1.1)

At the reference vector every shifted coordinate is zero, so every residual vanishes algebraically. This gives a transparent feasibility check. It does not make the system linear, eliminate other branches or establish that automatic search must return this particular vector. The historical primary run returned 100 rows in each precision mode, reaching its cap.

A controlled extension can change the coefficient −1.3 to −α while preserving the reference construction. With all shifted variables equal to z, the equations reduce to $z^3 + 0.30z^2 - \alpha z = 0$. At α = 1.3 this yields z = 0 and z = [−0.30 ± √5.29]/2, or z = 1 and z = −1.3. Each common shift remains inside the supplied bounds for this reference vector, giving three analytically known ring solutions. They are a known subset rather than the complete eight-variable root set.

The common-shift construction provides a better coverage target than the planted point alone: students can check all three vectors x = reference + z. It also reveals how structural analysis can strengthen a synthetic benchmark. Any additional vectors returned by the solver should be retained and checked independently.

25 1 4 Separate automatic discovery from fixed reference checks

The archive contains forty primary automatic searches: twenty models in two modes. Each returned at least one row. It also records twenty-four additional automatic searches for replacement models 14, 16, 17, 18, 19 and 20 using seeds 38 and 43 in both modes. Together these are sixty-four recorded automatic searches; the thirty-six-run replacement subset includes those models' twelve primary seed-42 runs.

A separate file records forty checks with every variable temporarily fixed to its reference value. Those tests assess feasibility and parser/domain consistency, not the performance of discovering an unknown vector. They must not be added to an automatic-search success count without identifying their different purpose. The twenty local reference rechecks performed for this chapter did not invoke the solver.

Evidence in the archive	What it supports	What it does not establish
40 primary searches and 24 additional searches	Reported nonempty searches under the recorded settings	Complete coverage or cross-platform speed
40 fixed-bound reference checks	Reported feasibility of reference vectors	Automatic discovery performance
Double re-evaluation of returned rows	Reported residual and domain consistency at checker precision	Binary128 coordinate-digit accuracy
Reference points and model catalogue	Reproducible anchors and known declarations	Unique roots or complete solution lists

The package evaluator checked returned candidates with ordinary Python double functions and used an absolute residual test below 10^{-6}, together with bounds, integrality and inequality filters. The largest independent residual stored across the forty primary records is approximately 9.89×10^{-10}. This is a check of recorded numerical consistency. The archived summary files do not include every returned coordinate vector, so this chapter could not independently repeat that returned-row audit. The binary128 residual figures are engine-reported measurements; a double checker cannot confirm their full digit accuracy.

For Model 14, normal-mode row counts for seeds 38, 42 and 43 were 19, 24 and 23, while extended-mode counts were 18, 23 and 23. For Model 17 every listed run reached 100, so the cap masks any larger count. These observations teach two different limitations: variability in finite discovery, and truncation by a reporting limit.

25 1 5 Record the experiment and distinguish the clocks

Save model bytes, exact executable identity and hash, command tokens, arithmetic mode, requested tolerance, all bounds, merge setting, seed, starts, iterations, cap and stopping reason. Record CPU, operating system, memory and thread settings where relevant. The Chapter 24 wrapper supplies a starting point for preserving raw streams and result JSON. Computational reproducibility depends on retaining data, methods and environment [1].

Use the minimum non-null cumulative found_seconds value for first accepted point time, and the maximum for the final accepted point represented in the result. Measure total process wall time outside the executable. A search may continue after accepting its final reported point. Thus millisecond point discovery and a two-second process run can both be correct measurements of different events.

Metric	Definition	Interpretation
Valid-row count	Verified distinct admissible candidates	Search yield within a stated budget
Known-set coverage	Matched reference roots / complete known count	Meaningful only with a complete reference set
First accepted point	Minimum non-null found_seconds	Discovery latency, excluding unspecified host costs
Final represented point	Maximum non-null found_seconds	Latest acceptance timestamp among retained rows
Process wall time	External start-to-completion measurement	Includes launch I/O and continued search
Residual and coordinate error	Original residuals and independent reference comparison	Separate equation consistency from coordinate accuracy

A schematic record with Found seconds values 0.02, 0.40 and 0.07 has first-point time 0.02 s and final represented acceptance time 0.40 s. Adding them to obtain 0.49 s has no cumulative-time meaning. A timeout without final JSON should be recorded as interrupted, not assigned a fabricated final-root time.

25 1 6 Run matched comparisons and preserve unsuccessful cases

To reproduce one archive configuration, work inside its extracted folder and use commands such as the following. Paths must match the installation. Normal and extended configurations have different default residual targets, so state whether the comparison concerns equal targets, attainable residual quality or discovery yield.

```
equation_solver.exe --file 01_cubic_feedback_ring.txt ^
  --seconds 2 --starts 30000 --iterations 300 ^
  --max-solutions 100 --integer-neighbors 2 --seed 42 ^
  --tol 1e-9 --json

equation_solver.exe --file 01_cubic_feedback_ring.txt ^
  --seconds 2 --starts 30000 --iterations 300 ^
  --max-solutions 100 --integer-neighbors 2 --seed 42 ^
  --extend --tol 1e-24 --json
```

The supplied validator can be used without an executable to recheck reference points. With an executable, --quick limits it to the requested search allowance rather than its normal retry sequence. Choose a new output filename so historical validation records remain intact.

```
python validate.py --output reference_local.json
python validate.py --exe C:/Patodia/equation_solver.exe ^
  --seconds 2 --quick --output validation_local.json
```

The current validator hard-codes seed 42 and uses a double independent evaluator. It is a useful starting harness, not a complete cross-solver study. Use a documented separate harness to vary seeds, enforce process-tree deadlines where required, and preserve the full exported rows and failure outcomes. Do not silently retry until a case succeeds and then report only the successful attempt.

For a classroom study, three seeds and several repeated executions are a manageable initial design. For a paper, choose the number of repetitions around the variability and claim being investigated. Report distributions, medians and spread, together with timeouts and failed cases. A run returning no point within its budget is a computational outcome, not an infeasibility proof.

When comparing other solvers, describe whether they require initial points and how the host supplies them. Count that initialization and any multistart wrapper in the documented method. Use equivalent equations, bounds, variable types, targets and resource policies. Separate mathematical reformulation from algorithmic differences; do not describe unrelated optimization output as a root-solving comparison.

25 1 7 Benchmark exercises and worked answers

Problem 1. A known finite system has twenty roots and a verified run matches fifteen distinct reference roots. What is its coverage? Does finding a sixteenth row necessarily raise coverage?

Answer 1. Coverage is 15/20 = 75 percent. A new row raises it only if that row is a distinct, valid match to a previously unmatched reference root. A duplicate or a false acceptance does not. An apparently extra valid root also requires checking whether the reference count was correct.

Problem 2. A synthetic collection reports 100 rows with --max-solutions 100. Can its exact root count be written as 100?

Answer 2. No. The run reached its cap. The model may have more roots or even nonisolated branches; an independent count or validated isolation result is needed for a total.

Problem 3. Why must the forty fixed-reference checks be kept separate from automatic searches?

Answer 3. Fixing every coordinate supplies the intended point as the entire search domain. Passing checks the point’s feasibility and the interface, while automatic discovery must locate admissible points in the original domain. Their success rates answer different questions.

25 2 Teaching

25 2 1 Use the solver to make mathematical reasoning visible

At university level, the solver can support lectures, supervised laboratories, coursework and research preparation. Its ability to explore alternative real solutions without a student-supplied starting vector can help students see that a model may admit several configurations. Require students to predict the answer structure before running it and explain each returned branch afterwards.

A useful classroom sequence is prediction, formulation, execution, verification and interpretation. First estimate signs, units and likely root count. Then declare unknowns and constraints, run a recorded calculation, substitute into the original relationships and discuss what each root means. A polished screenshot alone does not show that the student understood the model.

Begin with problems where students can check the complete answer by algebra. Move to coupled or realistic models after those habits are established. Preserve unsuccessful runs and ask students to identify whether the

difficulty is syntax, model inconsistency, domain restriction, search coverage or numerical sensitivity. These are different learning outcomes.

25 2 2 Plan a six laboratory university module

Lab	Topic and task	Required evidence
1	Linear and polynomial systems; solve circle and line intersections	Hand derivation full root list and substitutions
2	Integer modelling; distinguish individual items from bundles	Units declarations and finite-enumeration oracle
3	Geometry and engineering; compare link positions or heat balances	Physical domain and interpretation of each root
4	Numerical reliability; cancellation repeated roots and close roots	Independent residuals and coordinate-error reasoning
5	Benchmark design; compare selected archive cases across budgets	Prespecified settings raw results and failure records
6	Original mini-project; build and defend one model	Assumptions reference checks and reproducible submission

The module can sit within numerical methods, engineering mathematics, computational science or a discipline-specific modelling course. Adapt prerequisites: calculus and linear algebra support sensitivity analysis, while chemistry or electrical courses supply their own constitutive laws. The solver should reduce repetitive numerical work while leaving derivation, units and scientific interpretation assessable.

An instructor can demonstrate the desktop interface, then use the Chapter 24 CLI workflow for repeated experiments. Students need not begin with programming, but a small automation exercise helps them retain exact inputs and compare runs consistently. Use supplied examples and documented interfaces rather than requiring access to proprietary engine internals.

25 2 3 Teach multiple solutions with a complete reference

Consider two length-five links joining fixed points A = (0,0) and B = (6,0) to a free point (x,y). Ask students to predict the number of assembly positions before solving. The model is small enough to verify exactly.

```
var x,y
x^2+y^2=25
(x-6)^2+y^2=25
bound x=[0,6]
bound y=[-5,5]
```

Subtracting the equations gives x = 3 and then $y^2 = 16$. Both (3,4) and (3,−4) satisfy the distances. Students can plot the two circles, label the reflected positions and explain why a negative y coordinate is not an invalid link length. A request for an upper-only assembly changes the domain to y in [0,5], retaining one solution.

For assessment, ask what happens when the pivot spacing becomes 10 or exceeds 10. At 10 the circles are tangent at (5,0); above 10 they cannot intersect. This connects a numerical outcome to a geometric existence argument and distinguishes two roots, a repeated geometric intersection and no root.

25 2 4 Teach variable types and scientific constraints

Use Model 15 from the archive to discuss binary choices alongside continuous responses. Its declared order is x1 through x6, b1, b2, with binary b1 and b2. The planted reference is (0.4,0.7,1.1,1.4,1.8,0.4,0,1). Every shifted quantity in its equations is zero there. The inequality filters also hold: $x1+x2=1.1\leq 5$ and $x1^2+x2^2=0.65\geq 0.01$.

Ask students to list the four binary assignments before solving each continuous subproblem. This finite discrete enumeration is a useful comparison with a mixed search, but it does not prove that all continuous roots for an assignment were found. Removing binary declarations creates a real relaxation and changes the feasible set; a fractional switch is not an acceptable discrete solution.

In chemistry or materials teaching, replace abstract bounds by justified concentration or composition restrictions. In mechanical teaching, use joint limits and nonnegative cable tensions. Students should explain why a restriction is part of the model rather than add it only because it removes an inconvenient answer. Residual correctness and scientific admissibility are separate checks.

25 2 5 Assess reasoning and reproducibility

Assessment component	Suggested marks	What demonstrates learning
Formulation and assumptions	25	Correct unknowns units equations and stated simplifications
Domains and variable types	15	Justified bounds signs integrality and function domains
Independent verification	25	Original substitutions complete-set argument where available
Numerical experiment	20	Saved settings precision raw exports and honest failure handling
Interpretation and communication	15	Meaning of branches limitations and clear reproducible report

The percentages form one proposed rubric, not a university requirement. Award credit for identifying a correct no-solution case and for explaining why a run is inconclusive. Avoid rewarding only the fastest time or longest root list, since those criteria can encourage loose tolerances, altered bounds or unreported failures.

A submission can contain the equation text, a one-page model explanation, raw result files, a short verification script or worksheet, and a discussion of limitations. During a brief oral check, ask the student to change one parameter and predict the effect before rerunning. That reveals understanding beyond copied output.

25 2 6 Use equation solving within a PhD research method

For doctoral research, begin with a scientific or methodological question supported by a literature review. Equation solving can provide evidence for that question: locating alternative steady states, checking a new constitutive model, studying parameter identifiability or measuring numerical robustness. Running supplied benchmarks quickly is useful evaluation work, but by itself does not establish the original contribution expected of a PhD.

A research workflow can proceed from a derived model and stated assumptions to independent reference cases, controlled parameter studies, candidate verification and comparison with observations or an established implementation. Keep exploratory runs separate from the final evaluation set. A model chosen after many trials because it produces favorable results should not be described as a prespecified test.

For a steady system $F(x,\lambda)=0$, nonsingular local sensitivity satisfies $J(x,\lambda)\, dx/d\lambda = -\partial F/\partial\lambda$, where J is the Jacobian with respect to x. Derive or evaluate these terms externally when needed. Near a singular J, sensitivity can become large and branches may merge. A missing row at the next parameter value is not enough to establish that a branch ceased to exist.

For research, the cumulative engine provides supported IVP integration, regular BVP collocation and selected PDE/integral modes as developed in Chapters 16 to 22. Rigorous branch tracking, stability analysis, general PDE systems and uncertainty propagation remain separate capabilities requiring appropriate analysis and tools.

Noisy experimental data also require an estimation framework. Forcing every measurement into an exact equality can create artificial inconsistency. Least-squares fitting, likelihood analysis or other statistical methods need a defined objective and error model. Derived stationarity equations can generate candidates, but they still need classification and comparison; a stationary solution is not automatically a global optimum.

25 2 7 Define research projects with testable contributions

Project direction	Research question	Evidence beyond solver output
Multiple steady states	Which parameter changes create or remove admissible equilibria?	Branch analysis stability analysis and experimental or reference validation
Formulation and scaling	Which transformations improve reliability while preserving roots?	Paired models conditioning analysis and held-out cases
Precision and identifiability	Which digits or parameters are supported by equations and data?	Independent high precision checks perturbation study and uncertainty
Mixed discrete configuration	How do switch or stage choices change feasible operating points?	Enumeration references per-configuration checks and a separate objective if optimizing
Benchmark methodology	How robust is discovery across families budgets and seeds?	Complete small references diverse cases transparent harness and failure accounting

A thesis chapter should state the contribution, the evidence and its scope separately. A newly observed candidate can motivate a conjecture; a complete-root statement needs stronger analysis. A new physical interpretation needs domain evidence, not just numerical novelty. A negative result or a documented model failure can be scientifically valuable when the methodology is sound.

For a research archive, retain equation files, raw decimal exports, checker code, source data where shareable, software hashes, computational environment and the exact mapping from runs to published tables. Describe acceptable numerical variation instead of demanding identical discovery order when that is irrelevant. These practices follow the broader reproducibility principles in the National Academies report [1].

25 2 8 Teaching and research exercises with worked answers

Problem 4. A student excludes $y<0$ in the two-link example because negative values look unphysical. What should the instructor ask?

Answer 4. Ask whether y is a signed coordinate or a length, and whether the physical assembly excludes the lower half-plane. Signed coordinates may be negative. An upper-only restriction is valid only when it is part of the stated assembly question.

Problem 5. A PhD study finds one fitted parameter vector with residual 10^{-25}. Does this establish identifiability and experimental accuracy?

Answer 5. No. Identifiability asks whether other admissible parameter values can produce the same observations. Experimental accuracy depends on the measurements and model assumptions. A small residual addresses consistency of one numerical vector with the entered equations.

Problem 6. A researcher changes equation scale, bounds, precision and seed at once and observes more rows. Can the improvement be attributed to scaling?

Answer 6. No. Several causes changed together. Use matched comparisons or a prespecified factorial experiment, retain original residual checks and report the relevant interactions. Keep a separate dataset for evaluating any choices tuned during exploration.

25 3 Further Exploration

25 3 1 Translate an industrial question into a bounded pilot

An industry application begins with a specific unanswered calculation, not a request to apply a solver everywhere. Identify the owner of the model, the unknowns, required data, equations, admissible ranges and the decision that the answer could support. Choose a small subsystem whose result can be checked against a hand calculation, measurement or trusted simulation.

Equation solvers are especially useful when an operating point must satisfy several coupled relationships or when alternative feasible configurations matter. A general solver can examine a reduced algebraic subsystem without supplying all the thermodynamic, material or equipment data itself. Established process frameworks such as IDAES combine equation-oriented modelling with libraries and optimization capabilities [2]; this illustrates the broader ecosystem in which a focused equation calculation may sit.

Treat the following application areas as pilot candidates. Their value depends on suitable domain equations, calibrated coefficients and independently verified results. The synthetic benchmark names are not evidence that those industries have already validated the corresponding models.

Industry or discipline	Possible equation task	Data and independent check
Chemical processing and materials	Steady mass/energy balances or equilibrium speciation	Property and reaction data; conservation and trusted reference cases
Electrical and electronics	Nonlinear DC device operating points	Device law parameters; circuit balances and measurements
Energy and thermal equipment	Heat-loss partition surface temperature or exchanger balances	Geometry and properties; energy balance and test data
Water and utilities	Small pump/pipe operating points and branch flows	Pump curves and loss coefficients; hydraulic checks
Manufacturing and robotics	Link positions calibration constraints and discrete configurations	Dimensions joint limits; geometric and measurement checks
Biotechnology and laboratory analysis	Steady balances or exact synthetic parameter recovery	Units domain and observation model; independent assay or reference
Economics and business modelling	Equilibrium break-even or input-output consistency	Defined coefficients and period; accounting and sensitivity checks
Environmental modelling	Steady mixing partition or simplified mass balances	Source data and transport assumptions; conservation and observations

25 3 2 Explore a nonlinear electrical operating point

A simple device model illustrates a suitable electronics pilot. A 5 V source drives a 1 kΩ resistor and an idealized diode. Let V be diode voltage in volts and i current in milliamperes. Assign saturation current 10^{-9} mA and thermal/ideality voltage scale 0.05 V. These teaching parameters are not a fitted commercial component specification.

```
var V,i
V+i=5
i=1e-9*(exp(V/0.05)-1)
bound V=[0,5]
bound i=[0,5]
```

The reference is approximately V = 1.104158773 V and i = 3.895841227 mA. Eliminating i gives a strictly increasing function $V + 10^{-9}[\exp(V/0.05)-1]-5$, so the stated domain contains one root. Check both voltage balance and the assigned diode law. In milliwatts the power balance is $5i = i^2 + Vi$.

An engineering extension can add temperature dependence or measured series resistance, then compare predicted operating points with observations. The equation solution does not by itself establish device rating, thermal reliability or transient behavior. Those require corresponding models and data. Begin with the known simplified limit so that each extension has an interpretable check.

25 3 3 Explore water energy and thermal balances

Chapter 11 developed parallel-pipe and pump/system examples. A water-utility pilot could begin with two branches having a shared head loss and measured total flow. An energy pilot could solve a steady heat balance and partition convection and radiation. Multiple admissible operating points should be examined before selecting an operating branch.

For illustration, assign pump head $H = 50-0.5Q^2$ and required system head $H = 10+0.3Q^2$, with Q in L/s and H in metres. The numerical coefficients include those flow units. The positive intersection is a compact independent reference.

```
var Q,H
H=50-0.5*Q^2
H=10+0.3*Q^2
bound Q=[0,10]
bound H=[0,50]
```

Subtracting gives $Q^2 = 50$, so $Q = \sqrt{50} \approx 7.071067812$ L/s and H = 25 m. Check both original curves at that flow. A larger model may require signed head losses for reverse flow, changing friction factors, valve states and storage dynamics. The small algebraic intersection is only one part of that system.

EPA's EPANET models hydraulic and water-quality behavior over time in pressurized networks containing pipes, pumps, valves, tanks and reservoirs [3]. A focused Patodia calculation can be compared with a small extracted hydraulic case; it should not be presented as an equivalent replacement for those network and time-dependent capabilities. Agreement on the shared equations is a useful pilot criterion.

25 3 4 Explore manufacturing and discrete configuration choices

Manufacturing models can combine whole batch counts, material balances and continuous process settings. Begin by specifying whether a resource number is an exact-use requirement or a capacity. Equality and inequality versions can have very different feasible sets. A feasibility model identifies permitted choices; selecting the best choice needs an explicit objective and an appropriate optimization method.

Consider an assigned workshop model: a and b count whole batches. Each A batch uses three machine-hours and two labour-hours; each B batch uses two and four. Require exactly 32 machine-hours and 40 labour-hours.

```
var a,b
int a,b
3*a+2*b=32
2*a+4*b=40
bound a=[0,10]
bound b=[0,10]
```

The unique solution is a = 6 and b = 7, verified by 18+14=32 and 12+28=40. Changing machine use to 30 yields the unique real solution (5,7.5), so the exact-use integer version is infeasible. Rounding the fractional count changes the resource balances and is not a valid repair.

For robotics, retain the multiple assembly positions from the teaching example and add actual joint limits and collision checks. A position root does not establish a collision-free motion or an acceptable load path. Those are additional tasks. A pilot can test calibration consistency or enumerate alternative endpoint positions before a larger motion-planning workflow considers them.

25 3 5 Explore business laboratory and environmental models

Business applications can use equations for internally consistent demand/supply assumptions, break-even output and sector balances. Replace an assumed demand intercept by a sequence of values and track how the admissible equilibrium changes. Record the monetary unit and period. Such a sensitivity exercise describes the chosen model, and becomes an empirical forecasting exercise only after the model and data have been assessed appropriately.

A laboratory application may recover parameters from exact synthetic observations to validate the computational pipeline, then move to a separate statistical fit for noisy observations. Identifiability analysis should precede claims that many displayed parameter digits are meaningful. Different parameter combinations can produce the same observations even when every residual is small.

For environmental balances, a useful starting case is steady conservative mixing. Let q1 = 2 L/s at concentration c1 = 10 mg/L and q2 = 3 L/s at c2 = 30 mg/L. With complete mixing and no reaction or loss, outlet flow q and concentration c satisfy the following equations.

```
var q,c
q=2+3
q*c=2*10+3*30
bound q=[0.001,10]
bound c=[0,40]
```

The reference is q = 5 L/s and c = 22 mg/L. The mass rate is 110 mg/s on each side, and the outlet concentration lies between the inlet values. This elementary reference can anchor an extension with reaction, partitioning or additional streams. Each added process requires its own law and parameters; a numerical root cannot supply them implicitly.

25 3 6 Build a pilot from evidence to a useful decision

A practical pilot has a small, written acceptance target. For a pump case, it might require reproducing the simplified intersection, matching an established hydraulic calculation for agreed operating conditions and explaining any discrepancies against measured data. For a device case, it might require matching a documented operating-point dataset within a stated measurement uncertainty. Choose thresholds before reviewing the new results.

Stage	Concrete deliverable	Decision supported
Define	One subsystem unknown list data and domain	Is this an equation-solving task?
Establish baseline	Hand or trusted-tool reference on a simple case	Can the formulation and interface be checked?
Extend	One documented physical effect or configuration at a time	Does the richer model add useful explanatory power?
Validate	Residuals domain checks sensitivity and data comparison	Are the results adequate for the stated pilot purpose?
Integrate	Versioned model and automated evidence record	Can the calculation be repeated reliably?
Review	Domain interpretation and unresolved limitations	What further work is justified?

Track the cost of preparing and maintaining the model as well as solve time. A millisecond root calculation may still sit inside a workflow dominated by data cleaning, property evaluation or review. Measure the user's complete task when evaluating practical value. A faster reduced solve is useful when it removes an actual bottleneck or enables a previously impractical exploration.

Use Chapter 24's CLI or authorized DLL interface to embed the calculation in an existing tool. Preserve full decimal strings, the run snapshot and failure states. Keep domain-specific data providers and statistical or optimization components explicit. A clear interface makes it possible to replace or improve one component without silently changing the mathematical problem.

25 3 7 Plan further mathematical and computational exploration

Extend the benchmark collection along deliberate axes: variable count, equation density, coordinate scale, branch separation, derivative singularity and discrete choices. Change one axis at a time when isolating an effect, and retain small analytically solved cases. Keep newly constructed cases separate from held-out application models used to assess generality.

For parameter studies, record all verified branches rather than only the first root or the one nearest a preferred value. Study how a domain restriction changes admissibility, and distinguish that from a search failing to rediscover a branch. If a proof is required, use the applicable supported certification path or an independent mathematical argument over the stated domain; a longer heuristic run alone does not supply the proof.

For precision studies, compare coordinate errors against independent references as well as residuals. Include cancellation, repeated roots and nearly dependent equations from Chapter 23. For integration studies, vary host concurrency and model size while recording complete task time and preserving numerical semantics. These experiments develop reusable knowledge about where the solver is effective and where another method or a richer model is needed.

25 3 8 Further exploration exercises and worked answers

Problem 7. A pump/system calculation returns a valid intersection. Does that establish motor power, acceptable suction conditions and dynamic stability?

Answer 7. No. Power needs the relevant density, gravity and efficiency relationships. Suction conditions and stability require their own data and models. The intersection establishes equality of the two assigned head curves at the returned flow.

Problem 8. In the workshop example, exact machine use becomes 30 while labour use remains 40. Can b = 7.5 be rounded to 8?

Answer 8. Not while retaining the original exact-use equations. With a = 5, b = 8 uses 31 machine-hours and 42 labour-hours. The integer model has no solution because its unique real vector is fractional. A capacity or rescheduling question must be formulated explicitly as a new model.

Problem 9. A mixing pilot gives outlet concentration 35 mg/L for the stated 10 and 30 mg/L conservative inlets. What should be checked first?

Answer 9. Check flow units, concentration units, the mass balance and the assumptions of no reaction or extra source. With positive flows and conservative complete mixing, the concentration is a weighted average and cannot exceed both inlet concentrations. The stated model gives 22 mg/L.

25 3 9 Sources and a continuing programme of work

The supplied benchmark archive provides the numbered models, models.json, VALIDATION.md, validation.json, repeat_validation.json, fixed_reference_checks.json, validate.py and SHA256SUMS.txt used in the first section. Counts and settings attributed to those records are historical observations. The separate Patodia Equation Developer Guide dated 28 October 2026 supplies the command-line and integration contracts discussed in Chapters 13 and 14.

The external sources below support reproducible research and examples of established modelling ecosystems. They do not document an endorsement, adoption or performance validation of Patodia Equation Solver. The proposed laboratory sequence, assessment rubric, research directions and industrial pilots are developed for this chapter.

[1] National Academies of Sciences Engineering and Medicine Reproducibility and Replicability in Science Chapter 4 Reproducibility
https://www.nationalacademies.org/read/25303/chapter/7

[2] Institute for the Design of Advanced Energy Systems Overview
https://idaes.org/overview/

[3] United States Environmental Protection Agency EPANET
https://www.epa.gov/water-research/epanet

Continue from one independently checked case to a broader question. In benchmarking, preserve the task and report what was measured. In teaching, require explanation and verification. In research and industry, connect the equation result to evidence about the model and its intended use. This progression makes further exploration useful, reproducible and open to correction.

25 4 Current edition workflow and scope

25 5 Extend the benchmark programme to numerical functions

Add analytic IVP trajectories, event times, a stiff forced solution, regular BVPs, singular compatibility cases, manufactured PDE fields and separable integral equations. Measure coordinate/sample error as well as residuals and work. Keep discretisation studies distinct from arithmetic-precision studies. Cross-compilation and portable tests do not establish Windows graphical acceptance.

For teaching, ask students to derive a reference, predict a domain and classify the returned evidence before running. For research, preserve model bytes, grids, tolerances, licence edition, method, runtime identity and all unsuccessful outcomes. The new chapters provide scoped laboratories rather than a claim to solve unrestricted PDEs or arbitrary singular equations.

Chapter 26 Mixed Boundary Conditions and Adaptive PDE Accuracy

A boundary condition is part of the mathematical model, not a setting for improving a plot. A prescribed value, an insulated edge and a convective surface express different physical statements. Version 3 makes these distinctions explicit and combines them with numerical accuracy targets. This chapter connects the boundary syntax with outward signs, conservation and grid convergence.

26 1 PDE boundary conditions and normal signs

Specify exactly one boundary directive per side. u sets a Dirichlet value; normal sets the outward derivative du/dn; robin=[alpha,beta,g] sets alpha*u+beta*du/dn=g.

```
boundary left u=0
boundary right normal=0
boundary top robin=[1,2,3+x]
```

Side	Outward derivative
Left	-u_x
Right	+u_x
Bottom	-u_y
Top	+u_y

Alpha and beta must be nonnegative and not both zero at evaluated points. Boundary expressions can use x,t for 1D time equations or x,y for 2D steady fields. They must meet the initial/corner compatibility checks.

Heat-flux convention

normal is a derivative, not automatically a physical heat flux. If outward conductive flux is q_n=-k*du/dn, a prescribed flux q_n=q0 means normal=-q0/k. For convection -k*du/dn=h*(u-u_ambient), write h*u+k*du/dn=h*u_ambient, using robin=[h,k,h*u_ambient]. Supply consistent units yourself.

ODE BVP y' means the coordinate derivative at that endpoint; PDE normal uses the outward sign. Insulation is normal=0. u=0 fixes temperature/displacement and is a different condition.

26 2 Derive the normal signs before writing the model

For a one-dimensional interval, the outward direction points left at its first endpoint and right at its last. Therefore the normal derivative is -u_x on the left and +u_x on the right. A field u=x has coordinate derivative +1 everywhere, but its outward boundary derivatives are -1 and +1. Entering normal=1 at both ends would prescribe a different problem. On a rectangle the bottom and top signs follow the same argument using u_y.

For a convective thermal boundary, outward conductive flux is -k*du/dn and outward exchange is h*(u-u_ambient). Equating them gives h*u+k*du/dn=h*u_ambient. This becomes robin=[h,k,h*u_ambient]. The coefficients in this form must meet the implementation nonnegativity restrictions. If a physical model produces another sign convention, derive an equivalent admissible form rather than changing signs by guesswork.

26 3 Work an insulated heat mode

```
pde
type=heat
u_t=u_xx
domain x=[0,1]
interval t=[0,0.1]
initial u=cos(pi*x)
boundary left normal=0
boundary right normal=0
grid x=17
grid t=41
exact solution=exp(-pi^2*t)*cos(pi*x)
```

Differentiate the reference exp(-pi^2*t)*cos(pi*x). Its time derivative is -pi^2 times itself, and its second spatial derivative is the same. Its first spatial derivative vanishes at x=0 and x=1, so both ends are insulated. The spatial mean is zero and remains zero in the continuous equation. This reference checks diffusion sign, initial data, boundary signs and time decay simultaneously.

At x=0 the field decays from +1; at x=1 it rises from -1 toward zero. A negative value is not an error in this signed manufactured field. In a physical temperature model one might add a positive baseline, with the corresponding exact solution and compatible boundaries adjusted consistently. The solver does not decide a physical temperature scale from the symbol u.

26 4 Time-dependent Robin data

```
pde
type=heat
u_t=u_xx+1
domain x=[0,1]
interval t=[0,0.1]
initial u=x
boundary left robin=[1,1,t-1]
boundary right normal=1
grid x=17
grid t=21
exact solution=x+t
```

The reference u=x+t has u_t=1 and u_xx=0, giving the source +1. On the left, u=t and du/dn=-1, so u+du/dn=t-1. On the right du/dn=+1. This is why the two boundary directives have different forms even though the coordinate slope is constant. The exact solution has no curvature truncation error; it is especially useful for testing boundary elimination and time-dependent data handling.

26 5 A mixed-boundary Poisson field

```
pde
type=poisson
u_xx+u_yy=4
domain x=[0,1]
domain y=[0,1]
boundary left normal=0
boundary right robin=[1,1,3+y^2]
boundary bottom u=x^2
boundary top u=x^2+1
grid x=9
grid y=9
exact solution=x^2+y^2
```

For u=x^2+y^2, the Laplacian is 4. At the left normal derivative is zero; at the right u+du/dn=(1+y^2)+2=3+y^2. Bottom and top values are x^2 and x^2+1. The outward flux balance also checks the source: total area integral is 4 and integrated normal derivatives sum to 4. The example combines value, derivative and Robin boundaries in one problem.

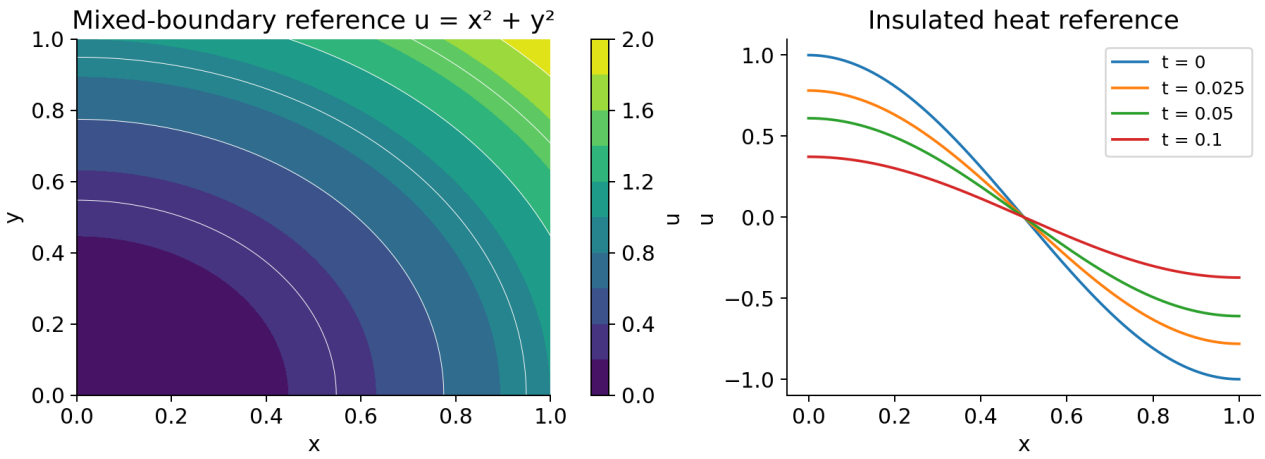


Figure 25.1 Analytical reference fields for the mixed Poisson and insulated heat examples. These plots are generated from the displayed formulas and are not Windows solver screenshots.

26 6 Pure Neumann fields and the mean value

```
pde
type=laplace
domain x=[0,1]
domain y=[0,1]
boundary left normal=-1
boundary right normal=1
boundary bottom normal=0
boundary top normal=0
mean value=0
grid x=9
grid y=9
exact solution=x-0.5
```

For this model the reference is x-0.5. Both x derivatives are +1, but outward derivatives are -1 on the left and +1 on the right. The mean is zero.

Two mathematical requirements

With all-normal conditions the PDE leaves an arbitrary additive constant. mean value fixes that constant. Source and boundary flux must also satisfy the continuous compatibility relation; for u_xx+u_yy=f it is integral(f over area)=integral(du/dn over boundary).

A separate composite-Simpson calculation assesses continuous compatibility. Incompatible data give incompatible_conditions; an unresolved assessment gives compatibility_not_resolved.

The finite-volume trapezoidal right-hand side is projected to zero balance. Its reported neumann_compatibility_correction is an O(h^2) uniform source adjustment for smooth compatible data; it prevents coarse quadrature imbalance alone from rejecting a valid problem.

Providing mean value does not repair incompatible flux/source data. Check outward signs and the continuous balance before adjusting numerical limits.

26 7 Accuracy targets and automatic refinement

```
absolute tolerance=1e-6
relative tolerance=1e-6
max refinements=4
max grid=801
max nodes=250000
refine=true
```

For each assessed returned value v, the estimated error must be no greater than absolute tolerance+relative tolerance*abs(v). A positive absolute tolerance is especially useful near zero. Defaults are 1e-3 for each PDE/eigen target and 1e-9 for each integral-equation target.

What is actually refined

PDEs double both active coordinates and compare finer solutions. Integral equations compare Richardson extrapolants from finer trapezoidal solves. Eigenproblems compare reported eigenvalues across grids. Eigenfunction errors are not certified by the eigenvalue target.

Returned coordinates remain those requested by grid directives; values are sampled from the finest computed solution. max grid and max nodes limit internal work. Time and cancellation budgets cover the full refinement process.

Outcome	Meaning
Target met	Estimated criterion passed under the method assumptions
accuracy_not_met	Budget/limits prevented acceptance; inspect retained samples
refine=false	Fixed-grid computation; accuracy_target_met is null

Do not paste these controls into ODE/BVP or quadrature models indiscriminately. Each mode has its own tolerance names, semantics and permitted budgets. No numerical target is a general proof of correct digits.

26 8 Convergence tables and optional exact references

```
exact solution=exp(x)
```

For scalar integral equations the reference uses x. For time PDEs use x,t; for steady fields use x,y. Coupled integral systems use exact 1=... through exact N=.... Eigen references have separate eigenvalue/eigenfunction keys.

Convergence / reference field	How to read it
Level and internal grid	Actual work, distinct from returned sample grid
Estimated error and target	Method estimate compared with requested tolerance
Observed order	Rate inferred from successive differences
Maximum absolute error	Largest sampled difference from a supplied reference
RMS error	Root-mean-square sampled reference difference
exact_* and error_* columns	Reference and signed numerical-minus-reference samples

Why exact examples matter

Most applications have no closed form. Known exact solutions still test signs, boundary conventions, convergence order and implementation changes. Manufactured solutions provide controlled references: choose a smooth function and derive the source and boundaries that it satisfies.

An incorrect supplied reference is reported as a disagreement; it does not force the numerical answer. Near roundoff or before asymptotic convergence, observed order can be absent or unreliable.

To teach raw-grid order, set refine=false and solve successive grids manually. Otherwise automatic refinement can make different requested grids produce similarly accurate returned values.

26 9 Derive the refinement divisor

Suppose a smooth method has U_h=u+C*h^p+smaller terms. Then U_h-U_(h/2) is approximately C*h^p*(1-2^-p). The finer error is approximately (U_h-U_(h/2))/(2^p-1). For p=2 the divisor is 3; for first-order upwind spatial error it is 1. The coarse error would instead contain the factor 2^p/(2^p-1). Confusing the two gives a systematically wrong interpretation of the same difference.

This expansion assumes a consistent limiting problem and an asymptotic regime. If boundary samples are reinterpreted between grids, the model itself has changed. If a thin layer is unresolved on every grid, apparently small differences need not establish accuracy. If the linear system is solved too loosely, iterative error can dominate the differences. Read the mesh and linear tolerances together.

26 10 Practice and worked explanations

Exercise 1. For u=2*x+3 on [0,1], what are the two normal derivatives?

Worked explanation. The coordinate derivative is 2. The left normal derivative is -2 and the right is +2. Constant offsets do not affect derivatives.

Exercise 2. Write convection for h=4, k=2 and ambient value 10.

Worked explanation. Use robin=[4,2,40], representing 4*u+2*du/dn=40. Check that these quantities use consistent units before nondimensionalisation.

Exercise 3. Why is mean value needed for an all-Neumann harmonic field?

Worked explanation. Adding any constant leaves all derivatives unchanged. A mean fixes that one-dimensional nullspace; it does not repair incompatible source/flux balance.

Exercise 4. Two second-order grid results differ by 0.0003. What is the asymptotic fine error estimate?

Worked explanation. Approximately 0.0001, from division by 3. This is an estimate under second-order convergence, not a certified bound.

Exercise 5. What does accuracy_target_met=null mean?

Worked explanation. In a fixed-grid run the target was not assessed. It does not mean zero error or that the requested tolerances were met.

26 Check mixed boundary convergence across jobs

Run the same pde model on progressively finer grids by substituting @n in grid directives. Keep its boundary definitions fixed and compare the reported error or residual along with the accepted field.

Chapter 27 Sturm Liouville Eigenvalues and Eigenfunctions

An eigenproblem asks which parameter values permit a nontrivial function satisfying a differential operator and homogeneous boundary conditions. A vibrating string, a diffusion mode and a simple quantum well can share this mathematical structure while assigning different physical meanings to the eigenvalue. Version 3 supports a regular self-adjoint class rather than arbitrary spectral problems.

27 1 Regular Sturm-Liouville eigenproblems

```
eigen
type=sturm_liouville
interval x=[0,1]
p=1
q=0
weight=1
boundary left u=0
boundary right u=0
modes=3
grid x=33
exact eigenvalue=(n*pi)^2
exact eigenfunction=sin(n*pi*x)
```

This represents -(p(x)*y')'+q(x)*y=lambda*weight(x)*y. Here the eigenvalues are (n*pi)^2 and eigenfunctions are proportional to sin(n*pi*x), for n=1,2,3. Exact references are optional.

Positive p and weight are required at evaluated points; q can have either sign. The interval must be finite and regular. Separated homogeneous Dirichlet, normal and supported nonnegative Robin boundaries are accepted.

The method uses a symmetric flux stiffness and lumped weight matrix, reduces to a symmetric tridiagonal problem, isolates eigenvalues using Sturm counts and constructs eigenvectors by shifted inverse iteration and orthogonalisation.

This is not an arbitrary eigenvalue solver: periodic, nonhomogeneous, singular-endpoint and general non-self-adjoint eigenproblems are excluded. Requested modes are the lowest modes, up to 20.

27 2 Derive the constant coefficient reference

For p=weight=1 and q=0, the equation is -y''=lambda*y. With y(0)=y(1)=0, lambda=0 gives a linear function forced to zero, and lambda<0 gives hyperbolic functions that cannot satisfy both endpoints nontrivially. For lambda=k^2>0, the left condition leaves y=A*sin(k*x). The right requires sin(k)=0, hence k=n*pi and lambda_n=(n*pi)^2 for n=1,2,... .

The zero function satisfies every homogeneous linear equation but is not an eigenfunction. Normalisation removes arbitrary amplitude from each nonzero computed mode. On [0,1], integral sin(n*pi*x)^2 dx=1/2, so the continuum L2-normalised sine has amplitude sqrt(2). The implementation uses its discrete weighted norm and may therefore have a slightly different finite-grid amplitude.

27 3 Eigenvalue accuracy and mode normalisation

Discrete eigenpair residuals are checked before acceptance. Automatic accuracy targets concern the reported eigenvalues. A small eigenvalue error does not certify every eigenfunction sample.

Normalisation and sign

Each mode has internal trapezoidal weighted-L2 norm one: the discrete weighted sum of y^2 is one. Its first significant sample is made positive. Consequently the reported mode can differ by an amplitude factor from a familiar textbook formula.

A supplied exact eigenfunction is normalised with the same internal weighted quadrature and sign-aligned before comparison. Mode sign itself is arbitrary mathematically.

Neumann zero mode

```
boundary left normal=0
boundary right normal=0
exact eigenvalue=((n-1)*pi)^2
exact eigenfunction=cos((n-1)*pi*x)
```

With p=weight=1, q=0 and interval [0,1], the constant zero-eigenvalue mode is n=1. The integer n in reference expressions is the one-based reported mode number, not universally the number of internal zeros.

Physical interpretation

For a normalised vibrating-string model lambda can correspond to squared angular frequency. The relation depends on how stiffness, density and length were represented. The solver supplies no automatic unit conversion or physical-frequency interpretation.

Close eigenvalues and underresolved modes need careful refinement. Plot eigenfunctions, inspect discrete residuals and increase appropriate grid budgets rather than relying only on the displayed eigenvalue digits.

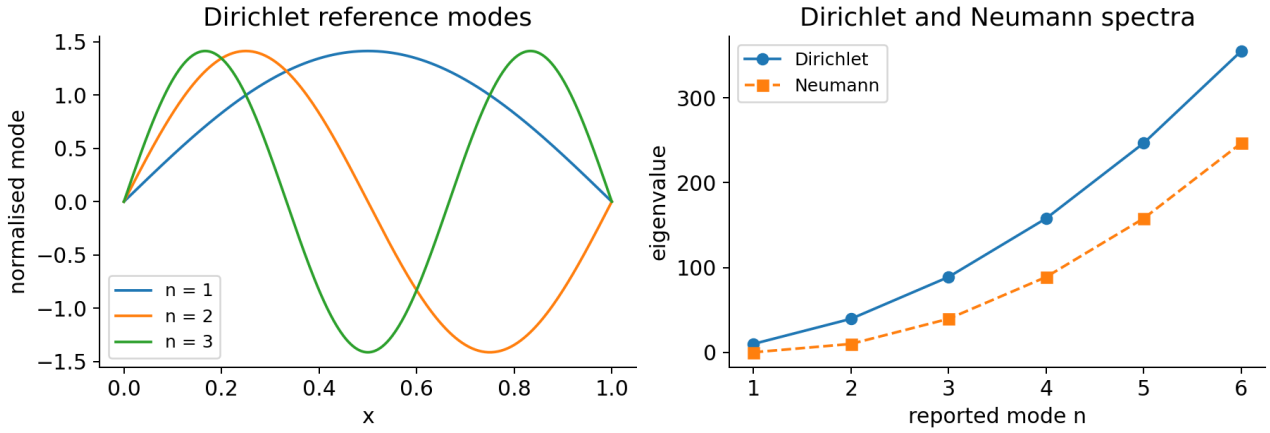


Figure 26.1 Continuum reference modes and eigenvalues. The right panel includes the Neumann zero mode and uses the reported one-based mode index.

27 4 Robin boundaries create a transcendental reference

```
eigen
type=sturm_liouville
interval x=[0,1]
p=1
q=0
weight=1
boundary left normal=0
boundary right robin=[1,1,0]
modes=3
grid x=33
```

Here y'(0)=0 and y(1)+y'(1)=0. For lambda=k^2>0, the left condition leaves y=A*cos(k*x); the right gives cos(k)-k*sin(k)=0. This scalar transcendental equation supplies an independent reference for positive eigenvalues. The equivalent k*tan(k)=1 has poles, so a sign change across a tangent pole must not be mistaken for an eigenvalue.

The lambda=0 case gives a constant after the left condition, then the Robin condition forces that constant to zero. Negative lambda yields cosh(mu)+mu*sinh(mu)>0 for mu>0, so there are no negative eigenvalues in this example. This branch analysis is useful because a root list for a transformed scalar equation alone might include spurious or missed cases.

27 5 Interpret p q and weight

The flux coefficient p and weight must be positive at evaluated points. q may change sign. For a stretched-string interpretation, p can describe tension and weight linear density after the model is scaled; lambda then relates to squared angular frequency. If p is multiplied by four with q=0 and unchanged weight, eigenvalues multiply by four and frequencies by two. If weight is multiplied by four instead, eigenvalues divide by four.

A negative eigenvalue is not automatically a numerical defect when q is sufficiently negative. Its physical interpretation depends on the original evolution equation. The solver reports the regular mathematical eigenproblem; it does not automatically classify stability, convert to Hertz or supply material data. Use the original dimensional equations for those conclusions.

27 6 Compare modes across grids and parameters

Order the lowest reported eigenvalues and compare the same mode number under refinement. For nearly repeated modes in a more general setting, vector orientation can be sensitive even when an invariant subspace is stable. This one-dimensional separated regular problem has a special spectral structure, but users should still inspect mode shape, residuals and grid resolution. A mode with many oscillations needs adequate spatial sampling.

An eigenvalue error target does not certify eigenfunction derivatives or fluxes. Comparing a normalised exact eigenfunction checks sampled shape; it does not establish uniform derivative convergence between nodes. Report which quantity was assessed. Avoid presenting a normalisation convention as a physical displacement amplitude.

27 7 Practice and worked explanations

Exercise 1. For interval [0,L] with two zero-value ends, what are the reference eigenvalues?

Worked explanation. With p=weight=1 and q=0 they are (n*pi/L)^2. Differentiate sin(n*pi*x/L) twice to verify the operator and check both endpoints.

Exercise 2. What is the first eigenvalue with two Neumann ends and q=0?

Worked explanation. Zero, with a constant eigenfunction. In the reference syntax this is reported mode n=1, so use n-1 in the cosine formula.

Exercise 3. Why can numerical and textbook modes have different signs?

Worked explanation. Any nonzero scalar multiple of an eigenfunction solves the homogeneous problem. Compare after consistent normalisation and sign alignment.

Exercise 4. Does a small eigenpair residual guarantee continuum accuracy?

Worked explanation. No. It tests the discrete matrix equation. Grid convergence and, where possible, independent continuum references assess discretisation error.

Exercise 5. Can a singular endpoint or periodic eigenproblem be entered as a regular separated problem?

Worked explanation. No. Those conditions change the supported mathematical class and require an appropriate specialist formulation or extension.

27 Eigenvalue jobs inside a workflow

```
workflow
eigen fixed_ends
    type = sturm_liouville
    interval x = [0,1]
    p = 1
    q = 0
    weight = 1
    boundary left u = 0
    boundary right u = 0
    modes = 1
    grid x = 33
end
for mode in fixed_ends.result.eigenpairs
    report mode.eigenvalue
end
```

Expected first eigenvalue near pi^2.

Repeat over grid sizes to compare the lowest eigenvalue with #pi^2. Match mode indices and check acceptance before comparing results.

Chapter 28 Linear Reaction Diffusion and Advection Diffusion

Transport models combine processes with different mathematical effects. Diffusion smooths gradients, advection shifts information and reaction can grow or attenuate it. The same sampled curve can look plausible under an incorrectly signed transport term. Version 3 provides explicit linear one-dimensional classes and numerical schemes whose assumptions can be checked.

28 1 Linear reaction-diffusion in one dimension

```
pde
type=reaction_diffusion
u_t=u_xx/10-2*u
domain x=[0,1]
interval t=[0,0.1]
initial u=sin(pi*x)
boundary left u=0
boundary right u=0
grid x=17
grid t=41
exact solution=exp(-(pi^2/10+2)*t)*sin(pi*x)
```

This equation combines diffusion with linear decay. The reference solution follows by applying u_xx=-pi^2*u to the sine profile, giving exponential decay at rate pi^2/10+2.

Linear diffusion, reaction and source coefficients can depend on x,t. The leading time coefficient must be a nonzero constant and diffusion must be strictly positive. Nonlinear terms such as u^2 are outside this class.

Numerical behaviour

Centred diffusion and Crank-Nicolson time stepping give second-order convergence for suitable smooth data. Dirichlet, normal and supported Robin boundaries use the same outward-normal convention as heat.

A-stable time stepping does not guarantee positivity for sharp or stiff data. Finer time steps and spatial resolution can be needed. Compare numerical and exact/error views where a reference exists.

28 2 Separate diffusion and reaction decay

For u_t=D*u_xx-r*u with a sine initial profile and zero-value ends, separation gives u(x,t)=exp(-(D*pi^2+r)*t)*sin(pi*x) on the unit interval. D*pi^2 is the decay contribution of the first spatial mode; r is the reaction contribution. Changing D or r can produce the same decay rate for this one mode, so one profile alone may not identify both parameters experimentally.

Higher spatial modes decay at rates D*(n*pi)^2+r. Comparing several known modes separates the dependence on n^2 from the constant reaction rate. This is a useful model-identifiability lesson even though the solver itself is not a regression or inverse-parameter fitting system. Parameter sweeps show forward consequences; they do not automatically infer D and r from data.

28 3 Why centred advection needs a separate check

In u_t=D*u_xx-v*u_x, centred spatial differences combine diffusion and advection in neighbouring coefficients. The relative strength is measured by a cell Peclet number of order abs(v)*dx/D. When this is large, a centred discretisation can give oscillatory or nonphysical profiles even with stable implicit time stepping. Refining dx reduces that ratio; choosing an upwind class addresses a different part of the approximation.

For v>0, first-order upwind approximates u_x at node i by (u_i-u_(i-1))/dx. Taylor expansion gives u_x-dx*u_xx/2+O(dx^2). Multiplying by -v adds a positive numerical-diffusion contribution v*dx*u_xx/2 to the leading

modified equation. This explains both the robustness and the smearing associated with the method. The error is first order in space when advection is present.

28 4 Linear advection-diffusion in one dimension

```
pde
type=advection_diffusion
u_t=u_xx/10-u_x+(-1+pi^2/10)*exp(-t)*sin(pi*x)+pi*exp(-t)*cos(pi*x)
domain x=[0,1]
interval t=[0,0.1]
initial u=sin(pi*x)
boundary left u=0
boundary right u=0
grid x=33
grid t=41
absolute tolerance=0.004
relative tolerance=0.004
exact solution=exp(-t)*sin(pi*x)
```

The advection term is -u_x. The forcing is chosen so the exact reference exp(-t)*sin(pi*x) satisfies the continuous equation. It is a manufactured benchmark, not an assumed input needed by the solver.

The method combines first-order upwind spatial advection, centred diffusion and Crank-Nicolson time stepping. Its mesh-error estimate uses spatial order one. It is an introductory linear transport scheme, not a high-order shock-capturing method.

Convection-dominated boundary layers and coarse time steps require care. The broader heat type retains centred advection only within its cell-Peclet check; select this explicit upwind class when appropriate.

The supplied example deliberately uses 0.004 absolute and relative targets. Requesting much tighter values can require substantially finer spatial grids and a larger work budget.

28 5 Verify the manufactured source term

Let u=exp(-t)*sin(pi*x), D=0.1 and v=1. Then u_t=-u, u_x=pi*exp(-t)*cos(pi*x), and u_xx=-pi^2*u. To satisfy u_t=D*u_xx-u_x+f, choose f=(-1+D*pi^2)*u+pi*exp(-t)*cos(pi*x). Substitution cancels the cosine transport term and leaves -u. A plus/minus mistake in f would solve a different equation while retaining smooth-looking data.

A manufactured source is a verification device. In a physical application the forcing is supplied by conservation, measurement or constitutive assumptions. One should not choose a source merely to make a preferred answer exact. When no exact function is known, inspect grid trends, mass/flux balances and comparison with independent stable schemes.

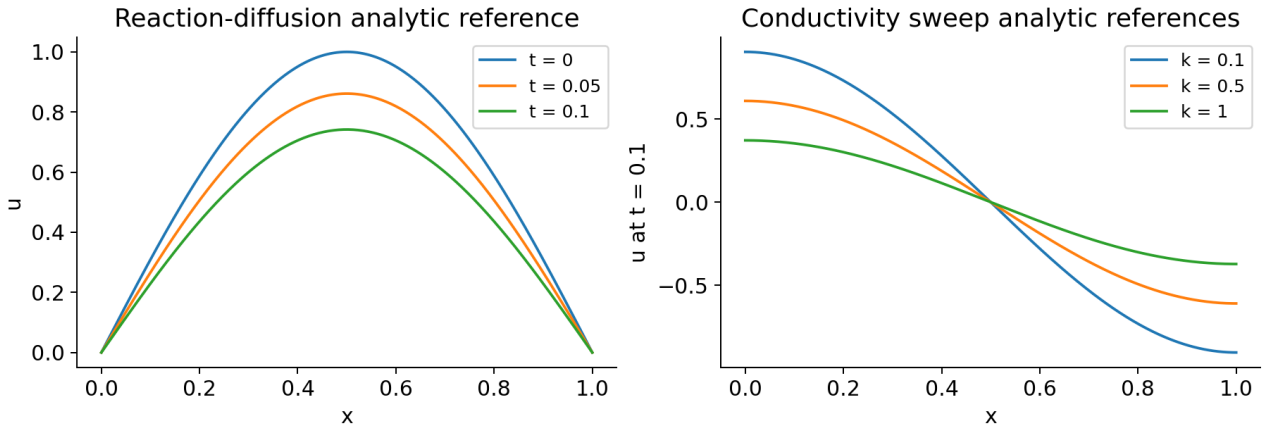


Figure 27.1 Analytical reference profiles for reaction-diffusion and an insulated heat conductivity study. Curves come from the stated formulas, not recorded solver runs.

28 6 Time accuracy positivity and scope

Crank-Nicolson is second order in time for smooth problems, but unconditional linear stability of a diffusion discretisation is not unconditional positivity or a guarantee against high-frequency oscillation. Very stiff decay with an excessively large step can alternate in sign numerically while decreasing in magnitude. Refine the time step and inspect the relevant physical quantities; a stable norm alone can conceal a poor transient.

The supported classes require linear state terms, positive diffusion and a nonzero constant leading time coefficient. They do not include nonlinear reaction kinetics, coupled species PDEs, shock-capturing fluid dynamics or arbitrary geometries. Treating u^2 as a known coefficient would change a nonlinear problem into a different linear model, not extend the solver automatically.

28 7 Practice and worked explanations

Exercise 1. What happens to first-mode decay if D doubles with r fixed?

Worked explanation. The diffusion contribution doubles, but the reaction contribution does not. The new rate is 2*D*pi^2+r, not twice the entire old rate unless r=0.

Exercise 2. Why does upwind advection usually smear a sharp profile?

Worked explanation. Its leading spatial truncation term behaves like additional diffusion proportional to abs(v)*dx/2. Reducing dx reduces this effect.

Exercise 3. A profile becomes oscillatory after increasing dt. Is diffusion necessarily negative?

Worked explanation. No. Coarse Crank-Nicolson time stepping can generate poorly resolved high-frequency transients even with positive physical diffusion. Check both the equation and mesh.

Exercise 4. Why use divisor 1 in a first-order spatial refinement estimate?

Worked explanation. For p=1, the finer-error divisor 2^p-1 is 1. This estimate still assumes the first-order term dominates.

Exercise 5. Can a source-free advective front be validated only at its peak?

Worked explanation. No. Peak agreement can coexist with positional or spreading error. Compare the full curve, transport direction and conservation evidence.

28 Compare transport coefficients across jobs

Define a reusable pde model with the transport coefficient as an explicit parameter. Sweep that parameter and inspect the accepted field and service diagnostics for each case. Chapter 3 shows the model and run syntax.

Chapter 29 Adaptive Integral Accuracy and Coupled Volterra Systems

The integral-equation chapters develop the base discrete systems. Version 3 now uses extrapolation and accuracy targets for supported smooth problems and extends Volterra marching to coupled linear components. This chapter distinguishes a raw trapezoidal value, an extrapolated output and an independently derived exact reference.

29 1 Read the current accuracy contract

A tiny discrete residual confirms the internal equations were solved consistently. It does not measure quadrature error, conditioning or the error in the final extrapolated samples.

Current automatic refinement

Successive trapezoidal solutions produce Richardson extrapolants that cancel the leading second-order term. The difference between two extrapolants, divided by 15, estimates fourth-order error under smooth asymptotic convergence. Returned values use the finer extrapolant.

```
absolute tolerance=1e-9
relative tolerance=1e-9
max refinements=6
refine=true
```

One preliminary refinement creates the first extrapolant; max refinements limits subsequent estimate-producing refinements. Every returned component must satisfy its absolute-plus-relative target. Reported discrete residuals belong to the unextrapolated internal solve.

Experiment	What to inspect
refine=false at 65,129,257 nodes	Raw trapezoidal convergence; no assessed target
Default automatic refinement	Internal grids, estimated errors and target flags
exact solution=exp(x)	Maximum and RMS error over returned nodes
Tighter precision	Roundoff effects; not a substitute for mesh refinement

accuracy_not_met retains useful numerical evidence but is not accepted accuracy. Near-singular Fredholm problems and unresolved kernel structure can invalidate simple convergence expectations.

29 2 Derive the scalar Volterra recurrence

For y(x)=1+integral_0^x y(s) ds, a trapezoidal grid with spacing h gives y_n=1+h*(y_0/2+y_1+...+y_(n-1)+y_n/2). Subtract the equation at n-1. The interior sums telescope and yield (1-h/2)*y_n=(1+h/2)*y_(n-1), with y_0=1. Thus y_N=((1+h/2)/(1-h/2))^N for N=1/h on [0,1].

Expanding the logarithm gives log((1+h/2)/(1-h/2))=h+h^3/12+h^5/80+... . Multiplying by N=1/h shows log(y_N)=1+h^2/12+O(h^4). The leading endpoint error is therefore approximately e*h^2/12. This explains why increasing from 64 to 256 subintervals reduces raw error by about 16, rather than implying that extra displayed digits become exact.

Requested nodes	Raw trapezoidal y at 1	Absolute error approximately
65	2.7183371346324106	5.53062e-5
257	2.7182852849432404	3.45648e-6
Continuum reference	2.71828182845904523536	0

29 3 Richardson extrapolation and its error estimate

Write Y_h=y+c2*h^2+c4*h^4+O(h^6). Combining R_h=(4*Y_(h/2)-Y_h)/3 cancels c2*h^2, leaving R_h=y-(c4/4)*h^4+O(h^6). A second extrapolant R_(h/2) has a leading error smaller by 16. Therefore abs(R_(h/2)-R_h)/15 estimates the finer extrapolant error under the assumed expansion.

The preliminary finer solve creates the first extrapolant. Further refinement creates an estimate-producing pair. This is why the final internal node count cannot be inferred from one output grid alone. The implementation returns the finer extrapolant sampled at requested coordinates; the discrete residual reported alongside it belongs to the internal unextrapolated solve.

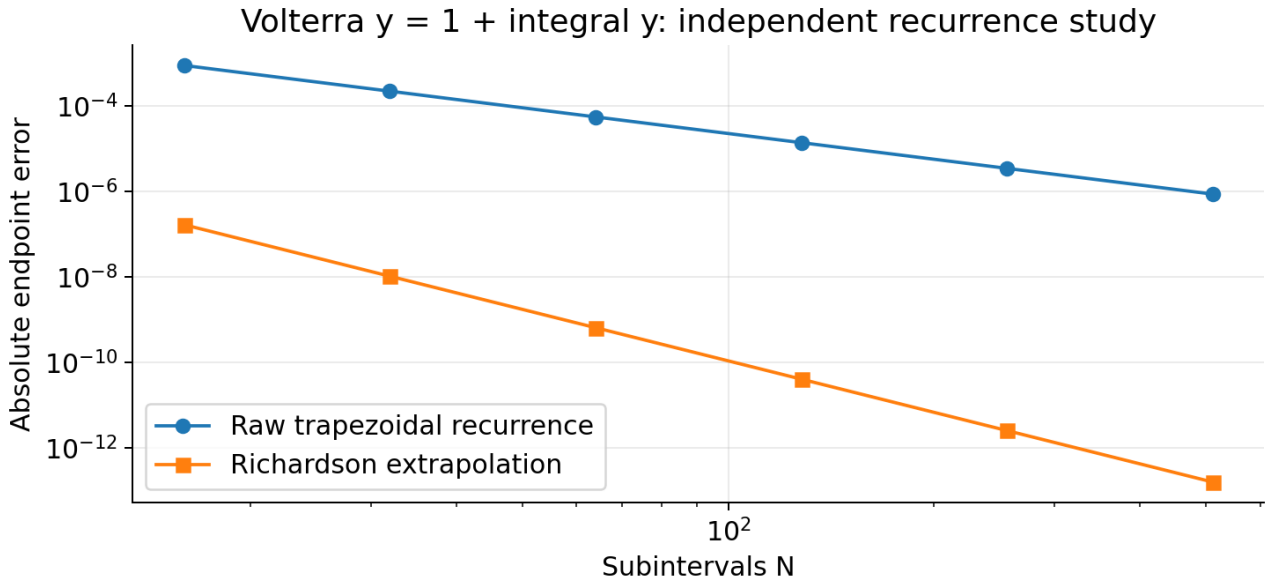


Figure 28.1 Independent high-precision evaluation of the derived trapezoidal recurrence and its Richardson extrapolation. The graph illustrates second- and fourth-order error trends; it is not a Windows timing benchmark.

29 4 Linear systems of Volterra equations

```
integral
type=volterra
components=2
interval x=[0,1]
forcing 1=1
forcing 2=2
kernel 1 1=0.5
kernel 1 2=0.25
kernel 2 1=1
kernel 2 2=0.5
lambda=1
grid x=17
exact 1=exp(x)
exact 2=2*exp(x)
```

The convention is y_i(x)=f_i(x)+lambda*sum_j integral_a^x K_ij(x,s)*y_j(s) ds. All forcing i and kernel i j entries are mandatory, including kernels that are zero.

For this example the reference vector is (exp(x),2*exp(x)). Multiplying the kernel matrix by that vector gives the same vector, which yields y'=y and initial values (1,2).

Up to eight components are supported. A pivoted small dense solve handles the coupled current endpoint at each marching step. Adaptive extrapolation assesses every returned component.

These are linear second-kind Volterra systems on finite intervals. Nonlinear or first-kind systems and coupled Fredholm systems are not added by this feature. Use the plot selector to inspect each component.

29 5 Derive a coupled reference through a matrix equation

For a constant kernel matrix A, a system y(x)=f0+integral_0^x A*y(s) ds differentiates to y'=A*y with y(0)=f0. The supplied matrix [[0.5,0.25],[1,0.5]] maps the vector [1,2] to itself, so y=exp(x)*[1,2] is an exact reference. This verifies all four kernels rather than checking only independent scalar copies.

At a trapezoidal marching endpoint, the unknown vector appears through a matrix I-lambda*h*A(x_n,x_n)/2. The solver must solve that coupled system rather than update components independently. A singular or unsafe pivot is a numerical warning about the endpoint matrix; shrinking h can change its conditioning, but cannot cure every ill-posed continuous problem.

29 6 Fredholm conditioning remains a separate issue

For the constant kernel K=1 on [0,1], let M=integral_0^1 y(s) ds and F=integral_0^1 f(s) ds. Integrating y=f+lambda*M gives (1-lambda)*M=F. Near lambda=1, small errors in F are amplified by 1/abs(1-lambda). At lambda=1, F=0 is compatible with a family while F!=0 is inconsistent. Refinement and extra precision cannot replace this operator analysis.

For a rank-one kernel x*s, the scalar moment instead leads to 1-lambda/3. These simple reductions provide powerful independent tests of a dense Fredholm solver and distinguish discretisation failure, parameter resonance and mathematical nonuniqueness. The current Fredholm implementation remains scalar; coupled support in this release is specifically Volterra.

29 7 Practice and worked explanations

Exercise 1. Why is the 65-node raw result not fixed by printing more decimals?

Worked explanation. The leading error is discretisation, approximately e/(12*64^2). More decimal output only describes that same approximation more precisely.

Exercise 2. Why divide extrapolant differences by 15?

Worked explanation. Their leading fourth-order errors differ by a factor 16 under halved spacing, so the fine-error divisor is 16-1.

Exercise 3. What is y(0) for the two-component supplied example?

Worked explanation. The integral has zero length, so y(0)=f(0)=(1,2). Check both initial components independently.

Exercise 4. Does a small LU residual imply a well-conditioned Fredholm solution?

Worked explanation. No. It measures the solved finite system; a near-singular operator can amplify quadrature and data errors.

Exercise 5. Which inputs are needed for a three-component Volterra system?

Worked explanation. All three forcing expressions and all nine kernel expressions, including explicit zero kernels, plus the common interval and controls.

29 Adaptive integral studies in workflows

Use the integral block pattern from Chapter 22 with a supported Volterra model and its refinement directives. Compare accepted trajectories on matching coordinates.

Chapter 30 Parameter Studies and Numerical Visualisation

A parameter study answers a forward question: how does the computed solution change when assigned input values change? It does not by itself optimise a design, identify parameters from measurements or classify nonlinear branches. Version 3 integrates repeated solves, per-case tables and comparison plots for the supported field, integral and eigenproblem modes.

30 1 Parameter sweeps and comparison tables

```
pde
type=heat
const k=1
u_t=k*u_xx
domain x=[0,1]
interval t=[0,0.1]
initial u=cos(pi*x)
boundary left normal=0
boundary right normal=0
grid x=17
grid t=41
exact solution=exp(-k*pi^2*t)*cos(pi*x)
sweep k=[0.1,0.5,1]
```

Each sweep value must be a finite scalar expression. Multiple sweep directives form a Cartesian product, capped at 64 combinations. Dependent constants are recalculated per case. The total stored output is capped at 200000 samples.

Sweeps apply to supported PDE, integral-equation and eigenproblem modes, not standalone quadrature or general algebraic branch following. The complete batch shares one time budget.

The summary retains every case status, target result, endpoint, maximum primary value and estimated error. Select a row for its own convergence/eigenpairs; overlay runs to compare curves or sections.

A summary maximum is a display statistic, not an optimised result. Failed cases are not silently omitted. A successful early case does not imply that all cases in the batch completed.

30 2 Predict a sweep before computing it

For insulated heat u_t=k*u_xx with initial cos(pi*x), the exact solution is exp(-k*pi^2*t)*cos(pi*x). At fixed positive t, increasing k decreases the absolute amplitude. The zero crossing stays at x=0.5. The sign does not change with positive k, and the spatial mean remains zero. These predictions check more than one endpoint statistic.

At t=0.1, compare k=0.1, 0.5 and 1. Each curve has a different amplitude but the same shape. If an overlay shifts the zero crossing, the selected section or variable may be wrong, or the model differs from this reference. If the initial frame changes across these cases, check whether an input constant was also used in the initial condition.

30 3 Sweep stiffness in an eigenproblem

```
eigen
type=sturm_liouville
const stiffness=1
p=stiffness
q=0
weight=1
interval x=[0,1]
boundary left u=0
boundary right u=0
modes=3
grid x=33
exact eigenvalue=stiffness*(n*pi)^2
sweep stiffness=[1,2,4]
```

Here p=stiffness, q=0 and weight=1, with zero values at both ends. Thus lambda_n=stiffness*(n*pi)^2. Changing stiffness multiplies eigenvalues but does not change continuum normalised sine shapes. If the physical relation is omega^2=lambda, angular frequency scales with the square root of stiffness. Compare the same one-based mode across cases.

More general parameter dependence can alter both eigenvalues and shapes. A table of lowest eigenvalues is not automatically a continuation of an arbitrarily labelled physical mode through degeneracy. Retain the definition of the reported mode and inspect the corresponding eigenfunctions. This prevents a comparison chart from becoming an unsupported branch-tracking claim.

30 4 Multiple sweeps are Cartesian products

```
const k=1
const r=0
sweep k=[0.1,0.5,1]
sweep r=[0,1]
```

When appended to a compatible model using k and r, these directives request six combinations. The constant declarations are part of a complete numerical model, not a standalone input by themselves. Dependent constants are recalculated per case. Every run shares the batch work deadline, so later cases can fail after earlier ones succeed. The summary must retain those failures rather than presenting only selected curves.

The cap of 64 combinations concerns cases, whereas the 200000 stored-sample guard concerns aggregate numerical output. A small number of large field grids can reach storage limits before the combination cap. Reduce unnecessary returned density or split an explicitly documented study into batches without concealing failed cases.

30 5 Heat maps contours and time exploration

Explore a field

After a completed PDE run, open PDE fields / animation. Choose numerical u, exact u or signed error_u when reference data exist. Enable the contour overlay to inspect level sets. Move the slider to inspect a spatial section or time frame.

Animate/Pause plays the available transient frames for heat, wave and transport equations. Playback does not integrate again or insert intermediate numerical time steps. A map uses a global colour range for that supplied field.

Explore curves and errors

Use the variable selector for an integral component, eigenfunction or reference/error curve. The numerical curve can overlay its matching exact curve. Table values and JSON are the appropriate records when more digits than display precision are needed.

Export	Use
Map PNG	Share a field or signed-error view
Comparison chart PNG	Retain curves or sweep overlays
Active table CSV	Retain convergence, eigen or comparison data
Full results JSON	Retain numerical strings, flags and nested data

A smooth picture does not establish accuracy. Contours can conceal unresolved layers; colour ranges across separate runs can differ. Use grid refinement and numeric comparisons alongside the plots.

For parameter sweeps, final-time PDE sections (or middle-y steady sections), integral curves and selected eigenmodes can be overlaid. The overlay is a visual comparison, not an optimisation or stability classification.

30 6 Compare quantities with the same meaning

A final-time section answers a different question from a maximum over all recorded times. A middle-y steady section is not the full two-dimensional field. A displayed maximum can change when reporting coordinates change, even if the underlying internal solve is similar. Record the selected section, component, time and colour range beside a visual comparison.

Signed error means numerical value minus the supplied reference, not relative error. A sign-changing error map can reveal phase or spatial bias; a small-looking colour range can also hide a large absolute error if each run rescales its own palette. Use exported numbers and a common numerical scale when making quantitative claims across runs.

30 7 Keep forward studies separate from deferred features

There is no dimensional unit checker, spreadsheet formula engine, general interpolation table, automatic regression, uncertainty propagation or optimisation objective in this release. A forward sweep can support those activities externally, but its numerical summary alone does not implement them. This boundary keeps the interpretation of a completed table precise and testable.

30 8 Practice and worked explanations

Exercise 1. Three values of k, four of r and two of length request how many cases?

Worked explanation. Twenty-four, because the directives form a Cartesian product. Verify that the combined model remains in scope for every case.

Exercise 2. Does moving the time slider improve the numerical time step?

Worked explanation. No. It selects computed frames. Solve with an appropriate time grid or accuracy target to change the numerical calculation.

Exercise 3. Can the smallest summary maximum be reported as an optimum?

Worked explanation. Only as the smallest statistic among those sampled cases, not as a proven optimum over a continuous parameter domain.

Exercise 4. Why retain failed sweep cases?

Worked explanation. They define the tested domain and reveal parameter-dependent conditioning or budget limits. Omitting them biases the apparent reliability of the study.

Exercise 5. Why can different colours represent the same value in separate runs?

Worked explanation. Each field can have its own extrema and colour range. Compare labelled numeric values or enforce a consistent external scale.

30 Script a parameter study

Use a workflow when a study needs several models, conditional acceptance checks or results passed between services. See Chapter 31 for a quadrature sweep and Chapter 43 for a multi-service study.

Chapter 31 Numerical Definite Integration on Finite and Infinite Intervals

Standalone quadrature computes the numerical value of an integral whose integrand and limits are known. It is different from an integral equation for an unknown function and different from a symbolic antiderivative. The finite computational task must still assess endpoint behaviour, tails, cancellation, sampling and arithmetic range. Version 3.0.1 makes this a separate mode with explicit diagnostics.

31 1 Standalone numerical definite integration

```
quadrature
integrand = 1/sqrt(t*(1-t))
variable = t
interval = [0,0.5]
absolute tolerance = 1e-10
relative tolerance = 1e-10
```

The reference value is pi/2, approximately 1.57079632679489661923. The endpoint t=0 is singular, but the integral converges. Supported endpoint singularities are handled internally; values exactly at the two limits need not exist.

What quadrature does

It returns one approximate numerical value of a one-dimensional definite integral of a known function. Both finite and infinite limits are supported. No exact solution or antiderivative is required.

What it does not do

It does not produce a symbolic indefinite integral or introduce integrate(...) into ordinary equations. Every parameter must already have a numerical value. It does not solve an equation containing an unknown-dependent integral.

Need	Use
Integral of a known function	quadrature
Unknown function in a second-kind integral equation	integral and type=fredholm or volterra
Symbolic antiderivative	Outside this numerical utility

A finite result and a successful status are both needed. On failure a candidate may be displayed for diagnosis, but it is not an accepted integral.

31 2 Derive the first singular reference

For integral_0^(1/2) 1/sqrt(t*(1-t)) dt, set t=sin(theta)^2 on theta in [0,pi/4]. Then dt=2*sin(theta)*cos(theta) dtheta and the denominator is sin(theta)*cos(theta). The transformed integrand is 2, giving pi/2. This calculation establishes the mathematical reference without implying that the numerical method must discover the substitution symbolically.

The endpoint value at t=0 does not exist as a finite number, but nearby area behaves like the integral of t^-1/2 and is finite. In contrast, integral_0^1 1/t dt diverges logarithmically. A finite sampled value away from the endpoint cannot distinguish those classes by itself; asymptotic analysis and honest tail diagnostics matter.

31 3 Infinite limits constants and interval notation

```
quadrature
const a=3
const b=2
integrand=a*exp(-b*t)
variable=t
interval=[0,+infinity]
```

The reference is a/b=1.5. Use const a=3 or constant a=3, not an algebraic equation a=3. Unknown parameters, equations, sweep directives and other model markers are rejected in quadrature mode.

Interval	Interpretation
[a,b]	Finite endpoints; scalar expressions or assigned constants
[a,inf]	Right infinite tail
[-inf,b]	Left infinite tail
[-infinity,+infinity]	Whole real line, split at zero internally
[b,a]	Reversed limits negate the result
[a,a] with finite a	Zero-width integral returns zero

inf and infinity, with optional signs, are case-insensitive endpoint tokens only. They are not general constants for use inside expressions. Identical infinite limits are invalid. variable defaults to x and must be an unassigned identifier.

US input uses interval=[0,0.5]. EU style uses interval=[0;0,5] with --style EU or the matching Desktop style. JSON continues to use decimal points.

Whole-line tails must converge independently; cancellation between divergent tails is not accepted as an ordinary improper integral. A principal value is a different mathematical problem and is outside scope.

31 4 Derive useful infinite references

The exponential right tail integrates exactly: integral_0^infinity exp(-x) dx=1. Its left-tail counterpart integral_-infinity^0 exp(x) dx is also 1. Both are absolutely convergent and exercise opposite transformations. Reversing either interval must negate the result without changing the mathematical convergence class.

For the Gaussian, square the whole-line integral, apply Fubini to the positive integrand and use polar coordinates in the plane. The double integral becomes 2*pi*integral_0^infinity r*exp(-r^2) dr=pi, so the positive one-dimensional integral is sqrt(pi). This is an analytical reference; the implemented utility itself remains one-dimensional.

A whole-line principal value can exist while ordinary tails diverge. For example, cancellation of positive and negative contributions in symmetric truncations does not establish that the two one-sided integrals exist. The solver requires independent absolute tail convergence and does not compute Cauchy principal values. Do not reinterpret an unresolved-tail message as a principal-value answer.

31 5 Quadrature parameters and sensible defaults

Parameter	Default	Allowed / role
absolute tolerance	1e-10	Nonnegative absolute error contribution
relative tolerance	1e-10	Nonnegative relative error contribution
max refinements	10	1..16; successive transformed-grid levels
max evaluations	1000000	32..10000000; evaluation budget
subdivisions	4	1..64; initial finite-interval pieces
max extent	8	2..12; transformed variable range

At least one tolerance must be positive. The target is absolute tolerance+relative tolerance*abs(integral). A positive absolute tolerance is useful for integrals near zero. This is an estimated acceptance threshold, not a certified mathematical bound.

```
absolute tolerance=1e-12
relative tolerance=1e-12
max refinements=12
max evaluations=1000000
max extent=8
```

Adjust the control named by the diagnostic

Start with defaults. If refinement_limit or evaluation_limit occurs, first inspect convergence and then raise the relevant budget if justified. subdivisions can help finite-interval local structure; it does not control an infinite-tail extent.

The normal engine time budget and cancellation mechanism also apply. Several qualifying levels are required before acceptance, so a very small refinement budget can prevent acceptance even when a candidate looks plausible.

Quadrature uses binary128 guard arithmetic in standard mode. Extended selection retains the normal entitlement checks. Neither mode promises hundreds of accurate digits.

31 6 How max extent and transformations work

max extent limits the transformed coordinate u, approximately -E<=u<=E. It is not the physical endpoint x=E and it does not replace an infinite bound with E.

Finite-interval tanh-sinh

```
x = (a+b)/2 + (b-a)/2 * tanh((pi/2)*sinh(u))
```

Samples approach both physical endpoints very rapidly. In implementation, stable endpoint distances are used; when rounding prevents a distinct interior point, the endpoint-precision condition enters the tail checks.

Right-tail exp-sinh

```
x = a + exp((pi/2)*sinh(u))
```

For a left tail the corresponding form is x=b-exp((pi/2)*sinh(u)). The quadrature weight contains the transformation derivative. On [0,inf], E=4 can already reach x of order 10^18; E=6 reaches about 10^138 and E=8 about 10^1017.

Increasing extent explores farther toward endpoints/tails. Increasing refinement makes sampling denser. Increasing one is not a substitute for the other, and a larger extent need not improve accuracy.

Very large transformed x can overflow intermediate expressions. Build 3.0.1.1 adds scaled evaluation for supported overflow cases, but invalid domains, unrepresentable function arguments and unresolved cancellation can still require reformulation.

31 7 Why double exponential transforms help

An algebraic endpoint singularity f(x) approximately C*(b-x)^(-alpha), with alpha<1, is integrable. Near a tanh-sinh endpoint, distance b-x decays roughly like exp(-C1*exp(abs(u))). The transformation derivative contributes a comparable rapidly decaying factor. For alpha<1, the weighted integrand decays very rapidly in u, making a finite transformed grid effective for many such cases.

This argument does not certify every expression. Finite arithmetic eventually rounds x onto an endpoint or loses a small difference inside the integrand. A narrow interior peak can also lie between sampled locations. Independent shifted grids, absolute-mass checks and tail estimates reduce false acceptance, but no finite sampling scheme can prove that every adversarial unsampled feature is absent.

max extent is not a cure for poor local resolution. Increasing the transformed span can reach more extreme x while leaving sampling too coarse for a narrow peak. Increasing refinement can sample more densely while still failing to explore a remote tail. The diagnostic determines which budget is relevant; a larger number is not automatically a better numerical calculation.

31 8 Quadrature acceptance and failure diagnostics

The solver compares ordinary and independently shifted transformed grids, changes across refinement levels, absolute integral mass, tail decay and a roundoff allowance. Acceptance requires consecutive qualifying levels.

Status / field	Meaning
integration_complete	Accepted integral in value; quadrature_successful=true
nonfinite_integrand	Domain failure, division by zero or unresolved range problem
interior_singularity	A detected interior denominator zero; split/analyse it
tail_not_resolved	Absolute tail decay was not adequately established
unresolved_zero	All samples vanished without proving a structural zero
precision_limit / roundoff_limit	Arithmetic or endpoint resolution limits
refinement_limit / evaluation_limit	Work budget exhausted before acceptance
time_limit / cancelled	Interrupted numerical work
candidate_value	Last unaccepted candidate when value is null

Read estimated_absolute_error, estimated_tail_error, evaluations and the convergence rows. A small phase difference alone is insufficient. Polynomial denominator checks catch common interior poles but cannot exclude every transcendental singularity or unsampled feature.

The intended improper class is absolute convergence. Symbolic integration, principal values, dedicated conditionally convergent oscillatory-tail algorithms and multidimensional quadrature are not implemented.

Failure to establish convergence is not a general proof of divergence. A plausible candidate is not promoted to an answer merely because it matches an expected reference. Version 3.0.1 adds targeted fallbacks for difficult quadrature cases. Check the final status, accepted value and error target after a fallback.

31 9 Reciprocal powers and intermediate overflow

```
quadrature
integrand=1/(1+x^2)^5
variable=x
interval=[0,inf]
```

Set x=tan(theta). The integral becomes integral_0^(pi/2) cos(theta)^8 dtheta. Repeated integration by parts gives I_m=(m-1)*I_(m-2)/m for even m, with I_0=pi/2. Thus I_8=(7*5*3*1)/(8*6*4*2)*pi/2=35*pi/256, approximately 0.42951462060797954432.

The original improper transformation sampled huge x, where (1+x^2)^5 overflowed despite a tiny reciprocal. Numerical engine build 3.0.1.1 retries supported nonfinite expressions with sign/log-magnitude scaling and combines the quadrature weight before materialising the weighted value. It preserves representable ordinary function arguments where possible. This corrects the reported reciprocal-power case without silently dropping nonfinite samples.

Scaling is not universal symbolic simplification. Invalid real domains, genuine division by zero, unrepresentable function arguments and unresolved cancellation still need diagnostics. Earlier-build workarounds max extent=6 or integrand=(1/(1+x^2))^5 work for this particular reference, but are not general remedies for all improper integrals.

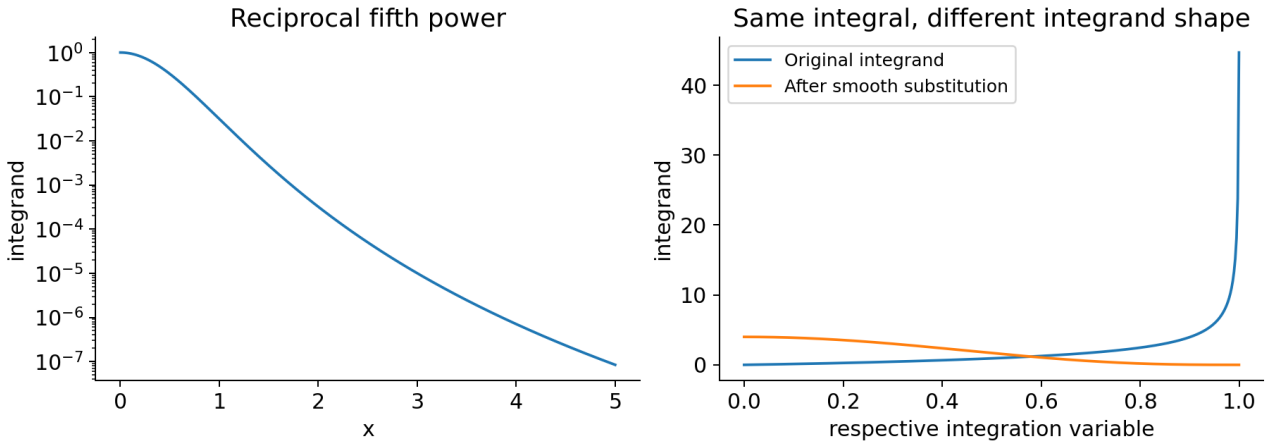


Figure 30.1 Analytical integrand profiles. The original endpoint-singular function and its smooth transformed counterpart have equal integrals on their respective variables; their pointwise heights need not match.

31 10 Bound a finite cutoff independently

For x>=R>0, (1+x^2)^5>x^10. Consequently the omitted right tail is below integral_R^infinity x^-10 dx=1/(9*R^9). R=10^20 gives a bound about 1.12e-181. That is a rigorous inequality for this mathematical tail; it is not an error bound on a floating-point quadrature result over the remaining finite interval.

The distinction is useful in reporting. State truncation error, discretisation/sampling estimate and arithmetic limits separately. A cutoff accurate to hundreds of digits does not make a binary128 result accurate to hundreds of digits. Binary128 arithmetic provides about 34 significant decimal digits, and an actual accepted result can have fewer trustworthy digits than that.

31 11 A nested exponential left tail

```
quadrature
integrand=exp(-0.0009*t^2-t-0.5*exp(-t))
variable=t
interval=[-10,inf]
```

For t=-s, the omitted left integrand is exp(g(s)), where g(s)=-0.0009*s^2+s-exp(s)/2 and s>=10. Its derivative g'(s)=-0.0018*s+1-exp(s)/2 is negative and decreasing there. Thus g(s)<=g(10)+g'(10)*(s-10), yielding a tail bound exp(g(10))/(-g'(10)). With g(10) approximately -11003.32, this is below 10^-4782.

The inner exp(-t) can overflow far to the left even though the outer negative exponential suppresses the true integrand. The finite cutoff is mathematically justified for this example by the independent bound. Do not use the same cutoff for another integrand without a new argument; shifting a lower limit changes the mathematical problem.

31 12 Endpoint cancellation and a regression case

```
quadrature
integrand=x/sqrt(1-sqrt(x))
variable=x
interval=[0,1]
```

The exact value is 64/35, approximately 1.8285714285714285143. The endpoint singularity behaves like sqrt(2)/sqrt(1-x), so it is integrable.

Earlier reported Windows failure

An earlier Windows run reported nonfinite_integrand: division by zero at an interior quadrature point, despite a candidate within about 2.77e-17 of the reference. Cancellation in 1-sqrt(x) near x=1 is consistent with that report. The updated native quadrature source accepts the original input in a focused check. Recheck the matching installed Windows build; this source-level result is not a new Windows acceptance test.

An equivalent stable reformulation

```
quadrature
integrand=x*sqrt((1+sqrt(x))/(1-x))
variable=x
interval=[0,1]
```

This follows by rationalising 1-sqrt(x)=(1-x)/(1+sqrt(x)). It was tested successfully in the native engine; Windows acceptance remains to be checked.

A smooth reference problem follows from s=sqrt(1-sqrt(x)): the integral becomes integral_0^1 4*(1-s^2)^3 ds=64/35. It is useful for separating endpoint evaluation from integration accuracy.

Retain both the original model and its equivalent reformulation as release regression tests. The current quadrature source includes generic checks for supported removable singularities, endpoint cancellation and recognised corners; it does not automatically stabilise every possible expression. Always read the accepted value and status from the installed build.

31 13 Derive the endpoint reference exactly

For I=integral_0^1 x/sqrt(1-sqrt(x)) dx, set s=sqrt(1-sqrt(x)). Then x=(1-s^2)^2 and dx=-4*s*(1-s^2) ds. The singular s in the denominator cancels and reversing endpoints gives I=4*integral_0^1 (1-s^2)^3 ds. Expanding gives 4*(1-1+3/5-1/7)=64/35.

An artificial zero denominator near an endpoint must not be confused with divergence. A close candidate from a failed run remains unaccepted. The original expression is a useful regression check for each matching build: verify

the status, accepted value and error target against 64/35 rather than overriding a failure because a candidate looks accurate.

31 14 Local structure and independent reformulations

Analyse known interior singularities and distinguish genuine poles from removable limits. The updated engine checks supported removable singularities and splits recognised nonsmooth corners automatically; unresolved or genuine interior singularities still need analysis and may be rejected. If you split the interval, require acceptance of every piece and combine its error estimates appropriately. Finite sampling cannot exhaustively exclude general transcendental singularities.

Rescale a narrow or remote feature into a well-resolved variable and transform both limits and the derivative weight. Merely substituting a new name in the expression changes no scale. An algebraically equivalent stable formula can preserve a small quantity that subtraction loses. Validate reformulations with independent algebra and retain both original and transformed models as tests.

Worked example with explicit ranges and declared singularities

Integrate 1/√|x−0.2| on [0, 1]. The integrand is unbounded at x = 0.2. Run each listing separately. The first supplies two intervals in one range directive and combines their contributions. The second supplies the overall range and asks the solver to split it at the declared point.

```
quad
expr = 1/sqrt(abs(x-0.2))
var = x
range = [0, 0.2], [0.2, 1]
```

```
quad
expr = 1/sqrt(abs(x-0.2))
var = x
range = [0, 1]
singularities = [0.2]
```

Reference. Substituting u = 0.2−x on the left and u = x−0.2 on the right gives contributions 2√0.2 and 2√0.8. The sum is 6/√5 ≈ 2.683281573. Both one-sided improper integrals converge. Compare the accepted value and error estimate from each run with the exact reference.

Exercise. Replace the square-root singularity with 1/|x−0.2|. Both sides now diverge logarithmically. Splitting identifies the boundary of each piece and leaves its convergence to be assessed. Chapter 35 shows how to investigate other candidate points before deciding on a split.

31 15 Practice and worked explanations

Exercise 1. What is integral_0^1 x^-1/2 dx and why is its endpoint allowed?

Worked explanation. It equals 2 from the antiderivative 2*sqrt(x). The unbounded endpoint is integrable; a quadrature need not evaluate the integrand exactly at zero.

Exercise 2. Why is max extent=8 not an upper limit x=8?

Worked explanation. Extent bounds the transformed coordinate u. The exp-sinh map can send u=8 to x of order 10^1017 on [0,infinity].

Exercise 3. What does a null accepted value with a plausible candidate mean?

Worked explanation. No accepted quadrature result was produced. The candidate is diagnostic evidence and must not be reported as a successful answer.

Exercise 4. For integral_R^infinity x^-6 dx, derive the cutoff tail.

Worked explanation. The integral is 1/(5*R^5), provided R>0. This bound alone says nothing about numerical error in another part of the calculation.

Exercise 5. Can extra precision remove all endpoint cancellation?

Worked explanation. It postpones range/rounding limits but does not eliminate ill-conditioned expression evaluation. Stable algebra or a justified transformation can still be needed.

Exercise 6. What is wrong with I=integrate(f(x),x,0,1) inside ordinary equations?

Worked explanation. Version 3.0.1 does not implement that expression function. Use standalone quadrature for assigned known data; unknown-dependent integral equations require the supported integral mode or another validated formulation.

Exercise 7. How should integral_-infinity^infinity x/(1+x^2) be classified?

Worked explanation. Its symmetric principal value is zero but its ordinary one-sided integrals diverge. The current absolute-tail quadrature is not a principal-value solver.

31 Multiple quadratures in one workflow

Use named quad blocks for different integrands or independent intervals. Each job has its own status, acceptance flag and error estimate. For one integral split across intervals, use multiple ranges or declared singularities within one quad job and inspect the combined acceptance.

```
workflow
quad left_piece
    expr = x^2
    var = x
    range = [0,1]
end
quad right_piece
    expr = x^2
    var = x
    range = [1,3]
end
if left_piece.accepted
    if right_piece.accepted
        report left_piece.value + right_piece.value
    else
        report right_piece.status
    end
else
    report left_piece.status
end
```

The expected sum is 9. For a sweep of the same model, define it once and run it with different arguments.

```
workflow
model power_integral(b) as quad
    expr = x^2
    var = x
    range = [0, @b]
end
values = []
for n in 1..3
    run area = power_integral(n) within 2
    if area.accepted
        append values with area.value
    else
        report area.status
    end
end
report values
report count(values)
report at(values, 2)
```

Expected values are approximately [0.3333333333333333, 2.6666666666666667, 9], count 3, and second entry 2.6666666666666667. In EU number style, use decimal commas and semicolons inside lists, intervals and function arguments, such as at(values; 2).

Each invocation has a two-second job cap. Increase the overall time budget if the set of jobs and overhead need longer.

Chapter 32 Convergence Verification and Reproducible Research

A trustworthy numerical study makes its question, assumptions, input, work limits and interpretation reproducible. Exact examples are valuable tests, but most research models have no closed form. The appropriate response is layered numerical evidence, not a requirement to invent an exact solution or a claim that a tiny residual proves all digits.

32 1 Distinguish verification validation and exploration

Verification asks whether the numerical implementation solves its stated mathematical problem as intended. Validation asks whether that mathematical model represents the physical or empirical system sufficiently for its purpose. Exploration asks what behaviours or hypotheses follow when inputs change. A manufactured solution supports verification but does not validate an experimental constitutive law.

An observed convergence rate supports a discretisation assumption. A physical conservation check supports a balance. Agreement between independent methods can expose or reduce implementation mistakes, but two methods can share the same wrong boundary data. Report which layer each check addresses rather than collecting unrelated small numbers under the word accuracy.

32 2 Exact solutions remain useful even when unavailable in applications

Choose u first, then derive the source, initial values and boundaries from its derivatives. A polynomial often tests indexing with small truncation error; a sine or exponential tests convergence. A time-dependent Robin reference tests signs and temporal compatibility. A coupled integral reference tests matrix indexing and all components. A diverse test set prevents a method from appearing reliable merely because one example is integrated exactly by its quadrature.

For eigenproblems, a known spectrum tests indexing, zero modes and grid convergence; normalised reference eigenfunctions check sampled shape. For quadrature, substitutions and recurrence relations can give independent constants. These are valuable development or teaching references, not capabilities the user must supply for every practical solve.

32 3 Build a study when no closed solution exists

First establish dimensional and domain consistency outside the software, because automatic unit checking is not included. Next test a simpler limiting case with a known solution. Then compare at least three sufficiently resolved grids or tolerance settings for the full problem. Check relevant conservation, sign and boundary relations. Where practical compare with a method that has different numerical failure modes.

Retain failed attempts. A plateau under refinement can indicate roundoff, iterative error, bad conditioning or a failure of the assumed convergence regime. A monotone sequence alone does not prove convergence to the intended solution. If sharp layers or singularities are present, use an appropriate specialised method rather than forcing a smooth-order interpretation.

Interpret bounded numerical investigation

For ODEs, BVPs, PDEs and integral equations, bounded numerical investigation can supply an additional validation step. Record the work limits, final status and reason. An unresolved or ambiguous outcome calls for examining the model assumptions, supplied data and numerical evidence before using the result. For example, compare plausible BVP branches by checking the boundary conditions and residuals for each one.

32 4 Calculate observed order from three grids

If errors behave like C*h^p, let D1 be a consistent norm of U_h-U_(h/2) and D2 the same norm of U_(h/2)-U_(h/4). Then p_observed=log(D1/D2)/log(2). Differences must compare the same physical coordinates, state components and normalisation. Near zero or roundoff, this quotient is unstable and should not be presented as a reliable order.

As an illustrative calculation, differences 0.004 and 0.001 give p=2. Differences 0.004 and 0.002 give p=1. These are explanatory numbers, not measured engine results. Real runs must record their own internal grids and difference definitions. Different components can be dominated by different error mechanisms.

32 5 Set tolerances in the scale of the result

The absolute-plus-relative field/quadrature target is atol+rtol*abs(value). If a true value is near zero, relative tolerance alone can demand an impractical absolute error. If component magnitudes differ greatly, one global target may have different practical significance across components. IVP component tolerances and field controls have different syntax; do not transfer a setting blindly between modes.

Do not tighten a target below justified input accuracy and expect a physically more accurate prediction. A conductivity known to three significant digits cannot support thirty-digit material conclusions, even if a numerical reference is evaluated that precisely. Distinguish discretisation studies using exact assigned coefficients from uncertainty in measured coefficients.

32 6 Build a Windows acceptance matrix

Interface or feature	Acceptance question	Record
CLI and DLL	Do the same models produce compatible statuses and values?	Model, options, JSON and actual binary version
Desktop tables	Are problem types, coordinate names and flags interpreted correctly?	Completed report and exported numerical data
Plots and animation	Are selected component, section, sign and frame labels correct?	Panel exports plus the underlying values
Projects and number styles	Are settings and bounds restored without changing meaning?	US/EU round-trip model comparisons
Licensing and cancellation	Are entitlements and interrupted-run snapshots handled honestly?	Account-specific tests without exposing private data
Distribution	Do the EXE, DLL, runtime files, examples and guides match?	Package checksums and local smoke-test output

Cross-compilation and PE inspection establish architecture, imports, exported names and version resources. They do not execute the numerical runtime on Windows or interact with the GUI. A native Linux pass is useful evidence, but the endpoint-cancellation report demonstrates why it cannot replace a platform acceptance pass. Keep that distinction in release notes and teaching examples.

32 7 Recorded regression evidence

The engine 3.0.1.1 release record contains 69 quadrature cases in each precision mode through CLI and DLL, 43 Version 3 numerical cases per precision/interface, 104 field CLI cases, 16 field DLL cases, 15 scalar-root checks and

23 teaching examples. These records concern their stated models and environment. They do not establish success on every integrand or certify Windows GUI behaviour.

Book production checked all 17 new complete Version 3 listings with the native engine in standard guard mode; all completed successfully. The endpoint result is a native success, not a Windows fix. The numerical models and mathematical references are paired in the chapter text. The existing approximately one hundred thousand words of application and method material retain their stated mathematical arguments and historical benchmark provenance. This edition does not turn every inherited timing into a freshly measured Version 3 performance claim.

32 8 A reproducible testing and reporting workflow

Retain the complete test

Save the exact input, expected reference and its derivation or independent source, number style, precision, model controls, work budget, public version and engine build. Retain JSON and the run report; use a screenshot only as supporting evidence.

Test group	What to check
Known-reference models	Values, maximum/RMS error and target acceptance
Models without closed forms	Refinement, conservation and independent-method agreement
Adversarial / invalid inputs	Clear diagnostics; no silently accepted invalid result
Budget and cancellation tests	Honest status, candidate/partial-data semantics
Windows CLI / DLL / Desktop	Same mathematical result and compatible interpretation
Plots and exports	Correct variables, signs, precision preservation and run snapshot
Packaging / guides	Matching EXE/DLL, examples, checksums and documented scope

For quadrature include singular endpoints, both infinite directions, reversed limits, overflow, cancellation, narrow features and divergent tails. For fields include normal signs, Robin compatibility, CFL failures and refinement-budget stops. Add the Chapter 31 split-integral pair and the Chapter 35 standalone singularity example to the installed-build checks.

Send a useful defect report

Provide the smallest reproducing model and the full status/reason. State whether a reformulation works and on which platform. Report candidate values as candidates. Distinguish a computed discrepancy from an incorrect reference or a changed model.

During the planned pre-release test period, keep a stable baseline. After a numerical change, rerun its regression cases and the relevant broader suites. Update this guide when an open issue is verified as corrected.

32 9 A university laboratory sequence

Session	Student task	Evidence to assess
1	Solve a polynomial and an integer model with known alternatives	Domain, full reference and substituted identities
2	Compare raw Volterra grids and extrapolation	Predicted order, endpoint error and internal work
3	Derive mixed boundary data from x^2+y^2	Outward signs, source and flux balance
4	Compare Dirichlet and Neumann eigenmodes	Mode index, zero mode and normalisation
5	Study transport and a parameter sweep	Forcing derivation, scheme order and failed cases
6	Test singular/infinite quadrature and an open failure	Convergence analysis, status and stable reformulation

Assess reasoning more heavily than screenshots. Students should distinguish what was supplied, computed, inferred and proved. A correct diagnosis of an unsupported or unresolved problem can demonstrate more understanding than an unchecked matching decimal. Research use should add independent references, experimental context and a retained computational record.

32 10 Practice and worked explanations

Exercise 1. A fine grid and a coarse grid agree to many digits. What remains uncertain?

Worked explanation. Both can miss the same narrow feature or share the same modelling error. Check another resolution, independent structure and method assumptions.

Exercise 2. What is observed order when successive differences are 0.008 and 0.002?

Worked explanation. log(4)/log(2)=2, provided the same norm and coordinates are used and differences are not dominated by other errors.

Exercise 3. A Windows failure passes on Linux. Is it fixed?

Worked explanation. No. It narrows the investigation but does not reproduce or repair the Windows runtime behaviour. Retain the exact Windows input, versions and diagnostic.

Exercise 4. Does an exact manufactured solution validate a fluid-property model?

Worked explanation. No. It verifies numerical treatment for that chosen equation and data. Physical validation requires appropriate constitutive and empirical evidence.

Exercise 5. What should a pre-release issue record contain?

Worked explanation. A minimal model, platform and binary versions, settings, accepted versus candidate status, independent expectation, reproducible steps and a regression disposition.

Chapter 33 Numerical Differentiation of Mathematical Expressions

A derivative is a local sensitivity, not simply another value of a function. The standalone diff/differentiate modes evaluate an expression at declared coordinates. First derivatives can use automatic differentiation; higher derivatives and Hessians require numerical differentiation and diagnostic checks. The aliases expr/expression, grad/gradient and hess/hessian are accepted.

33 1 A multivariable sensitivity calculation

Selected from challenging Problem 61

```
diff
expr = x^2*exp(y)+sin(x*z)+y^3*z
at x = 1
at y = 0
at z = 2
gradient = true
hessian = true
```

Reference. At (1,0,2) the function is 1 + sin(2). The gradient is (2+2cos(2), 1, cos(2)); each component follows by differentiating the original expression while holding the other coordinates fixed.

Interpretation. The Hessian is a matrix of second partials, not a list of unrelated slopes. The off-diagonal entries should be symmetric within numerical error.

33 2 Higher order and inverse hyperbolic derivatives

Selected from challenging Problem 65

```
diff
expr = exp(2*x)
at x = 0
wrt = x
order = 4
```

Reference. d^4/dx^4 exp(2x) at x=0 equals 16.

Interpretation. A higher-order numerical derivative is especially sensitive to step selection and cancellation; compare the result with the exact derivative rather than counting printed digits.

Selected from challenging Problem 68

```
differentiate
expr = acosh(x)
at x = 2
wrt = x
order = 1
```

Reference. d/dx acosh(x) = 1/sqrt(x^2−1) for x>1; at x=2 the answer is 1/sqrt(3).

Interpretation. The real-domain restriction matters. acosh is shared with the normal mathematical expression core; other Compute-only special functions are not necessarily shared.

33 3 Removable Taylor remainders

Selected from challenging Problem 66

```
diff
expr = (exp(1-x^2)-1-(1-x^2)-(1-x^2)^2/2)/(1-x^2)^3
at x = 1
wrt = x
order = 1
```

Reference. The continuous extension has value 1/6 at x=1 and first derivative −1/12.

Interpretation. Severe cancellation makes naive finite differencing of the original quotient unreliable. Stable Taylor continuation is essential.

33 4 Exercises

1. Find the gradient of x*y + sin(x) at (1,2). 2. Calculate the mixed partial of x^3y^2 + sin(xy) at (1,2). 3. Compare the fourth derivative of exp(2x) at 0 using successively tighter settings and explain any loss of numerical accuracy.

Answers. 1. (2+cos(1),1). 2. The mixed partial is $6x^2y$ + cos(xy) − xy sin(xy), giving 12 + cos(2) − 2sin(2) at (1,2). 3. Exact reference 16; observed numerical errors depend on precision and method.

33 Differentiation jobs inside a workflow

```
workflow
diff slope
    expr = x^3
    at x = 2
    wrt = x
end
report slope.result.requested_derivative
```

Expected first derivative 12 at x = 2.

Use at x = @query inside a for loop to compare derivatives at several coordinates. Inspect status before reading the requested derivative.

Chapter 34 Differentiation and Interpolation of Tabular Data

Tabular differentiation estimates slopes or curvature from measurements. Interpolation reconstructs a chosen mathematical curve between samples. Neither operation establishes the physical law behind noisy observations. Prefer explicit uncertainty propagation and avoid uncontrolled extrapolation.

34 1 A polynomial data table

Selected from challenging Problems 69 and 73

```
diff
data = [[0,0],[1,1],[2,4],[3,9],[4,16],[5,25]]
at x = [1.5,2.5,3.5]
order = 1
method = auto
```

Reference. The underlying table represents $y=x^2$, so reference derivatives are 3, 5 and 7.

Interpretation. Irregularly spaced x values are allowed; repeated x positions are rejected. Tabular uncertainty is distinct from error due to the chosen interpolation model.

Interpolation of the same quadratic table

```
interpolate
data = [[0,0],[1,1],[2,4],[3,9],[4,16],[5,25]]
at x = [1.5,2.5,3.5]
method = local_polynomial
```

Reference. Values are 2.25, 6.25 and 12.25.

Interpretation. The local polynomial method reproduces this quadratic. For nonsmooth or monotone measurements, alternatives may be preferable.

34 2 Choosing an interpolation method

Supported methods are linear, local_polynomial, pchip, cubic_spline, akima, barycentric and smoothing_spline. auto selects a local polynomial. PCHIP preserves monotone shape, natural cubic spline is smooth, Akima dampens some abrupt slope changes, and a smoothing spline uses a penalty lambda for noisy observations. Barycentric global interpolation accepts at most 32 samples; Akima needs at least five. The usual local window is limited to 15 observations.

```
interp
file = "tests/interpolation_sample.csv"
x_column = x
y_column = y
method = cubic_spline
at x = [1.5,2.5]
output_csv = "interpolation_output.csv"
```

34 3 Measurement uncertainty and CSV results

```
diff
file = "tests/tabular_research_data.csv"
x_column = time
y_column = displacement
error_column = sigma
method = savitzky_golay
window = 7
degree = 2
order = 1
uncertainty = true
output_csv = "derivative_output.csv"
```

Savitzky–Golay uses an odd window and fits a local polynomial; smoothing_spline uses lambda and supports first and second derivatives. The estimated standard uncertainty assumes independent measurement errors. CSV paths are relative to the working directory unless absolute, and an existing output_csv target can be overwritten. Extrapolation is disabled by default and permitted only for linear or local_polynomial interpolation.

34 4 Exercises

1. Use PCHIP and natural cubic spline on increasing but nonuniform data and compare overshoot. 2. Add standard deviations as the third element of each inline sample, then report uncertainty at a midpoint. 3. Repeat a local fit at two window sizes to test sensitivity.

34 Interpolation jobs inside a workflow

```
workflow
interpolate line_sample
    data = [[0,0], [1,2], [2,4]]
    method = linear
    at x = 1.5
end
for row in line_sample.result.results
    report row.value
end
```

Expected y = 3 at x = 1.5.

Use at x = @query inside a loop to evaluate a sequence of query coordinates with the same data and interpolation method.

Chapter 35 Finite One Sided and Infinite Limits

A limit is about behaviour near a point, even when direct substitution is undefined. The solver distinguishes finite_limit, positive_infinity, negative_infinity, does_not_exist and undetermined. An undetermined result means insufficient evidence, not mathematical nonexistence. Version 3.0.1 improves numerical limit evaluation. Check sided evidence and the reported classification against a mathematical reference when one is available.

35 1 Removable singularities

Selected from challenging Problem 78

```
limit
expr = (x^2-3*x+2)/(x-1)
at x = 1
```

Reference. Factor $x^2-3x+2=(x-1)(x-2)$. The limit equals −1.

Interpretation. Algebraic cancellation is preferable to sampling extremely close to the singularity.

Selected from challenging Problem 81

```
lim
expr = (2*exp(x)-2)/x
at x = 0
```

Reference. The limit is 2, by $\exp(x)=1+x+O(x^2)$.

Interpretation. Taylor recognition handles cancellation near zero.

35 2 Limits at infinity and subtraction cancellation

Selected from challenging Problem 80

```
limit
expr = x-sqrt(x^2+x)
at x = inf
```

Reference. Rationalize to $-x/(x+\sqrt{x^2+x})$; as $x\to+\infty$, the limit is −1/2.

Interpretation. Direct evaluation at extremely large floating x can produce a false zero because significant digits cancel.

35 3 Two sided limits must respect signs

Selected from challenging Problem 84

```
limit
expr = (sin(x)-cos(x))/(tan(x)-cot(x))
at x = #pi/4
```

Reference. The limit is $1/(2\sqrt{2}) \approx 0.3535533905932738$.

Interpretation. Both numerator and denominator vanish; analytic factorization or a correctly evaluated derivative ratio is needed.

Side-aware radical example

```
limit
expr = (exp(x)-1)/sqrt(1-cos(x))
at x = 0
direction = both
```

Reference. Right-hand limit is $+\sqrt{2}$; left-hand limit is $-\sqrt{2}$. Therefore the two-sided limit does not exist.

Interpretation. The identity sqrt(x²)=abs(x), not x, is decisive. direction = left or right can request an individual side.

35 4 Multivariable path evidence and exercises

```
limit
expr = x^2*y/(x^4+y^2)
at x = 0
at y = 0
```

Along y=0 the value tends to 0, while along y=x² it equals 1/2. This disproves the full limit. Agreement along a finite set of paths cannot prove existence. Exercise: explain why x²sin(1/x) tends to zero whereas sin(1/x) does not; then test both and inspect the evidence method.

35 5 Standalone singularity analysis

Use singularities to investigate where an expression becomes undefined or loses smoothness on a specified interval. Run the listing as shown to investigate the interval. Remove the comment marker // to supply the two candidate points explicitly.

```
singularities
expr = x * asin(2x * sqrt(1-x^2)) / sqrt(1-x^2)
var = x
range = [-1, 1]
// singularities = [-1/sqrt(2), 1/sqrt(2)]
```

Derive the interior candidates

Let g(x) = 2x√(1−x²). The identity 1−g(x)² = (1−2x²)² shows that the arcsine argument stays in [−1, 1] and reaches its endpoints at x = ±1/√2. Near either point, the departure of asin(g(x)) from its extremal angle is proportional to √(1−g(x)²) = |1−2x²|. That absolute value changes slope at each candidate. The prefactor x/√(1−x²) is finite and nonzero there, so the full expression has finite corners.

Check the endpoints separately

At x = ±1 the denominator vanishes. As x approaches either endpoint from inside the interval, g(x) tends to zero. Since asin(g)/g tends to 1, the expression approaches the same limit as 2x², namely 2. The original formula is undefined at each endpoint and has a finite inward limit. Compare this behaviour with the two finite interior corners in the report.

Use the analysis to guide inspection and the choice of quadrature splits. A candidate classification describes local behaviour. Assess the convergence and acceptance of the integral separately.

35 Limit and singularity jobs inside a workflow

```
workflow
limit sinc_limit
    expr = sin(x)/x
    at x = 0
end
report sinc_limit.value
```

Expected 1.

Use a separate singularities job to classify candidates before selecting one-sided limit tests.

```
workflow
singularities pole_scan
    expr = 1/(x-0.25)
    var = x
    range = [0,1]
end
for point in pole_scan.result.points
    report point.classification
end
```

Expected infinite_singularity for a pole at x = 0.25.

Chapter 36 Compute Functions and Special Functions

Compute Functions is a self-contained mathematical service. It accepts multiple named calculations, supports previously defined scalar values and supplies a Function Explorer. Its extra special functions are intentionally not injected into all equation and differential-equation grammars.

36 1 Multiple assignments and 113 bit arithmetic

Selected from challenging Problem 88

```
compute
precision = binary128
a = gamma(5)
b = erf(1)
c = besselj(0,2)
d = lambertw(5)
e = zeta(2)
f = a+b+c
```

Reference. gamma(5)=24; zeta(2)=$\pi^2/6$; Bessel $J_0(2)\approx0.2238907791412357$. Other entries are real special-function values.

Interpretation. cpp_bin_float_quad offers 113 significant binary bits, approximately 34 decimal digits of working precision. Actual accurate digits depend on conditioning and implementation; results are not formal certificates.

36 2 Function Explorer

Selected from challenging Problem 89

```
compute
function = besselj(0,x)
range = [0,20]
points = 500
```

Reference. At x=0, $J_0(0)=1$. Later sampled values oscillate with decaying envelope; the plot should display this structure.

Interpretation. The Explorer allows 2–20,000 samples including endpoints. Invalid sample points become null entries with status; they are not silently promoted to zero.

36 3 Function families and conventions

The Compute catalogue includes elementary and inverse trigonometric/hyperbolic functions, erf/erfc, gamma/lgamma/beta/digamma/polygamma, Bessel J/Y/I/K, Airy Ai/Bi, Legendre/Hermite/Laguerre/Chebyshev polynomials, zeta, Lambert W, Ei/Si/Ci and elliptic integrals. Bessel uses argument order (order,x). Lambert W uses the principal real branch by default; a branch −1 selects the alternate real branch where valid. Elliptic m means k^2. Real-domain errors should be explicit rather than converted to plausible numbers. Appendix D gives every function individually, with argument count, a brief description, domain and example. See Studies 40 17 and 40 18 for complete identity and exact-integer workflows.

36 4 Exercises

1. Verify erf(1)+erfc(1)=1 within reported precision. 2. Compute gamma(1/2)² and compare with π. 3. Explore erf(x) on [−3,3] with 301 points and locate its odd symmetry. 4. Test a function outside its real domain and record the error status.

36 Compute functions inside a workflow

```
workflow
compute function_value
    expr = sin(#pi/2) + sqrt(36)
end
for row in function_value.result.results
    report row.value
end
```

Expected 7.

All mathematical functions, including every function available in Compute, are available to Version 4.0 workflows. Call shared core and scalar Compute functions directly in workflow expressions. Use a compute job for list-valued Compute output such as primefactors. Inside a service job, the service's own expression grammar applies. To use a Compute scalar in another service, evaluate it in the workflow and substitute the numeric result with @name.

Chapter 37 Number Theory Functions and Exact Integers

The number-theory helpers operate in standalone Compute mode. Their input bounds are performance contracts for this release; they do not limit the underlying mathematics. Integer-valued results should be preserved as exact decimal strings when sent through interfaces that might otherwise round large integers.

37 1 Euler totient and prime factors

A counting problem related to the supplied integer collection

```
compute
precision = binary128
a = totient(5929)
b = primefactors(5929)
c = primefactor(5929,3)
```

Reference. $5929=7^2\cdot11^2$, so $\phi(5929)=5929(1-1/7)(1-1/11)=4620$. primefactors returns [7,7,11,11] and primefactor(5929,3)=11.

Interpretation. The list from primefactors is not a scalar expression. Use primefactor(n,k) for one 1-based factor.

37 2 Prime counting and deliberate limits

```
compute
a = primepi(1000000)
b = primepi(100000000)
c = mobius(30)
d = divisorcount(360)
e = divisorsum(28)
```

Independent references: $\pi(10^6)=78{,}498$; $\pi(10^8)=5{,}761{,}455$; $\mu(30)=-1$; divisorcount(360)=24; divisorsum(28)=56. primepi accepts $0\le n\le100{,}000{,}000$ inclusive. The other number-theory functions accept integer arguments up to 10^{12} within their individual positive/nonnegative domains. A larger request should fail promptly with a range diagnostic.

37 3 Exercises

1. Calculate $\phi(77)$ and verify it by counting $1\le k\le77$ with $\gcd(k,77)=1$. 2. Factor 360 and use the product formula to reproduce divisorcount(360). 3. Explain why primepi(10^8+1) is a deliberate unsupported request in this release rather than a claim about prime numbers.

37 Number theory functions inside a workflow

```
workflow
compute factor_list
    factors = primefactors(5929)
end
for row in factor_list.result.results
    report row.value
end
report primefactor(5929, 3)
```

The Compute job returns the formatted text "[7, 7, 11, 11]" for the factor list. The scalar primefactor call returns 11. To build a workflow list for further processing, call primefactor for each index and append the result.

For scalar number-theory calls, use the function directly in workflow expressions. For a collection of factors, use a Compute job or construct a workflow list from indexed primefactor calls.

Chapter 38 Linear Diophantine Families and Modular Results

The integer linear equation ax+by=c is exactly solvable by the Extended Euclidean Algorithm. Its answer may be no integer pair, a finite set under bounds or an infinite integer family. The solver can report a modular congruence together with the dependent-variable formula.

38 1 General integer solution

Exact linear model

```
int x, y
7*x + 11*y = 100
```

Reference. One complete family is x=8+11t, y=4−7t for arbitrary integer t. Equivalently x ≡ 8 (mod 11) and y=(100−7x)/11.

Interpretation. Reporting only the congruence on x would omit the second coordinate. The displayed representative pair is not the complete infinite result.

38 2 GCD obstruction and bounded families

No-solution example

```
int x, y
6*x + 9*y = 20
```

Reference. gcd(6,9)=3 does not divide 20; no integers satisfy the equation.

Interpretation. This is a completeness proof, not a failed numerical search.

Seven bounded solutions

```
int x, y
3*x + 5*y = 100
bound x = [0,100]
bound y = [0,100]
```

Reference. The seven pairs are (0,20), (5,17), (10,14), (15,11), (20,8), (25,5) and (30,2).

Interpretation. Bound constraints restrict the free integer parameter to a finite interval; the exact count can exceed the display-list limit without losing mathematical completeness.

38 3 Large coefficients and timing

```
int x, y
123456789*123456789*x - 987654321*y = 1359
```

This is a useful performance illustration from the development conversation. The exact solver computes a gcd, a particular solution and its parametric family without a floating-point root search. Timings below a millisecond should use useful microsecond/nanosecond display and should not be presented as a deterministic benchmark.

38 4 Exercises

1. State the solvability condition for 14x+21y=35 and parameterize all integer pairs. 2. Restrict x,y≥0 and count the feasible pairs. 3. Compare declaration int x,y against pos int x,y; explain the difference at zero.

Chapter 39 Integrated Research Challenges and Release Validation

The complete research workflow couples a mathematical formulation with a reproducible input, an independent reference, engine diagnostics and a platform build identity. The attached 100-challenge collection has 50 established benchmark cases followed by 50 later cases, and the equation archive contains hundreds of standalone model entries. These are a broad regression corpus, not a claim that every model is already accepted by an MSIX release.

39 1 A short cross-domain study sequence

Study A. Solve the exact linear integer example in Chapter 38, verify the gcd condition and record cardinality.

Study B. Use the quadratic measurements in Chapter 34 to calculate derivatives and interpolated midpoint values; then perturb one observation and compare sensitivity.

Study C. Compare the rationalized infinite limit of Chapter 35 with naive evaluation at $x=10^{10}$ and explain cancellation.

Study D. Use Compute to compare zeta(2) with $\pi^2/6$, and compare displayed working precision with actual residual.

Study E. Reproduce one Chapter 31 difficult improper quadrature case and distinguish accepted value, error estimate and interruption status.

39 2 Independent quadrature examples from the challenge collection

Selected from challenging Problem 95

```
quad
expr = 1/(1+x)^(5/4)
var = x
interval = [0,inf]
```

Reference. The integral equals 4, by substituting u=1+x and integrating u^(-5/4).

Interpretation. The aliases quad/quadrature, expr/integrand and var/variable are mode-specific. The mathematical improper tail must be controlled numerically.

Selected from challenging Problem 94

```
quadrature
integrand = exp(-x^2)*cos(10*x)
variable = x
interval = [-inf,inf]
```

Reference. The exact value is sqrt(π) exp(−25).

Interpretation. This Fourier–Gaussian integral tests a small result arising from cancellation of oscillations. A raw error claim should be compared with the result magnitude.

Selected from challenging Problem 99 — regression caveat

```
quadrature
integrand = x/sqrt(1-sqrt(x))
variable = x
interval = [0,1]
```

Reference. The exact value is 64/35.

Interpretation. A prior Windows run reported nonfinite_integrand near the endpoint. This example remains a mandatory acceptance test; its presence here does not assert that the final packaged Windows solver has passed it.

39 3 Release validation checklist

Freeze the Version 3.0.1 source and documentation baseline, rebuild all Windows executables and DLLs with Boost 1.83 headers, and run targeted differentiation, interpolation, limits, Compute, number theory, Diophantine and legacy suites. Verify on a clean MSIX installation with CSV paths, Desktop charts, licence controls, JSON precision metadata, runtime DLLs and package signatures. Preserve accepted references separately from estimated values and mark every untested scenario honestly.

Chapter 40 Choosing a Practical University Workflow

The quickest productive route through this book is to start with the result you need. A set of equilibrium coordinates, a time history, a spatial field, a derivative and a special-function table are different outputs. Choose the corresponding service before adjusting search settings. The examples in this chapter are workflows for teaching and research rather than a sequence of mathematical prerequisites.

40 1 Start with a question that can be checked

Write a short statement of the task beside the model. For an equilibrium calculation, state which operating conditions are fixed and which concentrations or temperatures are unknown. For a differential equation, identify the independent variable, the initial or boundary data, and the interval. For measured data, identify units and whether the observations include standard uncertainties. This makes a saved model understandable to another student or to your future self.

A useful working note has four sentences: what quantity is needed, what equations define it, what domain is admissible, and what independent observation would make the answer credible. For example, a robot positioning calculation needs angles within mechanical limits; substituting them into the end-point coordinates checks the position, while an additional orientation equation checks the tool direction. A small residual in the position alone does not establish the full task.

No user-supplied initial guess is required for ordinary algebraic root discovery. Search bounds are still useful information: they specify the domain in which roots matter and can reduce wasted work. Multiple returned rows represent candidate alternatives. Unless an exact route or an independent argument establishes completeness, a numerical search does not prove that every admissible root has been found.

40 2 Match the task to the service

Use ordinary equation input for simultaneous algebraic relationships. Add integer declarations when the unknowns are counts, divisors or congruence representatives. Use ode for a trajectory determined by initial data, bvp for a supported two-point problem, and pde for a supported field problem. Use integral when the unknown function occurs under an integral sign; use quadrature when the integrand is already known.

Use diff or differentiate for a derivative at specified coordinates, and use tabular differentiation when the information is a sample table rather than a formula. Use interpolate to reconstruct values between observations. Use limit when the question concerns approach to a point or infinity. Use compute or func for sequential named numerical calculations and the special-function catalogue in Appendix D. Use singularities to investigate undefined or nonsmooth points on an interval before choosing quadrature splits.

These services have different parsers, numerical algorithms and output contracts. A successful Compute evaluation of a special function does not establish that the same function can be placed inside every ODE or algebraic expression. Appendix D includes the availability matrix; consult it before designing a workflow around a function name.

40 3 Make the first run deliberately small

Begin with a model whose reference is known. For example, solve x+y=3 and x*y=2, then compare the two ordered pairs (1,2) and (2,1). For an ODE, start with a linear equation having an exact solution. For interpolation, use a quadratic table before introducing experimental measurements. This checks the route from model text to result without mixing interface uncertainty with a difficult scientific problem.

After this first success, change one feature at a time. Introduce the intended parameter, larger domain, extra component or measured dataset. Save the successful intermediate model. If a later calculation fails, the difference between the two files often identifies the cause much faster than a long error report.

Keep the output requests modest during exploration. A dense display grid does not necessarily improve the internal numerical solution; it may only request more samples. Refine the numerical controls that govern the method and compare the quantity of interest at common coordinates.

40 4 Read four separate layers of evidence

First read the execution status. Did the calculation finish, reach a budget, or reject the input? Next read the mathematical output: roots, a trajectory, a field, a derivative, a limit classification or a table. Then read diagnostics such as residuals, estimated errors or refinement history. Finally compare an independent reference or structural identity.

An attractive plot belongs to the second layer. It cannot replace the first, third or fourth. A narrow oscillation can fall between plot samples; an interrupted search can still display convincing candidate roots; a smooth interpolation curve can hide measurement uncertainty. Preserve the full report and exported values alongside the figure.

Use the result label exactly. A candidate or estimated value is not automatically an accepted result. An undetermined limit is not proof that the limit does not exist. A failed bounded root search is not an infeasibility proof. An error estimate is conditional on the numerical method and the function behaviour it has resolved.

40 5 A route for professors

Choose one short example with at least two plausible interpretations. Let students predict the possible solutions, run the model, and explain each returned branch. The robot example in Chapter 41 works well because different elbow configurations can reach the same point. A polynomial with a repeated root works well for discussing residual versus coordinate accuracy.

For a one-hour laboratory, allocate the opening period to formulation and prediction, the middle period to execution and checking, and the final period to a controlled change. Ask each group to submit the model, the relevant exported rows, one independent check and a paragraph explaining the change. Assess the reasoning and the reproducible record rather than the number of screenshots.

For a demonstration, keep a verified small model ready. If a difficult run exceeds its budget, discuss the diagnostic and return to the small model. This turns an unsuccessful calculation into a lesson about numerical evidence without presenting an unverified value as the answer.

40 6 A route for research scholars

Begin with a reference model of the same mathematical structure as the research calculation. A manufactured Poisson solution tests boundary handling and grid accuracy. A separable Fredholm kernel tests conditioning. A known special-function identity tests argument conventions. Only then substitute the research data.

Define a quantity of interest before refinement: the peak displacement, one integral, a threshold time, the first eigenvalue, or the number of feasible integer configurations. Compare that quantity across tolerances, grids and formulations. A maximum error over all samples answers a different question from an error in one scalar quantity; record both when they matter.

Separate numerical verification from scientific validation. Refinement can show that a discretisation is approaching a stable answer for the entered model. It cannot show that the constitutive law, parameter estimates or boundary measurements describe the experiment. A thesis should explain both kinds of evidence.

40 7 A route for students

Work through one complete input before changing its syntax. Copy the full listing, including the service header, domain and initial or boundary conditions. After the first run, predict the effect of one change and test the prediction. This is more informative than entering several unrelated formulas and collecting answers.

When a result is surprising, first check variable meanings, radians, multiplication symbols, signs and domains. Then check the mathematical formulation. Increase numerical effort only after these checks. More starts cannot repair a missing conservation equation, and more precision cannot repair a value entered in the wrong unit.

Keep a short learning record with the original question, your prediction, the model, the result and the explanation. At the end of a topic, select one successful example and one failed attempt that taught you something. Together they provide a stronger record of understanding than a folder of unexplained output images.

40 8 Use the example collection as a starting point

A .pdeq file is a project container. Its model text and saved settings belong together, but the settings may reflect the author's particular experiment. Read both before assuming that its time budget or bounds are suitable for your machine or question. A text file may contain several examples or explanatory material rather than one runnable model.

The supplied collection also includes result CSVs, archives and an executable. Count mathematical models rather than treating every archive entry as a separate example. This book selects cases by purpose. The archive is a source of test ideas; it is not evidence that every file is syntactically valid or that every saved result belongs to the latest engine.

For portable models, replace machine-specific CSV paths with paths in your own working folder. Preserve the source filename in your notes. Record any correction, including missing commas or explicit multiplication. The corrected teaching listings in Chapter 41 are intentionally written with explicit multiplication and complete list separators.

Chapter 41 Worked University Laboratory Studies

Each study below gives a purpose, a complete input, a mathematical reference or check, and a useful follow-up. The references are mathematical checks, not claims of a fresh Windows regression. Run the model in your installed Version 3.0.1 and retain its status and full output. Use the study to learn the workflow before applying it to a less controlled research model.

41 1 Robot positioning with two geometric branches

A three-link planar robot has lengths 2, 1.5 and 1. Its tool must reach (2.5,1.5) with the last link pointing upward. The source is difficult_equation02.pdeq. Angles are in radians. The third link contributes (0,1), so the first two links must reach (2.5,0.5). This reduction predicts two elbow configurations before the solver runs.

```
2*cos(q1)+1.5*cos(q1+q2)+cos(q1+q2+q3)=2.5
2*sin(q1)+1.5*sin(q1+q2)+sin(q1+q2+q3)=1.5
q1+q2+q3=#pi/2
bound q1=[-#pi/2,150*#pi/180]
bound q2=[-150*#pi/180,150*#pi/180]
bound q3=[-150*#pi/180,150*#pi/180]
```

The cosine rule gives cos(q2)=1/24. For each sign of q2, use q1=atan2(0.5,2.5)-atan2(1.5*sin(q2),2+1.5*cos(q2)), and then q3=pi/2-q1-q2. Both configurations satisfy the stated limits. The two-argument arctangent here is an independent geometric formula, not an additional solver directive.

Check both endpoint coordinates and the angle sum using unrounded exported angles. For discussion, tighten the third joint limit until one branch becomes mechanically inadmissible. Explain why discovering two mathematical branches does not decide which movement is collision-free; the model contains no obstacle geometry.

41 2 A jacketed reactor with physical admissibility

The source 01_exothermic_reactor.txt couples conversion X, reaction rate k, reactor temperature T and jacket temperature Tj. Its numerical constants describe an illustrative model. They are not calibrated plant data. Bounds encode the operating region of interest.

```
X=k*(1-X)
k=exp(29-10000/T)
T=300+200*X-0.5*(T-Tj)
Tj=290+0.2*(T-Tj)
bound X=[0,1]
bound T=[290,450]
bound Tj=[290,330]
bound k=[0.0001,1000]
```

Two equations can be eliminated exactly: Tj=(290+0.2*T)/1.2 and T=(2525+1200*X)/8.5. The concentration balance also gives k=X/(1-X), requiring X<1 for a finite positive rate. Substitution leaves one scalar temperature-rate consistency condition. Use it as an independent residual check on every returned branch.

Do not infer dynamic stability from the existence of a steady state. Stability requires a time-dependent model and a local or nonlinear analysis. For a parameter study, change the illustrative heat-generation coefficient 200 in separate saved models, then track branches by proximity and physical interpretation rather than row number.

41 3 An exact bounded integer family

A count model can have a finite complete answer even when its unconstrained version has infinitely many solutions. This study extends the linear Diophantine workflow in Chapter 38 and is a useful control case for teaching exact completeness.

```
int x,y
3*x+5*y=100
bound x=[0,100]
bound y=[0,100]
```

Reduction modulo 5 gives x=5*t. Substitution gives y=20-3*t. The nonnegative bounds require integer t from 0 through 6, so the complete answer contains seven ordered pairs. Direct substitution checks every pair exactly; the parameter bounds establish completeness independently of the displayed row count.

Change the first bound to [1,100] and predict which pair disappears. Then change the declarations to positive integers and inspect how zero is handled. When a display truncates a large family, distinguish the mathematical family representation from an explicitly enumerated table.

41 4 A trajectory with a closed reference

The source ode02.pdeq provides an uncomplicated ODE for checking the complete trajectory workflow. It is useful before experimenting with nonlinear or stiff models.

```
ode
y'=x+y
initial x=0
initial y=1
interval x=[0,1]
max step=0.1
initial step=0.1
output points=11
rtol=1e-7
atol=1e-10
```

Multiplying y'-y=x by exp(-x) and integrating gives y=2*exp(x)-x-1. In particular, y(0)=1 and y(1)=2*e-2. Compare all exported values at their reported x coordinates, rather than comparing rounded plot positions. Record the maximum absolute sample error.

Repeat with a smaller maximum step and tighter tolerances, changing one control at a time. Compare the same eleven coordinates. Explain why increasing output points alone need not force smaller internal integration steps and why local tolerances are not a formal global error bound.

41 5 Logistic growth and a threshold time

The source ode05.pdeq models a population with positive growth and finite capacity. The initial population is far below capacity, so early growth is nearly exponential and later growth slows.

```
ode
constant rate=0.7
constant capacity=1000
population'=rate*population*(1-population/capacity)
initial t=0
initial population=10
interval t=[0,15]
rtol=1e-8
atol=1e-9
output points=301
```

The exact reference is population(t)=1000/(1+99*exp(-0.7*t)). It is increasing and lies strictly between 0 and 1000 on this interval. Half capacity occurs at t=log(99)/0.7. These provide three checks: pointwise values, qualitative bounds and one meaningful threshold.

Estimate the threshold from exported samples and compare it with the analytic time. State the uncertainty due to locating a crossing between samples. A sampled crossing is not an automatically detected event unless an event feature was explicitly configured. Repeat with a different capacity while keeping the initial population fixed, and explain why both the limiting value and threshold change.

41 6 Coupled oscillation with an invariant

The source ode08.pdeq has a growing oscillatory solution. It checks variable ordering and phase as well as amplitude. A correct-looking oscillation can still have the wrong sign in one component.

```
ode
x'=x+y
y'=-x+y
initial t=0
initial x=1
initial y=0
interval t=[0,1]
method=RK45
output step=0.1
rtol=1e-8
atol=1e-10
```

The reference is x=exp(t)*cos(t), y=-exp(t)*sin(t). Therefore exp(-2*t)*(x^2+y^2)=1. Compare the two component errors and the invariant defect separately. A small invariant defect alone cannot detect every phase error.

For a longer interval, raw component errors grow with exp(t). Report scaled errors as well as absolute errors. Ask students to explain the negative sign of y from y'(0)=-1 before running the model. This simple prediction catches a surprisingly common interpretation error.

41 7 A two point boundary value problem

The source bvp01.pdeq is a reference problem for an endpoint-constrained curve. Unlike an IVP, the initial slope is not supplied by the user; the two endpoint values define the required solution.

```
bvp
y''=-2
interval x=[0,1]
boundary left y=0
boundary right y=0
output step=0.1
```

Twice integrating gives y=-x^2+A*x+B. The endpoint conditions give A=1 and B=0, so y=x*(1-x). The maximum is 1/4 at x=1/2 and the slopes at the endpoints are +1 and -1. Check the exported values and endpoint residuals.

A midpoint value is a useful scalar quantity of interest, but it does not check the entire curve. Change the forcing to another expression whose solution can be manufactured, or change one endpoint value and derive the new linear correction. Keep BVP controls distinct from algebraic multistart settings.

41 8 A manufactured Poisson field

The source pde01.pdeq solves a field on the unit square with zero values on all edges. It provides a controlled setting for assessing spatial discretisation.

```
pde
type=poisson
u_xx+u_yy=-2*pi^2*sin(pi*x)*sin(pi*y)
domain x=[0,1]
domain y=[0,1]
boundary left u=0
boundary right u=0
boundary bottom u=0
boundary top u=0
grid x=33
grid y=33
```

The exact solution is u=sin(pi*x)*sin(pi*y). Its Laplacian is the entered forcing and it satisfies every edge value. The centre value is 1. Compare the numerical field with this expression at grid nodes, and record both a maximum nodal error and the centre error.

Repeat with 17, 33 and 65 nodes in each direction. These grids halve the spacing and share nodes. If the leading error behaves like C*h^2, successive errors should fall by about four. Treat that ratio as an observed result to investigate, rather than assigning it in advance. Near-zero discrete residuals alone do not measure continuum error.

41 9 A standing wave and time resolution

The source pde02.pdeq begins with one sine mode and zero initial velocity. It is a useful contrast with the stationary Poisson problem because both space and time affect the numerical approximation.

```
pde
type=wave
u_tt=u_xx
domain x=[0,1]
interval t=[0,1]
boundary left u=0
boundary right u=0
initial u=sin(pi*x)
initial u_t=0
grid x=41
grid t=101
```

The exact reference is u(x,t)=sin(pi*x)*cos(pi*t). At t=1/2 the entire field is zero; at t=1 its sign is reversed. Check these features and a point with nonzero amplitude, since the zero snapshot alone can hide amplitude error.

Refine the time grid with the spatial grid fixed, then refine space with time sufficiently resolved. State which change controls the error you observe. For schemes with a stability restriction, respect the documented relation between time and space steps; increasing spatial resolution alone can make a previously stable setup inappropriate.

41 10 A Volterra equation with an exact integral check

The source integral02.pdeq specifies a second-kind equation y(x)=f(x)+integral from 0 to x of K(x,s)*y(s) ds. The triangular integration domain distinguishes it from a Fredholm equation with fixed limits.

```
integral
type=volterra
interval x=[0,1]
kernel=(x-s)^2
forcing=1+x-x^3/3-x^4/12
grid x=129
exact solution=1+x
```

For y(s)=1+s, the kernel integral is x^3/3+x^4/12, so adding the forcing gives 1+x. This identity checks the sign convention and supplies the exact function independently of the numerical output.

Compare 65, 129 and 257 nodes at common coordinates. At x=0 the integral vanishes, so y(0)=f(0)=1 is a particularly simple check. Retain the exact solution directive as a comparison aid; it is not an additional constraint that should replace solving the integral equation.

41 11 A nearly singular Fredholm operator

The source integral05.pdeq uses a separable kernel. It demonstrates how sensitivity can dominate even when the exact solution is simple.

```
integral
type=fredholm
interval x=[0,1]
kernel=x*s
lambda=2.95
forcing=1+x-2.95*5*x/6
absolute tolerance=1e-7
relative tolerance=1e-7
grid x=65
exact solution=1+x
```

For y=1+x, the moment M=integral from 0 to 1 of s*y(s) ds is 5/6. In general, y=f+lambda*x*M implies (1-lambda/3)*M=integral from 0 to 1 of s*f(s) ds. At lambda=2.95 the denominator is 1/60. Perturbations in the forcing moment can therefore be amplified strongly.

Compare a refinement change with a deliberate small perturbation to the forcing. These are different experiments: one probes discretisation and one probes conditioning. At lambda=3 the operator is singular; changing lambda to exactly 3 is not merely a harder version of the same well-conditioned problem. Analyse compatibility before interpreting a failure.

41 12 A full multivariable sensitivity report

This expands diff01.pdeq and challenging Problem 61. It is useful when an output depends on several parameters and one needs both local slopes and curvature.

```
differentiate
expression=x^2*exp(y)+sin(x*z)+y^3*z
at x=1
at y=0
at z=2
gradient=true
hessian=true
```

Let s=sin(2) and c=cos(2). The gradient at the point is (2+2*c,1,c). In variable order (x,y,z), the Hessian rows are (2-4*s,2,c-2*s), (2,1,0), and (c-2*s,0,-s). Derive these entries from the original expression before comparing them with the numerical report.

Symmetry of mixed partials is one useful check. A directional check is stronger: for a small vector delta, compare the actual function change with gradient dot delta plus one half delta transpose times Hessian times delta. Reduce the size of delta and observe whether the remainder decreases as expected for a smooth function. Very tiny perturbations eventually expose floating-point cancellation.

41 13 Differentiate observations rather than a formula

The quadratic table from diff-table01.pdeq is an exact-data control. It isolates the tabular workflow before measurement noise is introduced.

```
diff
data=[[0,0],[1,1],[2,4],[3,9],[4,16],[5,25]]
at x=[1.5,2.5,3.5]
order=1
method=auto
```

The exact reference slopes are 3, 5 and 7. Now change the observation at x=3 from 9 to 9.01 in a second model. Compare the change in the derivative at 2.5 and 3.5. Differentiation magnifies local data perturbations; more arithmetic precision does not remove that sensitivity.

For research measurements, keep the original samples and their units. Use an uncertainty column only when its values represent standard uncertainties under the method's assumptions. Report the selected window, polynomial degree or smoothing penalty. An uncertainty propagated from observation errors does not automatically cover model bias from the choice of smoothing curve.

41 14 Compare interpolation assumptions explicitly

The source interpolate01.pdeq uses the same quadratic table. The following version states a local polynomial method so the intended reproduction property is clear.

```
interpolate
data=[[0,0],[1,1],[2,4],[3,9],[4,16],[5,25]]
at x=[1.5,2.5,3.5]
method=local_polynomial
```

The reference values are 2.25, 6.25 and 12.25. Repeat with method=linear. Linear interpolation instead gives 2.5, 6.5 and 12.5, because each midpoint lies halfway between two observations on a secant. Both methods interpolate the samples; they make different assumptions between them.

Use pchip on monotone measured data when shape preservation matters, and compare it with cubic_spline. Do not promise that a natural cubic spline reproduces a quadratic everywhere: its natural endpoint curvature condition can conflict with that polynomial. Extrapolation requires separate justification and is disabled by default; current support is limited to the documented linear or local polynomial routes.

41 15 A limit with different answers on the two sides

The sign-sensitive radical example in Chapter 35 is a useful exercise in reading classifications rather than just numbers.

```
limit
expr=(exp(x)-1)/sqrt(1-cos(x))
at x=0
direction=both
```

Near zero, exp(x)-1 behaves like x and sqrt(1-cos(x)) behaves like abs(x)/sqrt(2). The right-hand limit is sqrt(2), while the left-hand limit is -sqrt(2). The two-sided limit therefore does not exist. Run left and right requests separately and compare the evidence.

The conclusion comes from the sign analysis, not from a finite sampling grid. For a multivariable expression, two paths with different limiting values disprove a common limit; agreement along selected paths cannot prove one exists. Preserve an undetermined status when the numerical evidence cannot settle the question.

41 16 An improper integral with a slow tail

Challenging Problem 95 is a clean test of tail handling. It has a simple exact answer but converges slowly under direct truncation of the original variable.

```
quad
expr=1/(1+x)^(5/4)
var=x
interval=[0,inf]
```

The exact integral is 4. The tail beyond a finite cutoff A is exactly 4*(1+A)^(-1/4). At A=10^8 the remaining contribution is still close to 0.04. This explains why a very large-looking finite endpoint can leave a visible error.

Compare the solver's accepted result and error estimate with the exact value. If the run stops early, record the status and any candidate separately. A finite cutoff experiment is useful only when the tail is bounded explicitly. The transformed improper-integral method and a direct truncated integral need not have comparable workloads.

41 17 Special functions with independent identities

This extends compute01.pdeq and the Compute reference. The sequential assignments calculate several identities without requiring the general equation solver to understand Compute-only functions.

```
compute
precision=binary128
g=gamma(0.5)
gamma_identity=g*g-#pi
a=erf(1)
b=erfc(1)
erf_identity=a+b-1
z=zeta(2)
zeta_identity=z-#pi^2/6
w=lambertw(5)
w_identity=w*exp(w)-5
```

Each identity should be close to zero in the selected arithmetic. Compare the magnitude with the scale of its terms. These checks can expose an argument or branch mistake, but related implementations can share errors; they are not independent certification of all digits.

For a plot, start a separate Function Explorer model with function=besselj(0,x), range=[0,20] and points=500. Check J0(0)=1. Export the samples before using them in a tabular workflow. A sampled plot is not an exact zero finder and can miss a zero between samples.

41 18 Exact arithmetic with scalar and list results

The updated Compute reference provides ten exact number-theory utilities. Their documented argument ceilings are part of Version 3.0.1's service contract.

```
compute
n=5929
phi=totient(n)
factors=primefactors(n)
third=primefactor(n,3)
g=gcd(n,77)
tau=divisorcount(360)
sigma=divisorsum(28)
mu=mobius(30)
p=isprime(97)
count=primepi(1000000)
multiple=lcm(14,22)
```

The references are phi=4620, factors=[7,7,11,11], third=11, g=77, tau=24, sigma=56, mu=-1, p=1, count=78498 and multiple=154. The list from primefactors is structured output; it cannot be added to a scalar. Use primefactor for a selected 1-based element.

Use the factorisation 360=2^3*3^2*5 to derive tau=(3+1)*(2+1)*(1+1)=24. Explain why primepi(100000001) is a deliberate range error in this release. A boundary test is successful when the documented error appears promptly; it should not be reported as an unexpected mathematical failure.

Chapter 42 From an Exploratory Run to a Reproducible Study

A useful research result is a calculation that another reader can reconstruct and assess. This chapter gives practical reporting patterns for the models in Chapter 41. They complement the integration examples in Chapter 24 and the numerical reliability discussion in Chapters 23 and 32.

42 1 Preserve the question and the exact input

Create one folder per study and separate raw data, model files, reports and figures. Keep the source observations unchanged. If data cleaning is necessary, save the transformed dataset separately and record the transformation. A model file alone cannot reproduce a CSV-driven result if the CSV was subsequently edited.

Preserve the complete model text and saved settings. Include any bounds entered through the Desktop that are not obvious from the model. Record the selected number style, arithmetic mode, tolerances, resource budgets and random seed when applicable. Save the product and engine version identity reported by the installed application.

Choose filenames that encode purpose and revision, for example poisson_grid33 and poisson_grid65. A date or run number can distinguish repeated runs. Do not identify branches solely as row 1 or row 2, because discovery order can change. Use coordinates, signs, physical interpretation or proximity to a previous branch instead.

42 2 Define acceptance before looking at the answer

For a reference ODE, choose a maximum sample error over specified times. For an exact integer model, require exact substitution and the expected cardinality or complete family. For a PDE, define a norm and the grid coordinates at which it is evaluated. For an improper integral, choose absolute and relative error requirements that make sense for the result magnitude.

A tolerance of 10^-8 means different things for a quantity of order 1 and one of order 10^-15. State the scale explicitly. If the expected value is zero, relative error alone is unsuitable. If the reference itself is numerical, record its method and precision and avoid treating its last printed digit as exact.

Use a practical three-way outcome: accepted under stated checks, rejected by a stated check, or unresolved because the evidence is insufficient. An interrupted run with a plausible estimate belongs in the last group unless the method supplies enough independent evidence for acceptance. Preserve unresolved runs; they explain the limits of the study.

42 3 Change one numerical factor at a time

A convergence study needs comparable models. Keep the equation, domain, physical parameters and measured data fixed while changing a grid or tolerance. Compare outputs at common coordinates. If the number of output samples changes, interpolate carefully or restrict the comparison to shared nodes, and state which choice you made.

For a spatial study, report the number of nodes and the actual spacing. With endpoint-inclusive nodes on an interval of length L, spacing is L/(N-1), not L/N. Grids with 17, 33 and 65 nodes halve the spacing. Doubling the node count from 32 to 64 does not produce exactly the same ratio.

When exact errors E_h and E_h/2 are available, the observed order is log(E_h/E_h/2)/log(2). Without an exact solution, use successive differences at common coordinates, while acknowledging that common-mode error can make agreement misleading. Ratios become uninformative near roundoff or when error terms cancel.

42 4 Separate sensitivity from convergence

In a sensitivity study, intentionally change a physical parameter or observation and keep numerical errors sufficiently small. In a convergence study, change numerical resolution while keeping the scientific problem fixed. Mixing both changes prevents a clear explanation of why the answer moved.

For the Fredholm study, moving lambda toward 3 changes conditioning. Refining its quadrature grid changes the numerical approximation. For the tabular derivative, perturbing one displacement changes the data; changing the local window changes the estimator. Write separate conclusions for these experiments.

A finite-difference parameter sensitivity can be estimated from outputs at p+delta and p-delta. Halve delta to test stability, but avoid steps so small that numerical solution noise dominates the output difference. Scale the result as p/Q times dQ/dp only when p and Q are nonzero and that dimensionless interpretation is useful.

42 5 Prepare figures that carry numerical meaning

Give axes names and units. State whether the curve is an analytic reference, a line connecting numerical samples or a fitted interpolation. Put the numerical method and resolution in the caption when they affect interpretation. A figure exported from the Desktop and a separately generated analytic figure have different evidential roles.

For trajectories, overlay the reference and add an error plot when the curves are visually indistinguishable. For fields, use a consistent colour scale when comparing grids or parameters. For multiple roots, identify branches with a legend and preserve the domain. For a sampled special function, mark undefined samples as gaps rather than zero.

Do not choose a logarithmic error axis without considering exact zeros and sign. Use absolute error for a nonnegative magnitude plot, or a signed linear plot when direction matters. Retain the numeric export used for the figure so a reader can inspect values hidden by visual resolution.

42 6 Diagnose the input before increasing effort

If syntax checking fails, inspect the first diagnostic and the nearest statement. Check explicit multiplication, balanced brackets, commas in data rows and the expected mode header. The supplied archive includes some informal or incomplete inputs; copying a filename does not make its contents a validated model.

If the model parses but produces nonfinite evaluations, check logarithms, square roots, denominators and finite-domain special functions. Restrict bounds only when the scientific problem justifies the restriction. If two algebraically equivalent forms behave differently, retain both and investigate cancellation, overflow and branch conventions.

If the model runs but produces no candidate roots, verify the number of independent equations, domain feasibility and search budget. For an exact integer equation, apply a gcd or congruence obstruction where possible. For continuous models, an empty numerical result is generally evidence about the search, rather than a proof about all real solutions.

42 7 Diagnose numerical disagreement by its pattern

A nearly constant offset suggests a boundary or initial-data mismatch. A phase drift suggests time discretisation or an incorrect sign or frequency. Errors concentrated near endpoints can arise from singular behaviour, boundary approximations or interpolation assumptions. Oscillations on a refined transport grid can indicate a method limitation rather than a need for still more samples.

Unexpected integer results require a different investigation. Check the declaration, sign convention, bounds and exact decimal values. Do not round a noninteger floating root and call it an exact solution. Substitute the proposed integers into the original identities without floating-point conversion.

If only displayed digits differ, inspect the full export and arithmetic metadata. A plot uses display arithmetic and may not retain every engine digit. Conversely, a long decimal string does not establish a long sequence of correct digits. Use residuals, references and conditioning to decide what the digits support.

42 8 A compact report for a dissertation or laboratory

Begin with the scientific question and the mathematical formulation. Give the model file and required data, then state the service, installed version and numerical controls. Present the quantity of interest, the independent check and the refinement or sensitivity evidence. Finish with the limits of the conclusion.

A suitable results sentence is: On the unit-square manufactured Poisson model, the reported maximum nodal error decreased across the three stated grids, with the observed ratios listed in the accompanying table. Fill in measured values from your own runs. Avoid converting this one observation into a claim about every PDE or every installation.

For a root search, report the admissible domain, number of distinct candidates found, verification residuals and the basis for any completeness claim. For a limit, report classification and side or path evidence. For Compute, report branch conventions and identities checked. These details let a reader judge the actual result without requiring the full application interface.

42 9 Choose a manageable research extension

A good extension changes one mathematical difficulty while retaining a reference. Move a Fredholm parameter toward singularity, increase oscillation in a quadrature integrand, add controlled noise to tabular data, or follow algebraic branches as a parameter changes. State the expected failure mechanism before running the experiment.

A larger model is not automatically a stronger research contribution. A carefully measured boundary between accurate and unreliable results can be more useful than a large collection of unexplained timings. Report unsuccessful cases with the same care as successful ones and retain enough information to reproduce them.

Chapter 43 A Complete Multi Service Workflow

Use a shared coefficient across an algebraic solve, definite integration, initial-value ODE, boundary value problem and Poisson PDE. A Compute function supplies the target coefficient, and the algebraic solve passes it to each downstream model.

43 1 Run the full study

```
workflow
// Solve for a shared parameter and use it in four numerical services.
target = gamma(3)
solve parameter
    x = @target
end

report parameter.status
for root in parameter.roots
    a = root.x
    quad area
        expr = @a*x
        var = x
        range = [0,1]
    end
    if area.accepted
        report area.value
    else
        report area.status
    end

    ode decay
        y' = -@a*y
        initial t = 0
        initial y = 1
        interval t = [0,1]
        output points = 11
    end
    report decay.status

    bvp beam
        y'' = -@a
        interval x = [0,1]
        boundary left y = 0
        boundary right y = 0
        output step = 0.1
    end
    report beam.status

    pde plate
        type = poisson
        u_xx+u_yy = -@a*#pi^2*sin(#pi*x)*sin(#pi*y)
        domain x = [0,1]
        domain y = [0,1]
        boundary left u = 0
        boundary right u = 0
        boundary bottom u = 0
        boundary top u = 0
        grid x = 17
        grid y = 17
    end
    report plate.status
end
```

43 2 Check the mathematical results

The algebraic root is 2. The accepted integral is 1. The ODE has y(1) = exp(-2), approximately 0.1353352832. The BVP has y(x) = x(1-x), with midpoint 0.25. The Poisson solution is sin(πx) sin(πy), with centre value 1. Compare the field at the centre and repeat on a finer grid to assess discretisation error.

43 3 Inspect and reproduce the run

job.status is the service status. job.accepted is a Boolean that reflects the workflow's acceptance rule for that service. A successful solve can return roots even when a full search has not been established.

job.value is available for accepted scalar results when the service returns a value. A finite limit has a scalar value; an accepted infinite limit is described by its status. Use job.result for the detailed response of the services other than solve and quad. For those two, the full response is still present in Run Report and the CLI's jobs[].result JSON.

Whole workflow. Set Compute → Solver Settings → Time budget, seconds in Desktop or --seconds S in the CLI. The CLI defaults to 20 seconds. A value of 0 disables the overall time budget.

Individual job. within takes a positive finite number or a numerical expression such as within budget/2. A supported job receives the smaller of its cap and the remaining workflow budget. If the overall budget is disabled, the explicit job cap still applies.

Timing precision. A budget is checked cooperatively. The workflow checks it between statements and jobs. Services without per-job support may finish their current operation before the workflow can observe that its budget expired. A capped job can also overrun briefly between internal checks. Give a batch enough total time for the jobs and workflow overhead.

Job failure versus script error. A service failure can be recorded and the script can continue while budget remains. A malformed or nonpositive within value or an invalid workflow expression stops the script with an error. An exhausted overall budget stops the batch.

Workflow limits include 4 MiB of source text, 10,000 input entries, 32 nested blocks, 32 active function calls, 1,000 jobs, 10,000,000 execution steps, and 100,000 items in a range or collection. Range endpoints must lie within ±1 billion. The step budget is shared across the entire run. step_limit preserves reports and jobs recorded before the stop. Recorded results are limited to 8 MiB and tracked collection copying and appending to 32 MiB. For large sweeps, split the run into batches.

Save the script, record the build and number style, and retain the run report or CLI JSON. Check each job independently before comparing against the mathematical references.

Appendix A Desktop command and result reference

This reference covers the current top-level menus. Commands that depend on a result or an idle worker can be disabled until their prerequisites are met. See Section 2.26 for Tools → Equation Generator and the Fun with Mathematics activities.

A 1 File

Command	Action
New	Start a blank model; prompt about unsaved work
Open	Open equations or a project
Open Recent	Reopen a recorded path
Save	Save current model/project
Save As	Choose a new file or format
Import equations	Insert text at the editor selection/cursor
Export results / differential trajectory	Save full returned table as CSV or full result as JSON
Export run report	Save completed report as text
Export ODE events CSV	Save returned event records
Print	Print completed report or current equations
Exit	Close, with unsaved-work and active-task handling

A 2 Edit Search and View

Menu command	Action
Edit > Undo / Redo	Undo or redo editor changes
Edit > Cut / Paste	Operate on editor text
Edit > Copy	Copy from focused Solutions, Run report, Diagnostics or editor as implemented
Edit > Select All	Select all equation-editor text
Edit > Copy selected result cells	Copy current cell or first rectangular result block
Edit > Comment / uncomment	Toggle // on selected lines

Menu command	Action
Edit > Indent selection / Unindent selection	Adjust selected-line indentation
Search > Find	Search the equation editor
Search > Find Next / Find Previous	Repeat with wraparound
Search > Replace	Replace editor text, including Replace All
Search > Go to line	Move editor caret to a line
View > Reset equation and solution fonts	Restore default fonts and save preferences

A 3 Compute

Command	Action
Check Syntax	Validate model syntax/configuration without solving
Variables and Bounds	Parse algebraic variables and edit bound overrides
Solver Settings	Change budgets and algebraic-search preferences
Precision	Choose double or eligible binary128
Number Style	Convert current editor notation to US or EU
Solve	Start the worker using a captured model/settings snapshot
Stop	Force-stop the active solve without a final partial result
Clear Results	Clear displayed result state
View Run Log	Select Diagnostics

A 4 Help and Examples

Help command	Purpose
Quick Start	Brief in-app workflow
User Guide (PDF)	Open bundled PDF in the system viewer
Equation Syntax	Basic syntax reference
Numerical ODE Guide	IVP methods, events and continuous output
Boundary Value Problems Guide	BVP syntax, controls and limits

Help command	Purpose
Large Integer Bounds and Values Guide	Exact literal bounds and intended display controls
PDE and Integral Equations Guide	Supported field and quadrature modes
Singular ODE Points Guide	Supported regular-series continuation
Supported Functions	Expression function list
Examples Guide	Built-in model descriptions
Licence Information	Inspect current entitlement
Activate Licence	Select and install a signed licence file
Machine Fingerprint	Obtain/copy individual machine identifier
Copy Diagnostics	Copy version/OS/worker/licence and task diagnostic text
About	Product version, description, website and attribution

Examples opens built-in category submenus and Additional Equation Library. Choose the category matching the mathematical task. Model-changing actions can be disabled during worker tasks; enabled status indicates which actions are currently available.

Keyboard shortcuts and troubleshooting

A 5 Keyboard reference

Shortcut	Action
Ctrl+N	New
Ctrl+O	Open
Ctrl+S	Save
Ctrl+Shift+S	Save As
Ctrl+P	Print
Alt+F4	Close application
Ctrl+Z / Ctrl+Y	Editor Undo / Redo
Ctrl+X / Ctrl+V	Editor Cut / Paste
Ctrl+C	Copy according to focused supported control
Ctrl+A	Select all editor text; the full-value dialog has its own selection command

Shortcut	Action
Ctrl+Shift+C	Copy selected result cells
Ctrl+/	Comment / uncomment selected editor lines
Ctrl+F	Find in editor
F3 / Shift+F3	Find Next / Find Previous
Ctrl+H	Replace
Ctrl+G	Go to line
F4	Check Syntax
Ctrl+Enter or F5	Solve
Esc	Stop an active solve when the main-window command is active
F1	Quick Start

Dialogs and focused controls can handle keys locally; for example Esc can close a dialog instead of stopping a worker. Use the menu when shortcut focus is uncertain. The full-value dialog has its own Ctrl+A/C handlers, but its visual wrapping remains subject to the known defect.

Original mathematical illustrations

The following four illustrations are preserved from the supplied complete book. They are mathematical figures rather than new Desktop screenshots. Their original captions identify the problem and should be retained when producing the final illustrated edition.

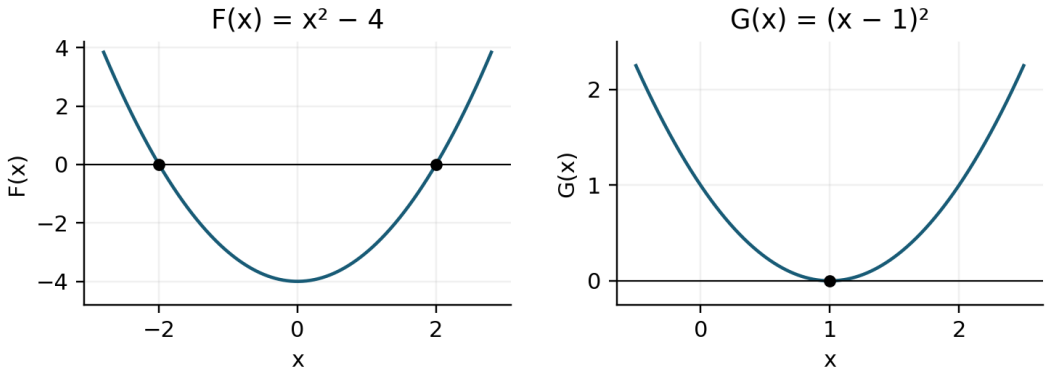


Figure 1 1 Two distinct roots and one repeated root

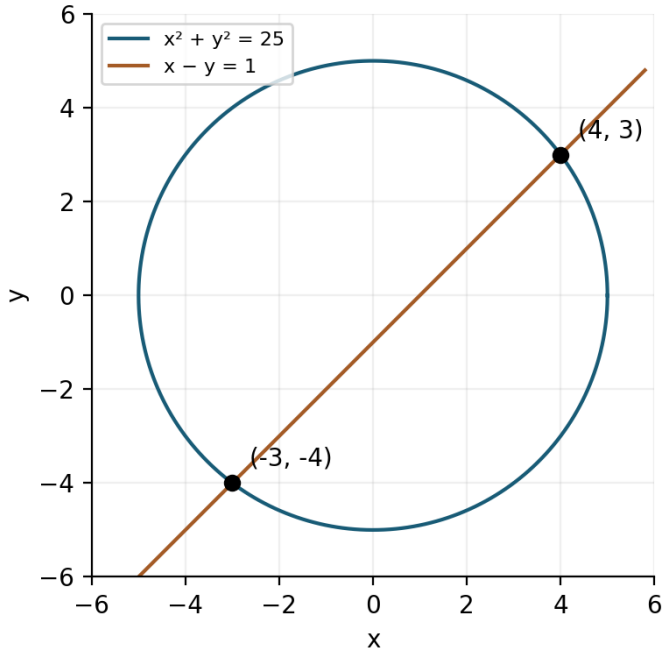


Figure 1 2 The common solutions of a circle and a line

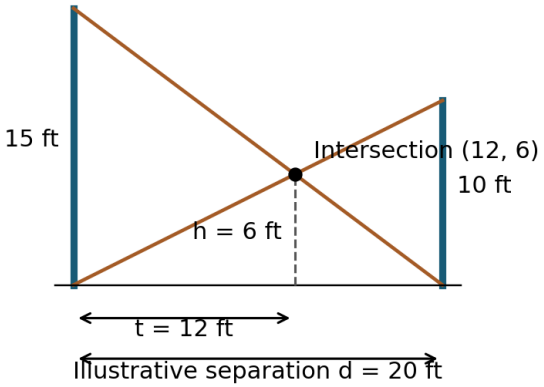


Figure 1 3 Crossed strings with an illustrative separation of 20 feet

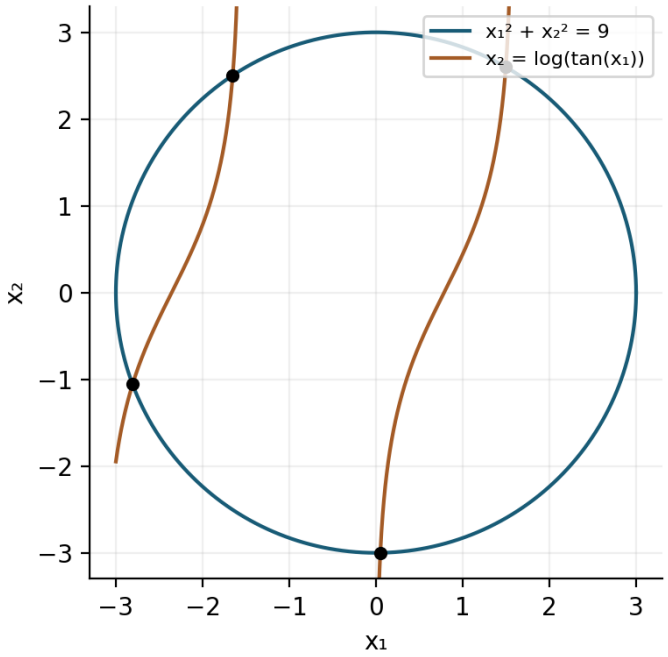


Figure 2 The circle and real logarithmic tangent branches with four reported intersections

Appendix B Version 3 worked model catalogue

The filenames below refer to the examples supplied with the complete Version 3.0.1 package. They are teaching and regression starting points. Run each separately and retain its status; the divergent diagnostic example is intentionally not a successful integral.

B 1 Boundary conditions

Example file	Mode	Use
V3_01_Heat_Neumann.txt	pde	Known reference or comparison model
V3_02_Heat_Robin.txt	pde	Known reference or comparison model
V3_03_Wave_Mixed.txt	pde	Known reference or comparison model
V3_04_Poisson_Mixed.txt	pde	Known reference or comparison model
V3_05_Neumann_Mean.txt	pde	Known reference or comparison model

B 2 Transport and integral accuracy

Example file	Mode	Use
V3_06_Reaction_Diffusion.txt	pde	Known reference or comparison model
V3_07_Advection_Diffusion.txt	pde	Known reference or comparison model
V3_08_Fredholm_Accuracy.txt	integral	Known reference or comparison model
V3_09_Volterra_Accuracy.txt	integral	Known reference or comparison model
V3_10_Coupled_Volterra.txt	integral	Known reference or comparison model

B 3 Eigenproblems and parameter studies

Example file	Mode	Use
V3_11_Sturm_Liouville.txt	eigen	Known reference or comparison model
V3_12_Eigen_Robin.txt	eigen	Known reference or comparison model
V3_13_Conductivity_Sweep.txt	pde	Known reference or comparison model
V3_14_Eigen_Parameter_Sweep.txt	eigen	Known reference or comparison model

B 4 Numerical quadrature

Example file	Mode	Use
V3_15_Quadrature_Endpoint.txt	quadrature	Known reference or comparison model
V3_16_Quadrature_Gaussian_Infinite.txt	quadrature	Known reference or comparison model
V3_17_Quadrature_Positive_Infinite.txt	quadrature	Known reference or comparison model
V3_18_Quadrature_Negative_Infinite.txt	quadrature	Known reference or comparison model
V3_19_Quadrature_Endpoint_Infinite.txt	quadrature	Known reference or comparison model
V3_20_Quadrature_Parameters.txt	quadrature	Known reference or comparison model
V3_21_Quadrature_Reversed.txt	quadrature	Known reference or comparison model
V3_22_Quadrature_Divergent_Diagnostic.txt	quadrature	Expected diagnostic
V3_23_Quadrature_Reciprocal_Power.txt	quadrature	Known reference or comparison model

B 5 Important scope distinctions

Finite or infinite integration of a known function is quadrature. An equation for an unknown function under an integral is integral. A supported separated spectral problem is eigen. Sweeps repeat supported numerical models; they are not a general objective optimiser. Version 3.0.1 also provides standalone tabular interpolation and differentiation with documented uncertainty options. General regression, automatic unit management and thermodynamic fluid databases remain outside the workflows described here.

Retain the original endpoint regression x/sqrt(1-sqrt(x)) on [0,1], its rationalised reformulation and the exact reference 64/35. Earlier platform-specific failure reports are historical observations. Recheck the current installed executable and retain its acceptance status and arithmetic metadata; a successful native source check is not a Windows acceptance test.

Appendix C Source basis and mathematical glossary

Source basis: the attached expanded 2.0.1 book, cumulative Version 3.0.1 source and examples, the updated User Guide, VERSION3_GUIDE.md, QUADRATURE_GUIDE.md and the engine 3.0.1.1 overflow-correction notes. Earlier exact-integer, ODE and BVP support documents retain their applicable scope. Later implementation changes take precedence over historical limitations. Earlier Windows endpoint-cancellation observations are retained as historical regression context; the updated native source and the installed Windows executable must be assessed separately. The Version 3.0.1 modification summary supplies the new syntax and Desktop features. The new worked values are mathematical references.

Residual. Difference between an entered relationship and its required equality or feasibility criterion.

Conditioning. Sensitivity of a solution to changes in data, distinct from arithmetic precision.

Exact integer verification. An original integer identity checked without floating-point rounding.

Enumeration completeness. Evidence that every point in the applicable stated discrete domain has been accounted for.

Trajectory. Samples from one numerical function or system solution over an interval.

Boundary residual. A numerical mismatch in an endpoint or boundary condition.

Defect. Difference between an interpolant derivative and the differential equation vector field.

Discrete residual. Mismatch in the finite-difference or quadrature equations, not a direct continuum-error bound.

Refinement. A change of grid/step/tolerances used to investigate numerical convergence.

Kernel. The function specifying how an integral operator weights other coordinates.

Singular continuation. A branch-compatible extension through a singular representation, available only for the supported class.

Run snapshot. The model/settings/data associated with a returned Desktop calculation, distinct from later editor changes.

Mathematical references retained from the supplied book

1. OpenStax. Chemistry 2e, Relative Strengths of Acids and Bases. Acid dissociation, the acetic-acid constant and the relationship between Ka and Kw.

https://openstax.org/books/chemistry-2e/pages/14-3-relative-strengths-of-acids-and-bases

2. University of Florida. The Lorenz System, PHZ4710 Introduction to Biological Physics. Background on the Lorenz differential equations. The book explicitly uses rho=26.

https://cmp.phys.ufl.edu/PHZ4710/files/unit5/Lorenz-system.html

3. Massachusetts Institute of Technology OpenCourseWare. Microelectronic Devices and Circuits, Fall 2009, Lecture 6. Ideal exponential diode behavior. The circuit parameters in this book are illustrative.

https://ocw.mit.edu/courses/6-012-microelectronic-devices-and-circuits-fall-2009/a342ce0afb84bfa8e5b7dc819ff3e9bc_MIT6_012F09_lec06.pdf

4. PARI/GP. GP Defaults and Transcendental Functions documentation. Working precision and independent numerical evaluation.

https://pari.math.u-bordeaux.fr/dochtml/ref/GP_defaults.html

https://pari.math.u-bordeaux.fr/doc.html

5. Python Software Foundation. subprocess documentation. Process launch, output capture and return codes.

https://docs.python.org/3/library/subprocess.html

6. Higham, Nicholas J. Accuracy and Stability of Numerical Algorithms. Second edition. Society for Industrial and Applied Mathematics, 2002. Further reading on rounding, stability and conditioning.

7. Developer-supplied problem statements and experimental records, October 2026. These include the fruit and triangle examples, the five-node recurrence, and the reported Patodia timing and residual observations. Timing records are not represented as independently audited performance measurements.

Historical attribution of the triangle problems is retained as attribution rather than a verified textual citation. Readers preparing historical scholarship should consult an identified edition or translation of the original work.

Current implementation and interface details should be checked against the matching public header and topic notes. Earlier chapter reference values and timings retain their stated origin. New worked models are checked separately in the revision record; a successful native calculation is not a Windows GUI test or a certified continuum error proof.

Appendix D Complete Compute Functions Reference

This appendix retains the Version 3.0.1 function catalogue and reference numbering. All mathematical functions, including every function available in Compute, are available to Version 4.0 workflows. Call shared core and scalar Compute functions directly in workflow expressions. Use a compute job for list-valued Compute output such as primefactors. Inside a service job, the service's own expression grammar applies. To use a Compute scalar in another service, evaluate it in the workflow and substitute the numeric result with @name.

This reference documents the standalone Compute Functions service as implemented in the Boost.Math / Boost.Multiprecision 113-bit backend. It covers execution syntax, argument counts, supported real domains, branch conventions, examples, Function Explorer, and practical accuracy limitations.

1 Getting started

Use compute or func on the first line. Each subsequent name = expression calculates a result; names assigned earlier in the script can be reused. There is no implicit multiplication: write 2*x, not 2x. Function names are case-insensitive, but variable names and special directives should be written as shown.

```
compute
precision = binary128
a = gamma(5)
b = erf(1)
c = besselj(0,2)
d = lambertw(5)
e = zeta(2)
total = a+b+c+d+e
```

Use expr = expression for one calculation stored as result. Supported operators are +, −, *, / and ^; parentheses control grouping. Constants #pi and #e (also pi and e) use high-precision values. Decimal literals are parsed directly at 113-bit precision; scientific notation such as 1e-25 is accepted. Maximum: 1,000 named calculations. A duplicate result name, an unknown name, invalid argument count, or nonfinite real result is an error. The service evaluates assignments sequentially; forward references are not resolved.

2 Precision and reliability

The default Compute evaluator uses Boost.Multiprecision cpp_bin_float_quad with 113 binary significand bits, approximately 34 significant decimal digits. precision=binary128 selects this backend explicitly; precision=auto uses the same default. precision=double selects the legacy evaluator. The 113-bit response identifies binary128_class, precision_bits=113, precision_digits=34 and certified=false.

Working precision does not certify 34 accurate digits. Condition numbers, proximity to poles, large arguments, cancellation, finite input accuracy, and algorithmic truncation can reduce attainable accuracy. In particular, numerical Si/Ci routines use series, quadrature, and asymptotic methods. Verify sensitive research results independently.

3 Function Explorer and data export

```
compute
function = besselj(0,x)
range = [0,20]
points = 500
```

The explorer evaluates the expression at equally spaced x values, including both endpoints; 2 ≤ points ≤ 20,000 (default 500). range must have two increasing real endpoints. The desktop renders sampled rows and a plot. Export the sampled table through the desktop CSV export action. If a point is outside the real domain, its y is null with status undefined; an all-invalid sample is an error. The independent variable is x.

4 Complete function catalogue

Notation: arguments separated by commas are required unless a row explicitly states alternatives. All functions return real, finite values only. The stated domains are mathematical regular domains or intended API restrictions; Boost may reject additional exceptional parameters.

Elementary and trigonometric functions

Function	Args	Meaning and domain	Example
sin(x)	1	Sine, radians Domain: All real x	sin(#pi/6)
cos(x)	1	Cosine, radians Domain: All real x	cos(#pi)
tan(x)	1	Tangent Domain: cos(x) ≠ 0	tan(#pi/4)
cot(x)	1	Cotangent Domain: sin(x) ≠ 0	cot(#pi/4)
sec(x)	1	Secant Domain: cos(x) ≠ 0	sec(0)
csc(x)	1	Cosecant Domain: sin(x) ≠ 0	csc(#pi/2)
asin(x)	1	Principal inverse sine Domain: −1 ≤ x ≤ 1	asin(0.5)
acos(x)	1	Principal inverse cosine Domain: −1 ≤ x ≤ 1	acos(0)
atan(x)	1	Principal inverse tangent Domain: All real x	atan(1)
acot(x)	1	Principal inverse cotangent, π/2 − atan(x) Domain: All real x	acot(0)
asec(x)	1	acos(1/x) Domain: |x| ≥ 1	asec(2)
acsc(x)	1	asin(1/x) Domain: |x| ≥ 1	acsc(2)
atan2(y,x)	2	Quadrant-aware angle, with y first. Domain: (x,y) is not (0,0). Shared Core rejects the origin; a backend-specific value there does not define a mathematical angle.	atan2(1,1)
sinh(x)	1	Hyperbolic sine Domain: All real x	sinh(1)
cosh(x)	1	Hyperbolic cosine Domain: All real x	cosh(1)
tanh(x)	1	Hyperbolic tangent Domain: All real x	tanh(1)
coth(x)	1	Hyperbolic cotangent Domain: x ≠ 0	coth(1)
sech(x)	1	Hyperbolic secant Domain: All real x	sech(0)
csch(x)	1	Hyperbolic cosecant Domain: x ≠ 0	csch(1)
asinh(x)	1	Inverse hyperbolic sine Domain: All real x	asinh(2)
acosh(x)	1	Inverse hyperbolic cosine Domain: x ≥ 1	acosh(2)
atanh(x)	1	Inverse hyperbolic tangent Domain: −1 < x < 1	atanh(0.5)
acoth(x)	1	Inverse hyperbolic cotangent Domain: |x| > 1	acoth(2)

Function	Args	Meaning and domain	Example
asech(x)	1	Inverse hyperbolic secant Domain: $0 < x \leq 1$	asech(0.5)
acsch(x)	1	Inverse hyperbolic cosecant Domain: $x \neq 0$	acsch(2)

Exponential, logarithmic and algebraic functions

Function	Args	Meaning and domain	Example
exp(x)	1	Exponential e^x Domain: All real x; finite output only	exp(1)
expm1(x)	1	Numerically stable exp(x) − 1 Domain: All real x; finite output only	expm1(1e-20)
log(x), ln(x)	1	Natural logarithm Domain: $x > 0$	log(#e)
log10(x)	1	Base-10 logarithm Domain: $x > 0$	log10(100)
log1p(x)	1	Numerically stable log(1+x) Domain: $x > -1$	log1p(1e-20)
logbase(x,b), logb(x,b)	2	Logarithm in base b Domain: $x > 0$, $b > 0$, $b \neq 1$	logbase(8,2)
sqrt(x)	1	Principal square root Domain: $x \geq 0$	sqrt(2)
cbrt(x)	1	Real cube root Domain: All real x	cbrt(-8)
pow(x,y)	2	Real power x^y. Domain: x>0 for arbitrary real y; negative x requires an integer exponent in this real evaluator. Use cbrt or a supported Compute root for real roots.	pow(2,10)
root(x,n)	2	Real n-th root Domain: $n \neq 0$; negative x requires odd integer n	root(-27,3)
abs(x)	1	Absolute value Domain: All real x	abs(-3.2)
hypot(x,y)	2	Euclidean norm $\sqrt{(x^2+y^2)}$ Domain: All real x,y; finite output only	hypot(3,4)
min(a,b)	2	Minimum of two values Domain: All real a,b	min(3,5)
max(a,b)	2	Maximum of two values Domain: All real a,b	max(3,5)

Discrete and rounding utilities

Function	Args	Meaning and domain	Example
sign(x)	1	Sign: −1, 0 or +1 Domain: All real x	sign(-4)
floor(x)	1	Greatest integer ≤ x Domain: All real x	floor(-1.2)
ceil(x)	1	Least integer ≥ x Domain: All real x	ceil(-1.2)

Function	Args	Meaning and domain	Example
round(x)	1	Nearest integer; half values away from zero Domain: All real x	round(-2.5)
trunc(x)	1	Truncate fractional part toward zero Domain: All real x	trunc(-3.8)
frac(x)	1	Fractional part x − floor(x) Domain: All real x; result in [0,1)	frac(-1.25)
factorial(n), fact(n)	1	Product 1×2×…×n (0! = 1) Domain: Integer 0 ≤ n ≤ 2000	factorial(6)

Gamma, beta and error functions

Function	Args	Meaning and domain	Example
gamma(x)	1	Euler gamma Γ(x) Domain: Real x except nonpositive integers; finite result	gamma(0.5)
lgamma(x)	1	Logarithm of |Γ(x)| under Boost.Math convention Domain: Real x except nonpositive integers	lgamma(5)
beta(a,b)	2	Euler beta B(a,b) Domain: Common regular case a>0,b>0; other real values depend on gamma poles and Boost implementation	beta(2,3)
erf(x)	1	Gaussian error function Domain: All real x	erf(1)
erfc(x)	1	Complementary error function 1−erf(x) Domain: All real x	erfc(1)
digamma(x)	1	Logarithmic derivative Γ′(x)/Γ(x) Domain: Real x except nonpositive integers	digamma(1)
polygamma(n,x)	2	n-th derivative of digamma Domain: Integer 0 ≤ n ≤ 8; x>0 recommended (required for n>0 per documented service contract)	polygamma(1,1)

Bessel, Airy and orthogonal polynomials

Function	Args	Meaning and domain	Example
besselj(v,x)	2	Bessel function J of order v Domain: x≥0 for noninteger v; real-valued result required	besselj(0,2)
bessely(v,x)	2	Bessel function Y of order v Domain: x>0 (real branch)	bessely(0,2)
besseli(v,x)	2	Modified Bessel I of order v Domain: x≥0 for general real v; real result required	besseli(0,2)
besselk(v,x)	2	Modified Bessel K of order v Domain: x>0	besselk(0,2)

Function	Args	Meaning and domain	Example
airy_ai(x)	1	Airy Ai solution Domain: All real x	airy_ai(0)
airy_bi(x)	1	Airy Bi solution Domain: All real x	airy_bi(0)
legendre(n,x)	2	Legendre polynomial $P_n(x)$ Domain: Integer n≥0 (≤10000); real x	legendre(3,0.5)
assoc_legendre(n,m,x)	3	Associated Legendre polynomial Domain: Integers n,m≥0; m≤n; typically |x|≤1	assoc_legendre(3,1,0.5)
hermite(n,x)	2	Physicists' Hermite polynomial Domain: Integer n≥0 (≤10000); real x	hermite(3,2)
laguerre(n,x)	2	Laguerre polynomial Domain: Integer n≥0 (≤10000); real x	laguerre(3,2)
assoc_laguerre(n,α,x)	3	Generalized Laguerre polynomial (implemented integer α) Domain: Integers n,α≥0 (≤10000); real x	assoc_laguerre(3,2,1)
chebyshev_t(n,x)	2	Chebyshev polynomial $T_n(x)$ Domain: Integer n≥0 (≤10000); real x	chebyshev_t(4,0.5)
chebyshev_u(n,x)	2	Chebyshev polynomial $U_n(x)$ Domain: Integer n≥0 (≤10000); real x	chebyshev_u(4,0.5)

Advanced special functions

Function	Args	Meaning and domain	Example
zeta(s)	1	Riemann zeta ζ(s) Domain: Real s ≠ 1; finite real result	zeta(2)
lambertw(x)	1	Principal real branch $W_0(x)$, W e^W = x Domain: x ≥ −1/e	lambertw(5)
lambertw(x,branch)	2	Real branch W_0 or W_{-1} Domain: branch integer 0 or −1; W_{-1} on −1/e ≤ x < 0	lambertw(-0.1,-1)
ei(x)	1	Exponential integral Ei(x) Domain: Real x ≠ 0 (principal real value)	ei(1)
si(x)	1	Sine integral $\int_0^x \sin(t)/t\,dt$ Domain: All real x; series/quadrature/asymptotic approximations	si(1)
ci(x)	1	Cosine integral $\gamma+\ln(x)+\int_0^x(\cos(t)-1)/t\,dt$ Domain: x > 0	ci(1)

Elliptic integrals

Function	Args	Meaning and domain	Example
elliptic_k(m)	1	Complete elliptic integral K with parameter $m=k^2$ Domain: 0≤m<1 (real finite principal branch)	elliptic_k(0.5)

Function	Args	Meaning and domain	Example
elliptic_e(m)	1	Complete elliptic integral E with parameter $m=k^2$ Domain: $0\le m\le 1$	elliptic_e(0.5)
elliptic_f(phi,m)	2	Incomplete elliptic integral of first kind Domain: $0\le m\le 1$; avoid singular integration paths	elliptic_f(1,0.5)
elliptic_einc(phi,m)	2	Incomplete elliptic integral of second kind Domain: $0\le m\le 1$ for real modulus	elliptic_einc(1,0.5)
elliptic_pi(n,m)	2	Complete elliptic integral of third kind Domain: $0\le m<1$; $n<1$ for regular real integral	elliptic_pi(0.2,0.5)
elliptic_pi(phi,n,m)	3	Incomplete elliptic integral of third kind Domain: $0\le m\le 1$; $1-n\sin^2(t)$ must not vanish on integration path	elliptic_pi(1,0.2,0.5)

6 Domain, branch and sign conventions

Angles are in radians. atan2(y,x) takes y first, then x. acot(x) is evaluated as $\pi/2 - \text{atan}(x)$, with range $(0,\pi)$.

Lambert W: omitted branch means W_0. The optional branch argument is exactly 0 or −1. W_{-1} is real only from $-1/e$ to 0 (excluding 0 for finite output). Elliptic functions use parameter $m = k^2$, not modulus k. The implementation obtains k using sqrt(m); thus negative m is not supported by this interface even where alternate analytic continuations exist. Associated Laguerre α is restricted to a nonnegative integer by this implementation; this is narrower than the general mathematical parameter. Associated Legendre sign conventions vary across references; check Condon–Shortley phases before comparing coefficients. The Bessel order v can be real. Domains with negative arguments can be branch-dependent; use positive x for Y and K and nonnegative x for general noninteger orders. For negative bases, pow uses real multiprecision power semantics; root(x,n) explicitly handles odd integer roots of negative x. Polygamma derivative order n is an integer 0–8. Factorials are accepted as integers 0–2000, but large values require appropriate handling of output size. Domain failures may be raised by Boost.Math, the parser, or the nonfinite-result guard. Exceptional endpoints are not automatically interpreted as limits. use an integer exponent with pow or ^ in the real power evaluator. For a real cube root of a negative argument use Shared Core cbrt, or a supported odd-integer root in Compute. A fractional floating-point exponent such as 1/3 is not a reliable substitute for cbrt on negative inputs.

7 Teaching and research examples

A Basic special functions

```
compute
g = gamma(0.5)
e = erf(1)
z = zeta(2)
identity = 6*z/(#pi^2)
```

The identity result should be approximately 1; it verifies zeta(2)=pi^2/6. Independently compare g with sqrt(#pi) to check gamma(0.5).

B Lambert W residual

```
compute
w = lambertw(5)
residual = w*exp(w)-5
```

Residual should be near numerical zero; this is a useful high-precision self-consistency test.

C Stable small differences

```
compute
a = expm1(1e-20)
b = log1p(1e-20)
ratio = a/b
```

Dedicated expm1/log1p avoid avoidable cancellation for tiny inputs.

D Orthogonal polynomials

```
compute
p = legendre(4,0.3)
h = hermite(4,0.3)
l = laguerre(4,0.3)
t = chebyshev_t(4,0.3)
```

Useful in approximation theory, quadrature, spectral methods, and numerical physics.

E Bessel sampling

```
compute
function = besselj(0,x)
range = [0,20]
points = 500
```

Plot oscillation and locate near-zeros visually; a sample plot is not a certified root-finding procedure.

F Real-domain boundary handling

```
compute
function = log(x)
range = [-1,3]
points = 41
```

The explorer reports undefined for $x \le 0$ and valid values for $x > 0$.

G Elliptic parameter convention

```
compute
k = elliptic_k(0.5)
e = elliptic_e(0.5)
f = elliptic_f(1,0.5)
```

For elliptic_k(m) and elliptic_e(m), the first argument is m. For elliptic_f(phi,m), the second argument is m. In each case m=k^2, not the modulus k.

H Hyperbolic and inverse functions

```
compute
a = acoth(2)
b = asinh(2)
c = coth(2)
d = sech(1)
```

Demonstrates restricted inverse domains and unbounded hyperbolic functions.

8 Troubleshooting and compatibility

Unknown function or variable. Check the function spelling, its Shared Core or Compute-only classification, and whether referenced names were assigned earlier.

Wrong number of arguments. Check the Args column. lambertw and elliptic_pi have more than one supported signature.

Nonfinite result or domain error. Check the real domain, poles, branch restrictions and endpoint behaviour.

Precision setting. Use precision=binary128, precision=auto or precision=double. binary128_class is response metadata, not an accepted precision directive.

No valid Explorer samples. Use finite increasing range endpoints and enough points, and check that at least one sample has a finite real value.

Build consistency. Use a matching rebuilt Desktop and engine. Copying new source or Boost headers beside an old executable does not update its numerical implementation.

9 Reproducibility checklist

- Record solver version and selected precision.
- Record the exact input script, argument order and parameter conventions.
- Check real domains and compare sensitive values with an independent reference.
- Cross-check functional identities or recurrence relations where possible.
- For sampled functions, record the interval and point count; distinguish the sampled interpolation from the underlying mathematical function.

10 Function availability across solver services

Compute Functions is deliberately isolated. The other solver services use the main mathematical expression parser (or a related numerical evaluator), not the new Boost.Math function dispatcher. Consequently, a function working in compute is not evidence that it will parse in an equation, integral, ODE, PDE or limit script. The classifications below describe function-name availability, not equal precision or identical algorithm behavior.

Function family and members	Classification	What this means
Elementary trig and inverse sin, cos, tan, asin, acos, atan	Shared core	Main parser: yes; other services depend on their own expression route
Extended trig and inverse cot, sec, csc, acot, asec, acsc	Shared core	Main parser: yes; availability within each service depends on that service’s expression parser
Hyperbolic and inverse sinh, cosh, tanh, asinh, acosh, atanh, coth, sech, csch, acoth, asech, acsch	Shared core	Main parser: yes
Core exponential and algebra exp, log, ln, log10, sqrt, abs, cbrt, log1p, expm1, pow, atan2, min, max, hypot	Shared core	Main parser: yes; function signatures are shared
Utilities and rounding sign, floor, ceil, round, trunc, frac, factorial, fact, root, logbase, logb	Compute only	Not listed in main parser’s function-name set
Gamma beta error gamma, lgamma, beta, erf, erfc, digamma, polygamma	Compute only	Boost.Math backend does not automatically extend other parsers
Bessel and Airy besselj, bessely, besseli, besselk, airy_ai, airy_bi	Compute only	Not listed in main parser
Orthogonal polynomials legendre, assoc_legendre, hermite, laguerre, assoc_laguerre, chebyshev_t, chebyshev_u	Compute only	Not listed in main parser
Advanced special zeta, lambertw, ei, si, ci	Compute only	Not listed in main parser
Elliptic integrals elliptic_k, elliptic_e, elliptic_f, elliptic_einc, elliptic_pi	Compute only	Not listed in main parser

Function family and members	Classification	What this means
Exact number theory totient, gcd, lcm, isprime, primepi, mobius, divisorcount, divisorsum, primefactors, primefactor	Compute only	Exact-integer helpers exposed through standalone compute/func. Not callable inside the main equation, ODE, quadrature, PDE or limit expression grammars.

Algebraic equations and systems use Shared Core functions in their supported expression fields. Compute-only functions do not extend this parser.

Differentiation and limits also accept Shared Core expression names. Domain, smoothness, derivative availability and series/limit requirements remain separate constraints.

ODE right-hand sides, events and regular BVP expressions use Shared Core functions within their respective model grammars. Their structural requirements still apply.

Quadrature, supported PDEs and integral equations accept Shared Core functions in their mathematical expression fields. Their numerical evaluators, pole guards and working precision can differ from Compute.

Interpolation and tabular differentiation process supplied observations and supported numerical controls. They do not expose the Compute special-function dispatcher. Use compatible exported data when a tabular workflow is appropriate.

Shared Core identifies common function names. Precision, domain, conditioning and supported submethods still depend on the calling service. In Version 4.0 workflows, scalar Compute functions are directly callable.

11 Exact number theory function reference

These exact-integer utilities are available to workflows. Use scalar calls in workflow expressions and a compute job for formatted list output.

11 1 Complete function catalogue

Function	Args	Meaning and accepted domain	Example / result
totient(n)	1	Euler's totient ϕ(n): number of integers from 1 to n relatively prime to n. $1 \le n \le 10^{12}$.	totient(5929) = 4620
gcd(a,b)	2	Greatest common divisor of nonnegative integers. $0 \le a,b \le 10^{12}$; gcd(0,0)=0.	gcd(42,77) = 7
lcm(a,b)	2	Least common multiple of nonnegative integers. $0 \le a,b \le 10^{12}$; lcm(0,b)=0.	lcm(14,22) = 154
isprime(n)	1	Primality predicate: numeric 1 if n is prime, otherwise 0. $0 \le n \le 10^{12}$.	isprime(97) = 1
primepi(n)	1	Exact number of primes p ≤ n. 0 ≤ n ≤ 100,000,000 (inclusive).	primepi(1000000) = 78498
mobius(n)	1	Möbius μ(n): 0 for repeated prime factors, otherwise (−1) to the count of distinct factors. $1 \le n \le 10^{12}$.	mobius(30) = -1
divisorcount(n)	1	Number τ(n) of positive divisors of n. $1 \le n \le 10^{12}$.	divisorcount(360) = 24
divisorsum(n)	1	Sum σ(n) of positive divisors of n. $1 \le n \le 10^{12}$.	divisorsum(28) = 56
primefactors(n)	1	Sorted list of prime factors with multiplicity. $1 \le n \le 10^{12}$; for n=1 returns an empty list.	primefactors(5929) = [7,7,11,11]
primefactor(n,k)	2	Select 1-based kth prime factor including multiplicity. $2 \le n \le 10^{12}$; 1 ≤ k ≤ 64 and k must exist.	primefactor(5929,3) = 11

11 2 Multi calculation example

```
compute
precision = binary128
n = 5929
a = totient(n)
b = gcd(n,77)
c = lcm(14,22)
d = isprime(n)
e = primepi(1000000)
f = mobius(30)
g = divisorcount(360)
h = divisorsum(28)
i = primefactors(n)
j = primefactor(n,3)
```

References: a=4620; b=77; c=154; d=0; e=78498; f=−1; g=24; h=56; i=[7,7,11,11]; j=11. The value of i is a structured list, not a scalar that can be used in a later arithmetic assignment. Choose primefactor(n,k) when a single factor is required.

11 3 Maximum n for primepi

The maximum supported argument in Version 3.0.1 is exactly 100,000,000 inclusive. primepi(100000000) = 5,761,455. primepi(100000001) must return a range error rather than attempting a lengthy count or silently approximating. The cap is a deliberate predictable-runtime rule in the current segmented-sieve implementation; it

is not a mathematical restriction on prime counting. The other exact-integer number-theory routines accept inputs up to 10^{12}, subject to individual lower bounds. Factorisation by exact trial division can still take noticeably longer for a large prime near the ceiling. Reject unsupported inputs before expensive work. In optional double mode, exact integer constraints still apply and the lcm implementation refuses results beyond the exactly representable double integer range.

11 4 Precision and structured outputs

By default, Compute uses binary128-class Boost.Multiprecision arithmetic (113 binary significant bits, approximately 34 decimal digits). Integer factorisation and divisor calculations use exact integer operations within the stated input ceilings; 113-bit working precision does not imply that every unrelated special-function result has 34 correct decimal digits. The JSON output represents factor-list members and exact integer results as decimal strings where needed to avoid loss in clients that parse JSON numbers as double.

isprime returns numeric 0 or 1 and can participate in arithmetic. primefactors returns a list of integer strings in JSON, including [] for primefactors(1); it cannot be embedded in scalar arithmetic or passed directly to a general equation expression.

11 5 Function scope and regression checks

All ten number-theory functions are available to workflows. Standalone non-Compute jobs use their own expression grammar.

```
python tests/run_compute_number_theory_regression.py engine/bin/equation_solver.exe
```

Run the number-theory regression with the matching rebuilt Windows CLI and repeat relevant Desktop report and CSV checks. Verify prime-counting landmarks, factor multiplicities, n=1, zero arguments, out-of-range and nonintegral inputs, missing indexed factors and double/binary128 routing.

The final installed MSIX must be tested independently. This document does not claim that its current payload has been rebuilt and validated.