**PATODIA INFOTECH PRIVATE LIMITED**

# Patodia Equation Solver User Guide

*Version 4.0 Windows x64*

Version 4.0 guide update | 11 November 2026

A practical guide to writing models, using Desktop and interpreting exact and numerical results. Sections 89–92 show how to automate studies with workflow scripts in Version 4.0.

Version 3.0.1 also includes standalone differentiation and interpolation (including tabular data), numerical limits, standalone Compute Functions, high-precision special functions, exact linear Diophantine families and number-theory utilities. The new sections at the end of this guide explain their inputs, scope and limitations.

Exact verification, estimated numerical accuracy and completeness are separate claims. The reported Windows endpoint-cancellation case remains open.

Patodia Infotech Private Limited, Bengaluru, India
panna@patodiainfo.com
https://patodiainfo.com/EquationSolver/

# Contents

Select a section below or use Word Navigation Pane headings.

01 Choose the right solver

02 Install and update the Windows package

03 Trial licensing and precision entitlement

04 The Desktop workspace and readable text

05 First solve and reusable projects

06 Write equations and declare domains

07 Functions constants and number styles

08 Bounds conditions and search settings

09 Understand precision accuracy and completeness

10 Results exports printing and stopping

11 Algebraic roots mixed integers and examples

12 Integer remainders and modular models

13 Integer products and affine factor shortcuts

14 Exact integer input and returned strings

15 Arbitrary magnitude exact integer bounds

16 Long values copying and the viewer limitation

17 Standard positive Pell equations

18 Generalised Pell equations with integer N

19 Negative zero and square Pell parameters

20 Pell completeness limits and practical choices

21 General quadratic equations in two integers

22 Signed quadratics and bounded parabolas

23 Bounds and limits for general quadratics

24 Positive weighted sums of squares

25 Signed zero and mixed square coefficients

26 Weighted square completion and exact checks

# Contents continued

Select a section below or use Word Navigation Pane headings.

27 Initial value ODEs

28 Higher order and coupled ODEs

29 Choose RK4 RK45 or SDIRK2

30 Stiff ODEs events and continuous output

31 Automatic continuation at supported singular points

32 Boundary value ODEs

33 Worked BVP with logarithmic boundary data

34 Worked BVP with Robin conditions

35 BVP guesses controls and convergence

36 PDE scope coordinates and fields

37 Poisson and Laplace on a rectangle

38 Laplace with boundary values from a diagram

39 Read the square Laplace results

40 Heat diffusion in one spatial dimension

41 Wave propagation and the CFL condition

42 PDE refinement residuals and grid limits

43 Fredholm integral equations of the second kind

44 Volterra integral equations of the second kind

45 Interpret integral equation accuracy

46 Command line use with the same models

47 DLL integration and structured results

48 Reference controls for ordinary ODEs

49 Reference controls for BVP PDE and integrals

50 Diagnose numerical and input failures

51 When numbers differ from a reference

52 Keyboard commands and support

# Contents continued

Select a section below or use Word Navigation Pane headings.

53 Desktop File Edit Search and View commands

54 Desktop Solver Settings and Compute commands

55 Desktop Help Examples and licence locations

56 Desktop plots exports and completed snapshots

57 Scope and validation of this edition

58 What is new in Version 3.0.1

59 PDE boundary conditions and normal signs

60 Pure Neumann fields and the mean value

61 Accuracy targets and automatic refinement

62 Convergence tables and optional exact references

63 Heat maps contours and time exploration

64 Regular Sturm-Liouville eigenproblems

65 Eigenvalue accuracy and mode normalisation

66 Linear reaction-diffusion in one dimension

67 Linear advection-diffusion in one dimension

68 Linear systems of Volterra equations

69 Parameter sweeps and comparison tables

70 Standalone numerical definite integration

71 Infinite limits constants and interval notation

72 Quadrature parameters and sensible defaults

73 How max extent and transformations work

74 Quadrature acceptance and failure diagnostics

75 Quadrature reference examples and tail bounds

76 Endpoint cancellation: an open test case

77 A reproducible testing and reporting workflow

78 Numerical differentiation of expressions

79 Differentiation of tables and CSV measurements

80 Interpolation and measured data

81 Numerical limits and one sided behaviour

82 Compute Functions and the Function Explorer

83 Compute function catalogue and isolation

84 Compute number theory and input ceilings

85 Exact linear Diophantine equations

86 Release readiness and regression checks

87 Split integrals and investigate singularities

88 Equation Generator and Fun with Mathematics

89 Run your first workflow in Desktop

90 Write conditions loops and functions

91 Sweep parameters and combine services

92 Set budgets and interpret workflow results

GETTING STARTED

# 01 Choose the right solver

Patodia Equation Solver 3.0.1 combines algebraic roots, exact integer methods, differential equations, linear integral equations, regular eigenproblems and standalone numerical definite integration in one Desktop workflow.

| Problem | Start the model with | Result |
|---|---|---|
| Algebraic roots / integer models | Equations and domain declarations | Root vectors / verified integer values |
| Initial-value / boundary-value ODE | ode / bvp | A sampled solution curve |
| Heat, wave, Laplace / Poisson | pde and type | A sampled field with diagnostics |
| Reaction / advection-diffusion | pde and type | One-dimensional time-dependent field |
| Second-kind integral equation | integral and type | Samples of the unknown function |
| Regular Sturm-Liouville problem | eigen | Eigenvalues and eigenfunctions |
| Known-function definite integral | quadrature | One approximate integral value |

### Three different meanings of success

Exact integer acceptance verifies an identity. Numerical root acceptance checks a residual. Differential, integral, eigen and quadrature results use their stated numerical checks. None of these alone establishes every possible solution or a certified continuous-error bound.

Initial guesses are not normally needed for algebraic root discovery. Nonlinear BVPs may need a starting curve and can have multiple branches. Numerical exact-reference fields are optional.

The standalone chapters retain the Version 3.0.1 baseline. Sections 89–92 cover the Version 4.0 workflow interface. Before claiming a distributed Windows build supports a feature, rebuild the CLI, DLL and desktop executables and validate them inside the final MSIX package. Examples provide reproducible syntax and interpretation, not certification of every numerical answer.

GETTING STARTED

# 02 Install and update the Windows package

Extract the complete ZIP into a new writable folder. Do not run from inside the archive. In the complete source-and-binaries package, launch dist\PatodiaEquationSolver.exe. No compiler or Python installation is needed for normal use.

| Keep together in dist | Purpose |
|---|---|
| PatodiaEquationSolver.exe | Desktop editor, results and plots |
| equation_solver.exe | Worker process launched by the Desktop |
| equation_solver.dll / equation_solver.h | C ABI library and integration header |
| libquadmath-0.dll | Binary128 arithmetic runtime |
| libgcc_s_seh-1.dll / libwinpthread-1.dll | Compiler and threading runtimes |
| docs / examples / licenses | Documentation, models and notices |

### Update without mixing components

Save projects and exports, close the application, and extract the corrected package into a fresh folder. Keep the previous folder as a backup until acceptance checks pass. Store personal models separately from bundled examples. If replacing an existing portable copy, replace the complete dist folder.

Baseline product version: 3.0.1. The overflow-corrected engine EXE and DLL have file/product version 3.0.1.1; the compatible unchanged Desktop has file version 3.0.1.0.

The supplied installer kit is not an installer EXE. Windows binaries were cross-compiled and inspected; Windows execution and GUI interaction require local acceptance testing. Run Run_Version3_Checks.ps1 from dist where PowerShell policy permits.

**USER GUIDE**

# 03 Trial licensing and precision entitlement

The existing automatic 14-day trial and signed licence workflow are retained. Trial state belongs to the current Windows account and is shared by the Desktop, CLI and DLL. Reinstallation or moving the application does not restart the trial.

| Action | Desktop location |
|---|---|
| Check edition scope and expiry | Help > Licence Information |
| Obtain the machine identifier | Help > Machine Fingerprint |
| Install the signed licence file | Help > Activate Licence |
| Confirm product version | Help > About |

### Individual and institutional use

Commercial and Professional individual activation normally uses a machine fingerprint and a signed file from the publisher. Departmental and campus licences use the institution’s signed file and scope; eligible users should follow their administrator’s instructions. OEM integration and redistribution require the agreed entitlement.

Commercial permits double-precision Desktop and command-line solving; it does not permit DLL solving or extended precision. Professional, Trial, Academic and OEM entitlements permit binary128 and DLL computation subject to their terms. Exact integer methods have their own arithmetic path; selecting extended precision does not turn a heuristic unsupported model into an exact integer enumeration.

Keep the system clock correct. An expired or invalid installed licence is reported rather than silently replaced by a new trial. Do not edit a signed licence file.

The numerical capabilities in this guide do not change the licence’s expiry, institutional scope, DLL permissions or redistribution terms. Check the displayed entitlement rather than assuming that access to the Desktop grants every deployment right.

GETTING STARTED

# 04 The Desktop workspace and readable text

| Area | Use |
|---|---|
| Equation editor | One model, syntax highlighting and line numbers |
| Solutions / Run report / Diagnostics | Values, captured model, method, status and reasons |
| ODE / BVP / integral plot | Curves, phase portraits, integral components and eigenfunctions |
| ODE events | Located event times, states and terminal flags |
| PDE fields / animation | Heat maps, contours, sections and computed time frames |
| Convergence / eigenpairs / sweeps | Refinement history, reference errors and mode data |
| Parameter sweep | Per-case summaries, selection and comparison plots |

### Choose fonts and sizes

Use Equation font and Size above the editor, and Solution font and Size above the results. Preferences are saved automatically. View > Reset equation and solution fonts restores defaults. A larger display font does not increase numerical precision.

### Read columns by name

Poisson tables use x,y,u; time PDEs use x,t,u. Integral equations return x and the unknown function y or numbered components. Quadrature returns one integral value, not a trajectory. Exact/error columns appear when a compatible reference is supplied.

Plots and animation display completed samples. Editing a model or moving a slider does not solve again. Numerical strings in tables and exports preserve more precision than display plots.

**USER GUIDE**

# 05 First solve and reusable projects

## Solve two equations

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

Press F4 to check syntax, then F5 or Ctrl+Enter to solve. The result is x=3 and y=2. A two-second budget is a reasonable starting point for this small model. Read the variable names and residual, then open Run report to see the model used.

## Find alternatives

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

The two mathematical solutions are (2,3) and (3,2). A heuristic run can discover them in a different order. If a result cap or time stop truncates a run, more solutions can remain.

## Save the model and the calculation separately

Save a .pdeq project to retain the equation text, solver settings, number style and bound overrides. Save .txt or .eq when you want model text alone. A project does not store the Solutions table. Export CSV/JSON and the run report separately when you need to preserve a calculation.

Use File > Open for a complete model. File > Import equations inserts text at the cursor; importing two complete models can accidentally combine incompatible declarations. Save personal copies before editing supplied examples.

**LANGUAGE AND CONTROLS**

# 06 Write equations and declare domains

Use one equation, declaration or condition per line. Names are case-sensitive ASCII identifiers. Write explicit multiplication: xy is one name, whereas x*y is a product. Newlines are clearer than multiple top-level semicolon statements.

| Syntax | Meaning |
|---|---|
| x^2+y^2=25 | An equation; ** is also accepted for powers |
| int x / integer x | Signed integers including zero |
| pos int x / nonneg int x | Strictly positive / nonnegative integers |
| neg int x / nonpos int x | Strictly negative / nonpositive integers |
| binary selected | Only 0 and 1 |
| bound x=[a,b] | Inclusive algebraic bounds |
| // text or /* text */ | Line or block comment |

## Choose the intended model class

Integer-looking equations with real variables describe continuous sets. Declare integer domains when required. Bounds intersect domain restrictions; a positive declaration does not supply a finite upper bound.

Algebraic models use bound. IVPs and BVPs use interval. PDE spatial coordinates use domain, with interval t for time. Integral equations and eigenproblems use interval x. Standalone quadrature instead uses variable=t and interval=[a,b]. Do not combine different mode markers.

There is no general minimise/maximise objective, complex-root enumeration or symbolic antiderivative mode. Parameter sweeps are repeated numerical solves, not optimisation.

**USER GUIDE**

# 07 Functions constants and number styles

| Group | Examples |
|---|---|
| Trigonometric and hyperbolic | sin cos tan asin acos atan atan2 sinh cosh tanh |
| Exponential and logarithmic | exp expm1 log ln log10 log1p |
| Roots and magnitude | sqrt cbrt abs hypot |
| Two argument and rounding | min max pow floor ceil round |
| Mathematical constants | Use pi in differential models; #pi and #e are explicit constants in the general language |

Functions need parentheses and trigonometric arguments use radians. Both log and ln mean natural logarithm. Respect real domains: ln(x) needs x>0, sqrt(x) needs x>=0 and denominators must not vanish. Syntax checking does not establish that every evaluation along a solution is defined.

### Fixed constants are not unknowns

constant rate=0.7 fixes rate. In contrast, rate=0.7 is an equation in a variable in an algebraic model. Definitions may depend on other fixed constants but cannot be cyclic or depend on an unknown state. For specialised exact integer recognition, prefer explicit integer coefficient literals; evaluated named constants may not retain exact literal provenance.

| Element | US style | EU style |
|---|---|---|
| Decimal | 1.5 | 1,5 |
| Two arguments | max(1.5,2.5) | max(1,5;2,5) |
| Interval | [0,1] | [0;1] |
| CSV | Comma separators | Semicolon separators |
| JSON numeric strings | Decimal points | Decimal points |

Select Compute > Number Style before entering/importing text. Changing style converts the editor; existing results retain their run style. Split int x,y into separate declarations before converting to EU. Do not use thousands separators.

**USER GUIDE**

# 08 Bounds conditions and search settings

## Constrain an algebraic model

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

Only x=1 remains. Use separate <= or >= conditions; strict comparisons, !=, logical combinations and chained expressions such as 0<=x<=3 are not supported. General numerical conditions use residual tolerances. A hard variable bound is a different control.

Compute > Variables and Bounds displays algebraic variables. An override takes precedence over a bound in model text. Clear both cells to restore the model/default interval. A project retains overrides; a plain text model does not. For ODE/BVP/PDE/integral input, put domains and controls directly in the model.

| Desktop setting | Default | Applies mainly to |
|---|---|---|
| Time budget | 2 seconds | All numerical work; 0 removes time limit |
| Maximum solutions | 100 | Algebraic and integer result lists |
| Maximum starts / iterations | 30000 / 300 | Heuristic root searches |
| Residual tolerance | 1e-9 | Ordinary double root acceptance |
| Seed | 42 | Randomised algebraic discovery |
| Radius / growth / stages | 10 / 10 / 4 | Automatic algebraic search ranges |

A result cap is not a proof of completeness. Root-search starts, seed and radius do not control a PDE mesh. Differential and integral accuracy controls belong in the model.

**LANGUAGE AND CONTROLS**

# 09 Understand precision accuracy and completeness

| Arithmetic path | Meaning | How to assess it |
|---|---|---|
| Double | About 15-16 significant decimal digits | Residual, conditioning and convergence |
| Binary128 | About 34 significant decimal digits | Error estimates and roundoff sensitivity |
| Exact integer | Arbitrary-precision recognised integer paths | Exact verification and enumeration flags |

Ordinary algebraic extended solving can use double discovery followed by binary128 refinement. ODE, BVP, PDE and integral-equation methods use the selected arithmetic. Standalone quadrature uses internal binary128 guard arithmetic even in standard mode; its precision field reports binary128_guard or binary128.

### Read separate claims separately

A small residual is not a bound on solution error. A completed PDE solve does not prove the mesh resolves the continuum problem. An exact integer identity does not prove every permitted pair was enumerated. An accepted quadrature estimate is not a certified global bound.

Version 3 can refine supported PDEs, integral equations and eigenvalues automatically. Read accuracy_target_met as well as field_successful and status. refine=false leaves the target unassessed. Exact reference expressions are optional checks, not requirements for solving.

Printing 35 digits does not establish 35 accurate digits. More arithmetic precision does not remove discretisation error, poor conditioning, uncertain measurements or unresolved features. Hundreds of accurate digits are outside binary128 arithmetic.

For exact integer routes inspect integer_search_complete and applicable positive-domain completion fields. For numerical methods global_error_certified remains false.

**DESKTOP WORKFLOW**

# 10 Results exports printing and stopping

Identify the problem type before interpreting rows. Algebraic results contain root vectors; ODE/BVP and fields contain coordinate samples; eigen results include modes; quadrature provides a scalar integral. The captured Run report records the model actually used.

| Task | Action |
|---|---|
| Retain all numerical data | Export results / differential trajectory as JSON or CSV |
| Retain the explanation | Export run report |
| Copy a rectangular selection | Ctrl+Shift+C |
| Print | Choose equations or the completed report |
| Save events | Export ODE events CSV |
| Save Version 3 plots / tables | Use the relevant panel PNG / CSV export controls |

Preserve exact integers and extended values as decimal strings in downstream code. Import long CSV numbers as Text in a spreadsheet. Plots use display precision; they do not retain all table digits.

### Stopping and partial results

Desktop Stop terminates the worker. It cannot retrieve unfinished JSON, and a previous completed snapshot can remain visible. A cooperative engine time limit can return a mode-specific status. An accuracy_not_met field result may retain computed samples for inspection; those samples are not an accepted accuracy result.

A quadrature candidate_value is explicitly unaccepted when value is null. An initial IVP row is initial data, not a solved trajectory. An empty result is not a physical zero.

**DESKTOP WORKFLOW**

# 11 Algebraic roots mixed integers and examples

### Polynomial roots

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

The reference roots are -2, 1 and 3. The returned order is not a ranking. General algebraic discovery can return multiple solutions without a user starting guess, but a finite search is not universally exhaustive.

### A root far beyond the first two

```
(x-3)^6=3*x^4+1+exp(x-10)
bound x=[29,30]
```

The bounded root is approximately 29.7019590326. Without this bound, the equation also has roots near 1.4456401653 and 7.6713495999. A tiny residual can coexist with appreciable relative uncertainty in an ill-conditioned root; scaling matters.

### One integer and one continuous variable

```
int x
x+y=3.5
bound x=[0,2]
bound y=[0,5]
```

The three feasible points are (0,3.5), (1,2.5), (2,1.5). This is feasibility, not optimisation. Use File > Open for bundled examples not listed individually in the menu; V3_ files demonstrate the numerical additions.

**USER GUIDE**

# 12 Integer remainders and modular models

The % operator means mathematical integer remainder, not percentage. With positive modulus m, a%m is in 0 through m-1; for example (-4)%3=2. Declare every model variable integer when % is present.

```
pos int x,y
x%5=1
y%6=2
(y-x)%3=2
(x+y)%9=5
(x*y)%7=6
bound x=[1,100]
bound y=[1,100]
```

The four pairs in this box are (6,8), (51,80), (66,2) and (81,68). At (6,8), the five remainders are 1,2,2,5,6. A sufficient result cap and a completed search are needed to return the full bounded set.

| Rule | Meaning |
|---|---|
| Modulus | Positive constant integer from 1 to 1000000000 |
| Numerator | Integer polynomial operations; nonnegative integer powers up to 64 |
| Grouping | (x+y)%9 differs from x+y%9 |
| General modulo expressions | Supply finite explicit bounds for every variable |
| Automatic pure congruence mode | May enumerate one period of residue representatives |

A period rectangle is not an enumeration of an infinite set. Without bounds, this example has sufficient periods 315 in x and 126 in y. Read the modular stopping reason. General modulo ranges are limited to 20001 candidate integers per variable.

**USER GUIDE**

# 13 Integer products and affine factor shortcuts

Integer products can be much faster than a broad numerical search when a recognised factorisation structure is present. The cumulative engine recognises common affine rewrites, including offsets and supported auxiliary relationships.

```
pos int a,b
a*b=123456787

pos int a,b
a*(b+1)=123456787

pos int a,b,c
a*c=123456787
c=a+b

pos int a,b
a*(a+b)=123456787
bound a=[1,11111]
```

Each declaration and bound still applies to the original variable, not merely to the factor. For example b>0 means b+1>=2, while c=a+b restricts a divisor pair after recovery. The returned values must satisfy every original equation.

Do not introduce an auxiliary variable unless it expresses the intended model. Reusing an existing name for an integer quotient adds a constraint. If a form is not recognised by a shortcut, the existing search behaviour applies; a product-like appearance alone does not establish complete divisor enumeration.

The broader single-quadratic integer recogniser also covers many two-variable factored expressions. Additional equations or inequalities can select a different route. Inspect the method/arithmetic label and stopping reason.

**USER GUIDE**

# 14 Exact integer input and returned strings

The specialised methods parse preserved integer coefficient literals and verify candidate pairs with arbitrary-precision arithmetic. Prefer explicit literals when exactness matters. Do not replace a very large integer with a rounded decimal approximation.

| Input decision | Recommended practice |
|---|---|
| Integer domains | Use int, pos int or the intended signed restriction |
| Coefficients | Write exact integer literals in the polynomial |
| Large targets | Keep all decimal digits; no grouping commas |
| Finite search box | Write bounds for original variables |
| General quadratic model | Use one two-variable quadratic equation |
| Multiple conditions | Expect a different route unless explicitly supported |

### Exact literal bounds beyond floating point range

Version 3.0.1 accepts simple decimal/scientific bound literals beyond floating-point range on recognised exact two-variable integer quadratic, positive Pell and positive weighted-square routes. For example bound x=[1,1e500] works in ordinary double mode on an eligible model. This extension concerns bound literals; coefficient parsing and unsupported real/mixed-integer models retain their separate limits. Exact coefficient provenance remains important.

Numeric decimal/scientific bound literals are rounded inward exactly on the supported integer paths. An interval [1.2,5.8] therefore permits integers 2,3,4,5. Nonliteral bound expressions retain the evaluated parser semantics and cannot be assumed exact to unlimited digits.

### Exactness and completeness are separate

exact_integer_solutions confirms exact acceptance of the returned pairs. integer_search_complete describes completion in the declared integer domain and effective bounds. Positive-only routes may also expose positive_integer_search_complete. A solution/time/resource stop can leave either complete-looking short lists or empty lists incomplete.

Use precision=exact_integer, the stopping reason and integer-specific JSON completion fields. Exact coordinate verification does not imply complete enumeration or complete real-root coverage.

**USER GUIDE**

# 15 Arbitrary magnitude exact integer bounds

Enter original bound endpoints without calculating nearby floating values. On an eligible exact integer route, simple decimal or scientific literals are preserved and compared exactly even beyond double or binary128 range. Extended precision is not required merely to use these bounds.

```
pos int x,y
x^2-2*y^2=1
bound x=[1e499,1e500]
```

This interval means 10^499 <= x <= 10^500. It limits magnitude, not the number of returned rows. Maximum solutions controls the row cap. Use a sufficiently large cap for a complete bounded list and read integer_search_complete or the applicable positive-domain completion flag.

```
pos int x,y
x^2-2*y^2=1
bound x=[17.1,99]
```

Integer lower endpoints are rounded upward and upper endpoints downward. The second model excludes x=17 and includes x=99. Model bounds, Variables and Bounds, .pdeq save/reopen, CLI options and C ABI option strings preserve supported large literals. Dialog/CLI overrides still take precedence over model bounds.

| Accepted scope | Limit |
|---|---|
| Recognised exact routes | Two-variable integer quadratic, positive Pell and positive weighted squares |
| Literal notation | Simple decimal/scientific values; not a new arbitrary-precision expression language |
| Guards | 10000-character literal guard, 10000 magnitude digits, exponents within +/- 10000 |
| Unsupported large bounds | Real, mixed-integer, modulo and higher-dimensional numerical models are rejected |

Use 1e500 rather than 10^500 or (1e500) for this literal facility. Equation coefficient limits and per-method resource/time limits remain. A solution cap does not establish that the domain was exhausted.

**USER GUIDE**

# 16 Long values copying and the viewer limitation

Every result cell retains the full underlying value. Columns containing more than 120 characters use a bounded preview width capped at 520 device-independent pixels. Ordinary cells retain existing content/font sizing. The grid requests a horizontal scrollbar; resizing a column does not alter stored digits.

### The intended Full value dialog

Double-click a long result cell to open Full value. Its controls include a digit count for plain integers (otherwise a character count), resizable text area, Copy full value and Close. Its local Ctrl+A selects viewer text and Ctrl+C copies selection. The ordinary short algebraic-cell double-click opens solution details.

### Known issue in the supplied Windows release

The wrapped viewer has been reported to show only part of a solution. This guide does not claim a viewer fix. Visual wrapping and scrolling must therefore be treated as unreliable in 3.0.1. The Copy full value button is designed to use the raw value; verify critical data through a JSON export or plain-text editor.

### Preserve exact digits

| Task | Reliable workflow |
|---|---|
| Copy one value | Focus its Solutions cell and press Ctrl+C |
| Copy a rectangle | Select one rectangular block; use Ctrl+C or Ctrl+Shift+C |
| Copy several blocks | The current routine copies the first selected block; prefer one rectangle or export |
| Retain all rows | File > Export results / differential trajectory > JSON or CSV |
| Use a spreadsheet | Import long-integer columns as Text; automatic numeric conversion can lose digits |

Grid copying uses tabs between columns and a line break after each row. Without a rectangular selection it copies the current cell. Existing Ctrl+C is retained. The main-window Edit > Select All targets the editor, not the complete grid. Full JSON preserves decimal strings independently of the visible cell fragment.

A bound such as 1e500, a 500-digit coordinate and a 500-row list are different quantities. Keep digit counts, row caps and enumeration scope distinct.

**USER GUIDE**

# 17 Standard positive Pell equations

A standard Pell equation is $x^2-Dy^2=1$ with positive nonsquare integer D. Use strictly positive integer variables to select the positive Pell route. A real-valued model instead describes an entire curve.

```
pos int x,y
x^2-2*y^2=1
```

| First positive x | First positive y | Exact check |
|---|---|---|
| 3 | 2 | 9-2*4=1 |
| 17 | 12 | 289-2*144=1 |
| 99 | 70 | 9801-2*4900=1 |
| 577 | 408 | 332929-2*166464=1 |
| 3363 | 2378 | 11309769-2*5654884=1 |

### Ask for a prefix or a bounded set

Set Maximum solutions to 5 to see these first five positive pairs. Because the unbounded equation has infinitely many positive solutions, reaching that cap is not complete enumeration of the infinite family.

```
pos int x,y
x^2-2*y^2=1
bound y=[1,100]
```

With y<=100, only three positive pairs remain. Continued fractions determine the fundamental unit and recurrence; returned pairs are checked by exact integer arithmetic. Extended floating precision is not needed to preserve their integer digits.

**USER GUIDE**

# 18 Generalised Pell equations with integer N

The positive Pell route supports $x^2-Dy^2=N$ for positive integer D and integer N, including positive, negative and zero right-hand sides. Original coefficients are subject to the recognised checked signed-64-bit coefficient limits; derived coordinates can be much larger.

```
pos int x,y
x^2-61*y^2=3
```

The first pair is (8,1), since 64-61=3. General N can require several recurrence families; do not assume one smallest pair generates every solution.

```
pos int x,y
x^2-2*y^2=7
```

The engine merges the discovered unit-orbits and removes duplicate pairs. Positive Pell results are ordered by positive x when the full required representative discovery has been established.

```
pos int x,y
x^2-61*y^2=2
```

Squares modulo 4 are 0 or 1. Since 61 is 1 modulo 4, the left side cannot be 2 modulo 4. Here an empty result can be backed by a modular impossibility proof. A timeout with no pairs is a different outcome.

**USER GUIDE**

# 19 Negative zero and square Pell parameters

| Case | Mathematical form and route |
|---|---|
| D positive nonsquare | The genuine Pell setting; positive exact route or signed quadratic reduction |
| D positive square | Difference of squares; finite factor case for N nonzero |
| D zero | $x^2=N$ with a free y; not a genuine Pell recurrence |
| D negative | $x^2+abs(D)*y^2=N$; positive-definite quadratic setting |
| N zero with nonsquare positive D | No strictly positive integer pair |
| N zero with square positive D | A line of integer pairs can remain |

### Negative right side

```
pos int x,y
x^2-2*y^2=-1
```

### Square D

```
pos int x,y
x^2-y^2=15
```

### Negative D belongs to the sum-of-squares class

```
int x,y
x^2+2*y^2=9
```

The user may write integer D and N, but specialised Pell support does not mean an unrestricted all-signs Pell routine. Negative/zero D and signed branches use other recognised integer routes when applicable. Free-coordinate cases need finite bounds if a complete finite list is required.

**USER GUIDE**

# 20 Pell completeness limits and practical choices

For unbounded nonsquare positive D, a nonzero-N solution generally belongs to an infinite unit family. The engine can return a finite prefix, not print the entire infinite set. For bounded requests, completion means that every permitted pair in the effective integer domain has been enumerated or excluded.

| Positive Pell guard | Current limit |
|---|---|
| Direct or reduced representative scan | 1000000 rows |
| Reduced representatives | 10000 |
| Continued fraction or orbit merge steps | 100000 |
| Generated coordinate size | 10000 decimal digits |
| Other stops | Time, cancellation and requested solution count |

### Why an apparently simple instance can be hard

General N requires representative discovery before complete family merging. Large reduced ranges can exceed a resource guard even when the recurrence from a known pair is easy. The engine does not invent missing family seeds or call a resource-limited empty search a proof of impossibility.

### Read the report in this order

First confirm the original equation and integer declarations. Then check whether explicit bounds were applied. Next read the arithmetic label and stopping reason. Finally inspect integer_search_complete and, for the positive Pell path, positive_integer_search_complete. Keep the returned coordinate strings intact when exporting.

If you need only a few large exact pairs, choose a modest solution cap. If you need all pairs below a stated y, supply that upper bound and a cap large enough for the result. Increasing time cannot remove hard digit/representative guards, and an infinite family never becomes a finite exhaustive list.

For signed int x,y, zero coordinates and sign reflections can be valid. The general quadratic route verifies recovered pairs, but its unbounded discovery order is not promised to be the smallest original x order.

**USER GUIDE**

# 21 General quadratic equations in two integers

A generic equation of total degree two in two integer unknowns has the form below. “Degree two” means the largest sum of exponents in a term is two; it does not mean two equations.

```
A*x^2 + B*x*y + C*y^2 + D*x + E*y + F = 0
```

All coefficients are integers and at least one quadratic coefficient is nonzero. The cumulative exact recogniser accepts expanded and factored polynomial equivalents, terms on either side, signed integer domains and explicit bounds. It handles cross terms and linear offsets as well as diagonal forms.

```
int x,y
x^2+x*y+y^2=7
```

This positive-definite equation has 12 signed integer pairs. Examples include (1,2), (2,1), (-1,-2) and (-2,-1); the complete set includes other sign combinations. With pos int x,y, only (1,2) and (2,1) remain.

### What selects the specialised path

Use one polynomial equation in exactly two declared integer variables. Write exact integer coefficient literals or exactly divisible constant polynomial expressions. Arbitrary transcendental expressions, noninteger coefficients, additional equations and separate inequalities can leave this recogniser and retain the existing solver behaviour.

A general quadratic curve can have finitely many, infinitely many or no integer points. The recogniser is a collection of exact reductions and bounded searches, not a symbolic parametrisation of every conic.

**USER GUIDE**

# 22 Signed quadratics and bounded parabolas

### Indefinite equation reduced to Pell

```
int x,y
x^2+2*x*y-30*y^2=1
bound y=[-1000000,1000000]
```

The identity $(x+y)^2-31y^2=1$ explains the Pell connection. The finite y interval controls recovered solutions. Signed coordinates and y=0 branches are included; the latter give (1,0) and (-1,0).

### A parabola with one bounded coordinate

```
int x,y
y=x^2
bound x=[-3,3]
```

A finite interval for one coordinate can suffice: the remaining linear or quadratic equation is solved exactly at each integer slice. Here there are seven pairs, and no explicit y bound is needed.

### Methods chosen from the structure

Positive-definite forms reduce to weighted sums of squares; bilinear and square-discriminant forms can reduce to integer factors; nonsquare indefinite forms can reduce to generalised Pell when transformed coefficients fit its limits. Parabolic/degenerate forms often use exact slices. Every recovered pair is substituted back into the original exact polynomial.

Small finite intervals up to 100001 coordinates select direct exact slicing first. Larger cases can use reductions or a capped slice search. Pairs found by the general quadratic route are sorted lexicographically in declared variable order; that order need not follow discovery times.

**USER GUIDE**

# 23 Bounds and limits for general quadratics

A finite integer box is a mathematical domain. An automatic sample window is only a search choice. The distinction matters most for parabolic and degenerate forms with infinitely many points.

| Situation | Interpretation |
|---|---|
| Complete bounded exact enumeration | All permitted pairs in the effective domain returned |
| Positive definite complete reduction | A finite full signed set can be found without user bounds |
| Unbounded indefinite or degenerate family | A limited set can be returned without symbolic parametrisation |
| Limited empty search | No discovered pair, not a proof of impossibility |

## Current guards

The general quadratic route has a 100000 stored-pair guard, a 1000000-divisor guard and a capped slice search of up to 2000001 coordinates when applicable. Factorisation and inherited Pell/sum-of-squares guards also apply. The Desktop requests at most 10000 displayed solutions, so it can truncate a larger finite set before the engine’s storage guard matters.

When a lower bound is missing, an incomplete fallback can start at -10000 or 20000 below an existing upper bound. A missing upper endpoint can be replaced by a sample window 20000 above the lower endpoint. These are not proven bounds on all solutions. Supply a meaningful finite coordinate interval when exhaustive enumeration matters.

## A reliable request

Declare the signed domain, enter one exact polynomial, specify a useful finite bound, choose a sufficiently large solution cap and time allowance, and inspect integer_search_complete. Do not add a separate inequality merely as shorthand for a bound if retaining the specialised single-equation path is important.

Exact verification guarantees the returned integer identities, not successful completion for every coefficient combination or a guarantee of a particular runtime.

**USER GUIDE**

# 24 Positive weighted sums of squares

For positive integer weights A and B, the equation $Ax^2+By^2=C$ is a weighted sum of squares. The positive-only exact route uses number-theoretic factorisation and modular-root methods, including nonprimitive representations.

```
pos int x,y
2*x^2+3*y^2=35
```

The two positive pairs are (2,3) and (4,1). Both give 35 exactly. If x,y are unrestricted signed integers, their sign reflections must also be considered through the signed quadratic route.

```
pos int x,y
x^2+y^2=20
```

These are nonprimitive pairs because both coordinates have a common factor. They are deliberately retained; the solver is not restricted to coprime representations.

### A large target with exact digits

```
pos int x,y
x^2+61*y^2=35677541036108645912385710
bound x=[1,1e30]
```

The expected pair is x=1234567, y=764772987169. The identity is checked with exact integers; the target is not accepted through a rounded double residual. Difficulty depends strongly on factorisation, not just the number of decimal digits.

**USER GUIDE**

# 25 Signed zero and mixed square coefficients

The requested form allows integer A, B and C. The mathematical class changes when their signs or zeros change. The positive weighted-squares shortcut covers A>0, B>0 and strictly positive x,y; broader signed/zero cases use general quadratic or other existing routes where supported.

| Coefficients | Meaning |
|---|---|
| A>0 and B>0 | Positive-definite ellipse; C<0 has no real or integer pair |
| A and B of opposite sign | Indefinite form, often connected to Pell or factors |
| A<0 and B<0 | Multiply the equation by -1 and reverse C |
| Exactly one weight is zero | One square is constrained; another coordinate may be free |
| Both weights are zero | Constant identity or contradiction; no genuine quadratic remains |

### Include zero and negative coordinates

```
int x,y
x^2+y^2=25
```

There are 12 signed pairs, including (0,5), (5,0), (3,4) and their applicable sign/order variants. A pos int declaration excludes every zero and negative coordinate.

### A zero weight needs a finite free-coordinate domain

```
int x,y
x^2+0*y^2=9
bound y=[-2,2]
```

The finite request has x=3 or -3 and y in -2,-1,0,1,2. Without the y interval, there are infinitely many integer pairs. Do not infer a finite full list merely from a successful calculation.

For A=B=0 the remaining statement may not introduce any unknowns. Reformulate the actual intended model instead of relying on a quadratic recogniser for an identity such as 0=0.

**USER GUIDE**

# 26 Weighted square completion and exact checks

The specialised positive path normalises common factors, factors the target, considers square divisors, constructs modular roots and verifies recovered coordinates exactly. Imprimitive pairs and shared factors with coefficients are accounted for.

| Guard | Current limit |
|---|---|
| Counted arithmetic and search operations | 10000000 |
| Square divisor subproblems | 65536 |
| Modular roots per subproblem | 65536 |
| Stored pairs | 100000 |
| Other limits | Factorisation recursion, retries, time and cancellation |

### Prime proof and incomplete factorisation

A difficult factorisation can stop the calculation even when a solution exists. Larger probable primes require the route’s primality proof before complete enumeration is claimed; probable primality alone does not justify a full-set or no-solution statement. Decimal length by itself is not a performance predictor.

### Check a result without losing digits

Keep large values as integers in a language with arbitrary-precision support. For example, Python’s built-in int can verify x*x+61*y*y against the full 26-digit target from the preceding example. Do not convert the strings to float first. In spreadsheets, import the columns as text or use a separate exact-integer tool.

A completed positive-domain enumeration reports positive_integer_search_complete=true, and the current cumulative integer completion field also reflects that result. A solution cap, factoring/time stop or root-combination guard leaves completion false. The reason explains which limitation applied.

The general real search_complete field is not repurposed: an ellipse can have finitely many integer points and infinitely many real points at the same time.

**DIFFERENTIAL EQUATIONS**

# 27 Initial value ODEs

An ordinary differential equation describes variation with one independent variable. An initial-value problem supplies every state at one starting point. Begin with ode, name the independent coordinate in interval, and supply its matching initial value.

```
ode
y'= -y
initial t=0
initial y=1
interval t=[0,2]
method=RK45
rtol=1e-8
atol=1e-11
output step=0.1
```

The exact solution is y=exp(-t); at t=2 it is about 0.135335283236613. The table contains t and y, including the original initial row. output step controls reporting density for adaptive methods, not every internal integration step.

### Coordinate and state rules

The independent variable need not be t: use initial x and interval x together when x is intended. Initial time must equal the interval’s first endpoint in the selected arithmetic. Backward integration is supported with decreasing interval endpoints and a positive step magnitude when required.

ODEs accept model constants and ordinary real functions. Do not add integer declarations, algebraic bound directives or a second mode marker. Domain violations and genuine singularities require explicit diagnosis, not an arbitrary epsilon.

A successful IVP normally reports integration_successful=true. Check integration_complete and reached_time separately if a terminal event stopped the integration early.

**DIFFERENTIAL EQUATIONS**

# 28 Higher order and coupled ODEs

### Write the original second order equation

```
ode
y''+4*y=0
initial t=0
initial y=1
initial y'=0
interval t=[0,3]
output step=0.05
```

The solution is cos(2t), and the table includes y and y'. For an order-m equation supply initial derivatives through order m-1. Primes must immediately follow the state name. Direct orders one through eight expand into first-order state chains.

### Equivalent first order system

```
ode
u'=v
v'=-4*u
initial t=0
initial u=1
initial v=0
interval t=[0,3]
output step=0.05
```

The engine integrates states together. Couplings through lower derivatives are allowed. Each equation’s highest derivative must enter linearly; y''=-sin(y) is acceptable, while (y'')²=1, sin(y'')=y or 1/y''=x is not.

Implicitly coupled highest derivatives must first be isolated into separate equations. At most 64 expanded first-order states are supported. Internal names beginning __ode_ are reserved. This is not a differential-algebraic or implicit-branch solver.

**DIFFERENTIAL EQUATIONS**

# 29 Choose RK4 RK45 or SDIRK2

| Method | Use | Accuracy control |
|---|---|---|
| RK4 | Fixed step textbook or controlled step-halving experiments | No adaptive local error estimate |
| RK45 | Default nonstiff adaptive IVPs | Dormand-Prince embedded 5(4) estimate |
| SDIRK2 | Explicitly selected stiff IVPs | Second-order implicit step-doubling and continuous checks |

## Fixed step RK4

```
ode
y'=y
initial t=0
initial y=1
interval t=[0,1]
method=RK4
step=0.1
```

The final step shortens if needed. A step directive without method also selects RK4. rtol/atol do not control ordinary RK4 accuracy. Compare step=0.1 with 0.05. RK4 rejects output points and adaptive initial/min/max step controls; output step, if used, must be an integer multiple of step.

## Adaptive and stiff choices

RK45 is the ordinary default. Use rtol and atol for estimated local error, and max step to restrict internal steps. SDIRK2 must be selected explicitly; there is no automatic stiffness detection or switch. Its implicit Newton solves can fail and reduce the step.

Method order describes asymptotic behaviour under suitable regularity. No selected method supplies a certified global-error bound. For a supported singular IVP, the actual method may instead be regular_singular_series.

**DIFFERENTIAL EQUATIONS**

# 30 Stiff ODEs events and continuous output

### Fast forced decay with a known solution

```
ode
y'=-1000000*(y-cos(t))-sin(t)
initial t=0
initial y=1
interval t=[0,1]
method=SDIRK2
rtol=1e-6
atol=1e-9
output step=0.1
```

The exact solution is cos(t). SDIRK2 is a two-stage L-stable method with step doubling, damped Newton and analytic-expression Jacobians where available. It is not BDF or Radau. Tighter tolerances can require many second-order steps.

### Locate a ground crossing

```
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
```

The event occurs near sqrt(20/9.81)=1.427843123. View ODE events or export its CSV. Event directions refer to increasing independent-variable time. Sign-change detection can miss tangencies or multiple crossings inside one step.

Choose one reporting control: output points, output step or sample times. save dense=true retains interpolation coefficients in JSON. Never extrapolate a dense segment outside its stated interval. Plot lines join returned samples; output density improves display, not necessarily integration accuracy.

**DIFFERENTIAL EQUATIONS**

# 31 Automatic continuation at supported singular points

The regular Bessel initial-value example can start at the original singular endpoint without user-calculated nearby values. The engine analyses a supported regular branch rather than replacing zero by an arbitrary small number.

```
ode
x*y''+y'+x*y=0
initial x=0
initial y=1
initial y'=0
interval x=[0,1]
step=0.1
```

The regular expansion is $1-x^2/4+x^4/64-\ldots$, with $y''(0)=-1/2$. The original row (0,1,0) is retained. At x=1, y is about 0.7651976865579666 and y' about -0.4400505857449335.

| Diagnostic | Meaning |
|---|---|
| regular singular series _ _ | Actual method for a compatible supported regular continuation |
| singular ambiguous _ | Conditions leave continuation freedom |
| singular incompatible _ | No supported finite regular branch matches the data |
| singular unsupported _ | Structure or required branch analysis is outside scope |
| singular series limit _ _ | Finite-series or summation budget is not adequate |

Supported scope is a scalar first/second-order linear IVP with a simple affine leading coefficient and bounded-degree polynomial coefficients after supported rational algebra. Compatible interior/final crossings are analysed too. The series uses up to degree 64 with tail/roundoff estimates, not a global certificate.

x*y'=y with y(0)=0 permits y=C*x and is ambiguous. Singular IVP support does not extend the regular two-point BVP solver to singular endpoints. General nonlinear singularities, systems and fractional/logarithmic branches are excluded.

**BOUNDARY VALUE PROBLEMS**

# 32 Boundary value ODEs

A two-point BVP imposes conditions at both ends of a finite increasing interval. Begin with bvp. The engine uses adaptive fourth-order Lobatto collocation to compute the curve.

```
bvp
y''=-2
interval x=[0,1]
boundary left y=0
boundary right y=0
output step=0.1
```

The exact solution is x*(1-x). For a second-order scalar equation provide two independent boundary conditions. For a system, their count must match the total converted first-order states.

| Boundary syntax | Meaning |
|---|---|
| boundary left y=0 | Prescribed value |
| boundary right y'=0 | Prescribed derivative |
| boundary left y+2*y'=3 | Robin combination |
| boundary left(y)-right(y)=0 | Coupled endpoint relation |
| guess y=0 | Initial curve for nonlinear iteration |

Direct orders through eight and at most 16 converted states are supported. Highest derivatives must enter affinely. This BVP mode does not fit an unknown eigenparameter or impose interior/multipoint conditions or integral constraints.

Version 3 adds separate PDE normal/Robin directives and an eigen mode. PDE boundary signs use the outward normal; ODE BVP derivative syntax refers to the written coordinate derivative. Do not interchange these conventions.

**DIFFERENTIAL EQUATIONS**

# 33 Worked BVP with logarithmic boundary data

For $x^2y''+xy'=1$, y(1)=0 and y(1.4)=ln(1.4)/2, both endpoints are away from the singular coordinate x=0.

```
bvp
x^2*y''+x*y'=1
interval x=[1,1.4]
boundary left y=0
boundary right y=log(1.4)/2
rtol=1e-8
atol=1e-10
output step=0.02
```

### Exact reference solution

```
y(x) = 0.5*ln(x)*(ln(x)+1-ln(1.4))
```

Dividing the equation by x gives (xy')'=1/x. Two integrations give $y=(\ln x)^2/2+C1*\ln x+C2$. The left condition sets C2=0; the right condition sets C1=(1-ln 1.4)/2.

The numerical table includes x,y,y'. At x=1.4 the function is about 0.168236118310606. The exact endpoint condition is a useful check, but checking only endpoints does not assess the interior curve.

### Avoid a different problem

Do not enter initial y'=0 unless it is a genuine given condition: this is a BVP, not an IVP. Do not use step=... as an RK4 instruction in this mode. output step sets reporting spacing and does not fix the adaptive collocation mesh.

If this model is moved to an interval containing x=0, the singularity needs separate mathematical analysis. The regular BVP engine does not inherit the IVP regular-series continuation mechanism.

**DIFFERENTIAL EQUATIONS**

# 34 Worked BVP with Robin conditions

Boundary conditions may contain values and derivatives together. For y''-2xy'-2y=-4x, y(0)-y'(0)=0 and 2y(1)-y'(1)=1, write each whole combination at its own endpoint.

```
bvp
y''-2*x*y'-2*y=-4*x
interval x=[0,1]
boundary left y-y'=0
boundary right 2*y-y'=1
rtol=1e-8
atol=1e-10
output step=0.05
```

### Exact reference solution

```
y(x) = x + exp(x^2)
```

Then y'=1+2x*exp(x²) and y''=(2+4x²)*exp(x²). Substitution gives -4x in the differential equation. At x=0, y-y'=1-1=0. At x=1, 2(1+e)-(1+2e)=1.

### How endpoint shorthand works

In boundary left y-y'=0, both y and y' are evaluated at the left endpoint. A boundary expression is not an additional equation that must hold at every interior point. Use explicit left(expression) and right(expression) only when combining information from both ends in one boundary condition.

The default zero guess is sufficient for many linear BVPs. For a difficult or nonlinear problem, provide smooth starting guesses for each state; derivative guesses are independent and are not obtained automatically by differentiating guess y.

**DIFFERENTIAL EQUATIONS**

# 35 BVP guesses controls and convergence

### A nonlinear BVP can have multiple branches

```
bvp
y''=-exp(y)
interval x=[0,1]
boundary left y=0
boundary right y=0
guess y=0
rtol=1e-6
output points=101
```

This Bratu problem has a small branch with a peak near 0.14054. A different initial guess can converge to another branch. One converged curve does not enumerate all branches; Newton failure does not prove that none exist.

| Control | Default | Role |
|---|---|---|
| rtol / atol | 1e-6 / 1e-8 | Sampled continuous-defect scales |
| boundary tolerance | 1e-8 | Absolute residual of written endpoint equations |
| mesh points / max nodes | 11 / 1001 | Initial and adaptive mesh counts |
| newton iterations | 25 | Iterations per mesh |
| max refinements | 16 | Mesh refinement rounds |
| output points | 201 | Reporting samples, not solve mesh |

Success requires sampled scaled continuous defect <=1 and endpoint residuals within boundary tolerance. The method refines intervals whose defect is too large. Neither criterion is a certified global error bound, especially for ill-conditioned, resonant or rapidly varying solutions.

Current caps include max nodes<=4000, 6000 nodal unknowns and 500000 sparse entries/output cells. Failure returns bvp_successful=false with no converged trajectory. Read singular_jacobian, nonlinear_failure, mesh_limit or other reason before changing a setting.

**PARTIAL DIFFERENTIAL EQUATIONS**

# 36 PDE scope coordinates and fields

Version 3 supports explicitly defined scalar linear PDE classes on uniform grids. Dirichlet, outward-normal Neumann and supported nonnegative Robin conditions are available.

| Type | Supported form / role |
|---|---|
| poisson / laplace | a*(u_xx+u_yy)=f(x,y), equal constant nonzero a; rectangular domain |
| heat | u_t=a(x,t)*u_xx+b(x,t)*u_x+c(x,t)*u+f(x,t) |
| wave | u_tt=c^2*u_xx+f(x,t), constant c^2>0 |
| reaction_diffusion | Linear reaction and positive diffusion in 1D |
| advection_diffusion | Linear 1D advection with upwind spatial differences |

The unknown is u. Use u_x, u_xx, u_yy, u_t and u_tt literally. Poisson/Laplace use domain x and domain y; time equations use domain x and interval t. Domains must be finite and increasing.

### Grid counts include endpoints

grid x=41 means 40 intervals. For length L, dx=L/(grid x-1). The requested grid sets returned sample coordinates. Automatic refinement may compute much finer grids internally and return the finest values sampled at those coordinates.

Excluded: periodic boundaries, arbitrary geometry, nonlinear or coupled PDE systems, multidimensional time-dependent PDEs and moving boundaries. Coordinate-dependent coefficients must stay within each supported linear class.

Boundary syntax and signs are detailed in sections 59-60; accuracy and exact comparisons in sections 61-62.

**PARTIAL DIFFERENTIAL EQUATIONS**

# 37 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=17
grid y=17
exact solution=sin(pi*x)*sin(pi*y)
```

For Laplace use type=laplace without an explicit equation, or a zero source in the Poisson form. Left/right mean minimum/maximum x; bottom/top mean minimum/maximum y. Prescribed corner values must agree.

### Numerical method

The current method uses a conservative rectangular finite-volume discretisation and sparse preconditioned conjugate gradients. Natural normal/Robin contributions are assembled at boundaries. linear tolerance controls the linear-system residual, not continuum solution error.

Requested spatial counts are 3..401, subject to 10000 returned nodes. Internal refinement has separate max grid and max nodes limits. See section 49 for current defaults.

PDE fields / animation offers numerical, exact and signed-error maps when references exist, plus contours and spatial sections. Pure Neumann problems additionally require mean value and compatibility checks (section 60).

**USER GUIDE**

# 38 Laplace with boundary values from a diagram

A square diagram with five labelled interior edge nodes gives seven grid nodes per axis and 25 interior unknowns. With no physical side length stated, use coordinates 0..6 and unit spacing. A uniform rescaling of both axes preserves this Laplace stencil.

| Boundary | Direction | Five labelled values |
|---|---|---|
| Top | Left to right | 0 0 0 0 0 |
| Right | Top to bottom | 0 0 0 0 0 |
| Left | Top to bottom | 15.45 29.39 40.45 47.56 50 |
| Bottom | Left to right | 50 47.56 40.45 29.39 15.45 |

The current PDE grammar expects expressions rather than a table of boundary samples. A continuous piecewise-linear expression can encode the listed values at the grid nodes. The following constants are changes in successive interval slopes, not coefficients of the Laplace equation.

```
pde
type=poisson
u_xx+u_yy=0
domain x=[0,6]
domain y=[0,6]
constant a=2.44
constant b=4.67
constant c=3.95
constant d=2.88
constant k=1.51
boundary left u=50-a*max(0,y-1)-b*max(0,y-2)-c*max(0,y-3)-d*max(0,y-4)-k*max(0,y-5)
boundary bottom u=50-a*max(0,x-1)-b*max(0,x-2)-c*max(0,x-3)-d*max(0,x-4)-k*max(0,x-5)
boundary top u=0
boundary right u=0
grid x=7
grid y=7
refine=false
```

Unlabelled corners are set to 50 at bottom-left and zero at the other corners. These assignments do not enter the interior five-point equations. refine=false requests the diagram’s original grid without a second solve.

**USER GUIDE**

# 39 Read the square Laplace results

### Where the five values appear

The first interior column is x=1. Read its u values from y=5 down to y=1 to move from top to bottom. The Solutions table lists x fastest, so these five entries are not consecutive rows.

| Point | Computed u | Approximate reference |
|---|---|---|
| (1,5) | 11.71231 | 11.78 |
| (1,4) | 22.52554 | 22.66 |
| (1,3) | 31.71112 | 31.86 |
| (1,2) | 38.91588 | 39.05 |
| (1,1) | 44.45794 | 44.54 |

The middle and lower entries can be found by coordinate rather than by their position in an exported file. The symmetry of the left and bottom boundary data also gives u(x,y)=u(y,x) for this discrete problem.

### Why the boundary expression uses slope changes

Along the bottom, the assigned values at x=0,1,2,3,4,5,6 are 50,50,47.56,40.45,29.39,15.45,0. Unit-interval slopes are 0,-2.44,-7.11,-11.06,-13.94,-15.45. Their changes are -2.44,-4.67,-3.95,-2.88,-1.51. Each max(0,x-j) begins changing slope only after x=j.

For example at x=3, the expression gives 50-2.44*2-4.67=40.45. At x=6 it gives zero apart from floating-point rounding. The same construction in y supplies the left boundary.

The reference column is approximate and differs by more than two-decimal rounding from the converged discrete values. It may come from another iteration/stopping convention; those numbers alone do not identify the cause. The converged stencil solution is still an approximation to the continuous PDE.

**PARTIAL DIFFERENTIAL EQUATIONS**

# 40 Heat diffusion in one spatial 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=17
grid t=41
exact solution=exp(-pi^2*t)*sin(pi*x)
```

initial u supplies displacement at the first time endpoint. Dirichlet, normal and Robin data may depend on x,t and must agree with compatible initial data. For insulated boundaries use normal=0, not u=0.

### Method and supported linear terms

Crank-Nicolson time stepping with centred diffusion solves a tridiagonal system at each time level. For smooth solutions the base heat scheme is second order in space and time. Diffusion must be strictly positive; the leading u_t coefficient must be a nonzero constant.

The heat type allows centred advection only when the cell Peclet number is at most 2. Use the separate upwind advection_diffusion type or a finer x grid when advection dominates. a(x)*u_xx is not (a(x)*u_x)_x.

Implicit stepping is not a guarantee of positivity, accuracy or a resolved boundary layer. Use the refinement table and check space/time resolution. Animation plays computed frames; it performs no extra solve.

**PARTIAL DIFFERENTIAL EQUATIONS**

# 41 Wave propagation and the CFL condition

```
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=17
grid t=81
exact solution=cos(pi*t)*sin(pi*x)
```

Both initial displacement and velocity are required. The wave coefficient c^2 must be a positive constant. Dirichlet, normal and supported Robin conditions are accepted. Variable-speed waves, damping and coupled systems are outside this implementation.

### Courant condition: c*dt/dx <= 1

The centred explicit scheme uses a second-order initial Taylor step and checks the Courant condition on every internal grid. Increasing grid t reduces dt. Increasing only grid x reduces dx and can make stability worse.

Initial displacement must satisfy boundary data; initial velocity must satisfy the time derivative of the boundary relation. Noncompatible or insufficiently regular initial boundary data can be rejected.

A stable wave can still have phase error. Refinement and exact-reference comparisons assess accuracy; the CFL condition alone does not.

**PARTIAL DIFFERENTIAL EQUATIONS**

# 42 PDE refinement residuals and grid limits

Automatic refinement is enabled by default. The solver doubles subdivisions in both space and time, or both steady spatial directions. It compares consecutive internal solutions and checks the estimated error at returned nodes against your target.

```
absolute tolerance=1e-3
relative tolerance=1e-3
max refinements=4
max grid=801
max nodes=250000
linear tolerance=1e-11
refine=true
```

| Quantity | Interpretation |
|---|---|
| Estimated discretisation error | Fine-grid difference / 3 for second-order schemes; / 1 for first-order upwind spatial convergence |
| Returned grid | Requested output coordinates, sampled from the finest computed solution |
| Maximum discrete residual | Residual of discrete equations; not a continuum error bound |
| accuracy_target_met | Whether the estimated target was met; null for refine=false |
| accuracy_not_met | Computed samples can remain available; field_successful is false |

The defaults above apply to PDE targets in ordinary double mode. Linear tolerance defaults to 1e-25 in binary128. max nodes is an internal grid budget; returned field data are capped at 10000 nodes. See section 61 for componentwise target semantics.

This replaces the older single-comparison, coarse-grid estimate. Automatic refinement is numerical evidence under convergence assumptions, not a certified global bound.

**INTEGRAL EQUATIONS**

# 43 Fredholm integral equations of the second kind

The convention is y(x)=f(x)+lambda*integral_a^b K(x,s)*y(s) ds. The unknown is y; x is the output coordinate, and s is the integration coordinate.

```
integral
type=fredholm
interval x=[0,1]
forcing=1+x/6
kernel=x*s
lambda=1
grid x=17
exact solution=1+x
```

For y=1+x the integral is 5*x/6, so the reference satisfies the equation. The reference is optional and is used only to calculate comparison errors.

## Method and accuracy

Composite trapezoidal Nystrom discretisation forms a bounded dense linear system solved with scaled partial pivoting. Successive solutions are Richardson-extrapolated to remove the leading second-order term. Differences between extrapolants estimate the remaining error.

Default absolute and relative targets are each 1e-9; max grid defaults to 513. Smooth finite kernels and a finite interval are required. Fredholm remains scalar. A near-singular operator can amplify small quadrature errors.

A singular discrete pivot reports singular_system; it is not a general proof that the continuous equation has no solution. Numerical definite integration of a known integrand belongs in quadrature mode instead.

**INTEGRAL EQUATIONS**

# 44 Volterra integral equations of the second kind

The convention is y(x)=f(x)+lambda*integral_a^x K(x,s)*y(s) ds. Its variable upper limit permits forward marching.

```
integral
type=volterra
interval x=[0,1]
forcing=1
kernel=1
lambda=1
grid x=65
exact solution=exp(x)
```

The exact reference is exp(x), with y(0)=1. At x=1 the reference is approximately 2.71828182845904523536. Version 3 automatically refines and extrapolates; grid x specifies returned nodes, not the entire internal accuracy budget.

### Why the earlier 65-node result differed

The raw trapezoidal 65-node calculation gives about 2.7183371346324106; 257 nodes give about 2.7182852849432404. The error reduction is consistent with second-order discretisation. Extra printed digits or arithmetic precision alone cannot remove it.

The current method includes the unknown endpoint term in each trapezoidal step, then uses successive Richardson extrapolants. The default internal max grid is 4097. Coupled linear systems with up to eight components are now supported (section 68).

Finite smooth kernels are required. Weakly singular kernels, first-kind or nonlinear integral equations, infinite intervals and principal values are outside this mode. Quadrature infinity support does not extend integral-equation domains.

**INTEGRAL EQUATIONS**

# 45 Interpret integral equation accuracy

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.

**AUTOMATION AND INTEGRATION**

# 46 Command line use with the same models

The CLI accepts the same model text as the Desktop. Run from dist or use the full executable path. A .pdeq project is a Desktop container; save its model as text for CLI use.

```
equation_solver.exe --file model.txt --json --seconds 20
equation_solver.exe --file heat.txt --extend --json --seconds 30
equation_solver.exe --file quadrature.txt --json --seconds 20
```

| Option | Purpose |
|---|---|
| --file PATH / --file - | Read a model file / standard input |
| --eq TEXT | Use a quoted model string |
| --check | Validate syntax without numerical solving |
| --json | Machine-readable JSON on standard output |
| --seconds N | Shared work budget; 0 removes the deadline |
| --max-solutions N | Algebraic / integer row cap |
| --extend | Entitled binary128 solving |
| --style US / --style EU | Input number notation |

Put mode-specific controls in the model. Algebraic --tol does not replace IVP/BVP rtol/atol, field absolute tolerance or quadrature tolerances. A sweep shares the complete run budget across all combinations.

Read problem_type and mode-specific success fields. Capture stdout and stderr separately. Input/licensing errors and numerical failures are different outcomes; do not infer success from the presence of numerical strings alone.

**AUTOMATION AND INTEGRATION**

# 47 DLL integration and structured results

```
eq_result* r = eq_solve(model_utf8, option_count, options);
if (r) {
    int status = eq_result_status(r);
    const char* json = eq_result_json(r);
    // Copy any needed strings before freeing r.
    eq_result_free(r);
}
```

Include equation_solver.h. Check for a null handle. Result strings remain owned by the handle until eq_result_free; do not free them separately. C ABI v1 ownership and licence checks are unchanged.

| problem_type | Primary result / success |
|---|---|
| Algebraic / integer result | solutions[].values; inspect status and completeness |
| ode_ivp | trajectory; integration_successful |
| ode_bvp | trajectory; bvp_successful |
| pde / integral_equation / eigenproblem | Field schema; field_successful and accuracy_target_met |
| parameter_sweep | Sweep schema with nested per-run results and statuses |
| quadrature | value; quadrature_successful and accuracy_target_met |

Map value positions using variables and retain numerical decimal strings. Field and eigen data use equation_solver.field_result.v1; sweeps use equation_solver.sweep_result.v1; quadrature uses equation_solver.quadrature_result.v1. Branch on problem_type rather than assuming every result is a root list.

Quadrature value is null on failure. candidate_value is not accepted. For field accuracy_not_met, computed samples can be present despite field_successful=false. Always inspect status, reason and the target flag.

**REFERENCE**

# 48 Reference controls for ordinary ODEs

| Directive | Default or scope | Meaning |
|---|---|---|
| method | RK45 | RK4 RK45 or SDIRK2 |
| step | Required for fixed RK4 | Positive integration-step magnitude |
| rtol / atol | 1e-7 / 1e-10 | Adaptive local scales; RK4 does not use them for step control |
| atol state | Optional | Per-state absolute tolerance, including derivative labels |
| initial step | Automatic | Adaptive first-step magnitude |
| max step / min step | Automatic / 0 | Adaptive step bounds |
| max steps | 100000 | Attempted-step cap, up to 10000000 |
| newton iterations | 12 | SDIRK2 stage Newton limit, 2..50 |
| output points | 201 | Ordinary adaptive reporting grid, 2..9999 |
| output step | Optional | Alternative positive reporting spacing |
| sample times | Optional | Explicit ordered reporting list |
| save dense | false | Retain segment polynomials in JSON |
| event name | Optional | Root function for a named event |
| event name terminal | false | Stop at a located event |
| event name direction | any | any rising or falling |
| event tolerance | 1e-10 double / 1e-24 extended | Absolute coordinate root-location tolerance |

Choose one reporting specification; do not combine output points, output step and sample times. Fixed RK4 has additional restrictions described earlier. State expansion, output-cell and dense-storage caps can limit an otherwise valid model. Singular-series handling has distinct actual-method and error semantics.

These defaults are mode defaults, not the Desktop algebraic tolerance. Changing Compute > Solver Settings residual tolerance does not replace ODE rtol and atol.

**REFERENCE**

# 49 Reference controls for BVP PDE and integrals

| Mode / control | Default / meaning |
|---|---|
| BVP rtol / atol | 1e-6 / 1e-8; sampled continuous defect |
| BVP mesh points / max nodes | 11 / 1001; max nodes <=4000 |
| BVP boundary tolerance | 1e-8 absolute endpoint residual |
| BVP newton iterations / max refinements | 25 / 16 |
| PDE grid x / grid y / grid t | 21 / 21 / 101; requested output nodes |
| Integral grid x / eigen grid x | 65 / 65; eigen modes defaults to 4 |
| PDE / eigen absolute and relative tolerance | Each 1e-3 |
| Integral absolute and relative tolerance | Each 1e-9 |
| max refinements: PDE / eigen / integral | 4 / 4 / 6; allowed 1..10 |
| max grid: PDE / eigen / Fredholm / Volterra | 801 / 1025 / 513 / 4097; internal axis cap |
| max nodes | 250000 default; up to 500000 internal nodes |
| Returned field nodes | At most 10000 |
| linear tolerance | 1e-11 double; 1e-25 binary128 |
| max iterations / boundary tolerance | 10000 / 1e-8 for applicable field checks |
| refine | true; false leaves target unassessed |

Requested PDE spatial axes permit 3..401 nodes (time PDE x requires at least 5); time counts permit 2..10001. Integral requested x permits 3..513; eigen x permits 3..1025. Totals and internal budgets can restrict valid combinations further.

These are mode defaults, not algebraic Solver Settings. BVP output points do not set its collocation mesh. Field grid directives set returned coordinates while automatic refinement uses finer internal grids. Quadrature controls are separate (section 72).

**TROUBLESHOOTING**

# 50 Diagnose numerical and input failures

| Status / symptom | Check or action |
|---|---|
| Missing runtime / launch failure | Extract and keep the complete dist folder |
| Extended mode rejected | Check entitlement and activation |
| Syntax / unexpected variable | Check spelling, mode, number style and products |
| Few roots / integer pairs | Read bounds, time, result cap and completion flags |
| IVP nonfinite_derivative | Check function domains and singular-IVP scope |
| IVP step_size_failure / newton_failure | Inspect stiffness, scaling and step limits |
| BVP singular_jacobian / nonlinear_failure | Check independent boundary conditions and guesses |
| Field accuracy_not_met | Inspect convergence and grid/time/roundoff limits |
| Field incompatible_conditions | Check corners, wave velocity and Neumann balance |
| Field compatibility_not_resolved | Neumann compatibility quadrature unresolved |
| PDE unstable_grid | Reduce dt, checking c*dt/dx <=1 |
| Integral singular_system | Inspect conditioning or the Volterra diagonal |
| Quadrature nonfinite_integrand | Check domains, cancellation, poles and overflow |
| Quadrature tail_not_resolved | Check tails, scaling, extent and convergence |

Retain the exact model and full reason. A failure diagnostic is not automatically a proof of mathematical divergence or nonexistence. A plausible unaccepted candidate is still unaccepted.

Do not shift a genuine singularity by an arbitrary epsilon. Algebraic reformulation or a change of variable must preserve the intended mathematics. See section 76 for the reported endpoint-cancellation issue.

**TROUBLESHOOTING**

# 51 When numbers differ from a reference

### Check the mathematical model first

Confirm coefficients, signs, powers, boundaries, outward-normal orientation, integer declarations and interval endpoints. Coordinate order in a table can differ from the order in a diagram. A changed boundary changes the solution.

### Compare like numerical problems

A coarse discrete Laplace system and a fine continuum approximation can disagree. ODE output spacing differs from internal stepping; BVP output density differs from its mesh. Field output coordinates differ from automatically refined internal grids.

| Problem | Useful controlled comparison |
|---|---|
| Roots | Meaningful bounds, scaling and tighter residual checks |
| RK4 IVP | Halve step at matching sample points |
| Adaptive IVP / BVP | Tighten tolerances; inspect steps/mesh and branches |
| PDE / eigenvalue | Inspect successive internal grids and estimated order |
| Integral equation | Compare extrapolants; use refine=false for raw-grid studies |
| Quadrature | Inspect tail/phase errors; reformulate cancellation if justified |
| Exact integer identity | Substitute with arbitrary-precision integers |

Exact solution, exact eigenvalue and exact eigenfunction fields are optional reference data. They are valuable for manufactured tests and teaching even though most applications have no closed form. Without them, use refinement, residuals, conservation checks and independent methods.

Do not choose a result merely because it matches an expected number. Check how the reference was obtained, its precision, and the solver status. A previous completed snapshot does not update when you edit the model.

**REFERENCE**

# 52 Keyboard commands and support

| Action | Shortcut |
|---|---|
| New / Open / Save | Ctrl+N / Ctrl+O / Ctrl+S |
| Save As / Print | Ctrl+Shift+S / Ctrl+P |
| Undo / Redo | Ctrl+Z / Ctrl+Y |
| Cut / Copy / Paste / Select all | Ctrl+X / Ctrl+C / Ctrl+V / Ctrl+A |
| Find / Replace / Go to line | Ctrl+F / Ctrl+H / Ctrl+G |
| Find next / previous | F3 / Shift+F3 |
| Check syntax | F4 |
| Solve | F5 or Ctrl+Enter |
| Stop | Esc during an active solve |
| Copy selected result cells | Ctrl+Shift+C |
| Comment selection | Ctrl+/ |
| Quick Start / Exit | F1 / Alt+F4 |

### Send a reproducible support request

Provide the exact model text, product version, problem mode, number style, precision, settings, bounds and the full stopping reason. For PDE/integral comparisons include the node counts, refinement setting and reference method. For exact integer requests include the full coefficient/target digits and whether all signed points or only positive pairs are wanted.

Help > Copy Diagnostics can include model text, file paths and licence/customer details. Review it before sending. Preserve a .pdeq project and a JSON result when reporting a repeatable issue.

`panna@patodiainfo.com`

Patodia Infotech Private Limited, Bengaluru, India. Product information: https://patodiainfo.com/EquationSolver/. Guide edition 4.0, 11 November 2026.

**REFERENCE**

# 53 Desktop File Edit Search and View commands

| File command | Purpose |
|---|---|
| New / Open / Open Recent | Start a model, load a file or reopen a recent path |
| Save / Save As | Save text or a .pdeq project; choose a new path/type |
| Import equations | Insert UTF-8 text at the current cursor/selection |
| Export results / differential trajectory | CSV table or complete returned JSON |
| Export run report | Save the completed report as text |
| Export ODE events CSV | Save located event records separately |
| Print | Choose completed run report or current equations |
| Exit | Unsaved-work and active-task prompts apply |

| Edit Search or View command | Purpose |
|---|---|
| Undo / Redo | Undo or redo editor changes |
| Cut / Paste | Operate on editor text |
| Copy | Copy selected data from a supported focused control |
| Select All | Select all equation-editor text |
| Copy selected result cells | Copy the current grid cell or first selected rectangle |
| Comment / uncomment | Toggle // on selected lines |
| Indent / Unindent selection | Change selected-line indentation in four-space steps |
| Find / Next / Previous / Replace | Search the editor with case/whole-word options and wraparound |
| Go to line | Move to a specified editor line |
| Reset equation and solution fonts | Restore and save font defaults |

Open accepts model/project files; it is not a PDF/Word or exported-result viewer. Imported text must use the active number style. UTF-8 input is limited to 5 MiB. A .pdeq file preserves model, settings and overrides, not completed result tables or plots. Save exports separately.

**REFERENCE**

# 54 Desktop Solver Settings and Compute commands

Fresh Desktop defaults are shown below. Saved preferences and projects can restore other values; they are not necessarily CLI defaults. Root-search controls do not replace ODE/BVP/PDE/integral directives in the model.

| Control | Default | Meaning |
|---|---|---|
| Time budget | 2 seconds | 0 removes the time deadline, not other resource limits |
| Maximum solutions | 100 | Returned algebraic cap; Desktop permits 1..10000 |
| Maximum starts | 30000 | Multistart budget |
| Iterations per start | 300 | Local search ceiling |
| Residual tolerance | 1e-9 | Algebraic acceptance scale |
| Random seed | 42 | Randomised search reproducibility |
| Automatic radius | 10 | Initial automatic range |
| Range growth / stages | 10 / 4 | Expansion factor and stage count |
| Integer neighbours | 2 | Nearby integer assignments |
| Deflation / inference / slices | Enabled | Search aids; not completeness proofs |
| Extended precision | Off | Eligible binary128 arithmetic; standard tolerance can switch to 1e-24 |

### Other Compute commands

Check Syntax (F4) validates grammar/configuration. Variables and Bounds parses algebraic variables and edits overrides; enter both endpoints or leave both blank. Precision selects double/binary128. Number Style converts current editor notation between US/EU. Solve starts a captured run; Stop force-terminates it. Clear Results clears displayed results. View Run Log selects Diagnostics. For Equation Generator and Fun with Mathematics, see Chapter 88.

Variable/type cells in the bounds dialog are read-only; declare integer and binary domains in the editor. Explicit overrides remain fixed during automatic range expansion. Differential and field accuracy controls belong in their model text; time and precision remain relevant Desktop controls.

While a worker is active, model-opening, new solves, settings, number-style changes and licence actions are disabled. Editing does not change the active worker snapshot. Force-Stop cannot retrieve unfinished partial results.

**DESKTOP REFERENCE**

# 55 Desktop Help Examples and licence locations

Help provides the Quick Start, User Guide PDF, equation/function references, ODE/BVP/singular-point guides, PDE/integral documentation, integer-value guides and examples. Open supplied V3_ files directly when they have no dedicated menu entry.

| Examples | Purpose |
|---|---|
| V3_01 to V3_05 | Neumann, Robin, mixed PDEs and mean constraints |
| V3_06 to V3_10 | Transport, integral accuracy and coupled Volterra |
| V3_11 to V3_14 | Eigenproblems and parameter sweeps |
| V3_15 to V3_23 | Finite/improper quadrature and failure diagnostics |

Loading a menu example clears previous overrides and results after the unsaved-work prompt. Work from personal copies when editing supplied models.

### Licensing and support

Help > Licence Information shows entitlement; Activate Licence selects a signed licence; Machine Fingerprint obtains the individual identifier; About identifies the application. Include engine file version as well when reporting a build-specific issue.

```
%LOCALAPPDATA%\PatodiaEquationSolver\license.txt
```

An explicit CLI --license path or EQ_LICENSE_FILE can override licence selection. Eligible automatic trials do not require a licence file. Reinstallation is not a valid trial reset.

Commercial permits double Desktop/CLI computation; extended and DLL solving require an eligible entitlement. Inspect Copy Diagnostics before sharing because it can include model paths and customer details.

**DESKTOP REFERENCE**

# 56 Desktop plots exports and completed snapshots

## Curves and fields

The ODE / BVP / integral plot supports curves and phase portraits, integral-system components and individual eigenfunctions. Select the intended variables. Numerical and matching exact curves can be overlaid; error_y or error_u selects signed-error data when available.

PDE fields / animation offers numerical, exact and signed-error heat maps, contour overlays and spatial sections. Time sliders and Animate/Pause step through computed heat, wave and transport frames. PNG export saves the map.

## Convergence and sweep comparisons

Convergence / eigenpairs / sweeps displays refinement levels, internal grids, error estimates and mode data. Export its active table as CSV or comparison chart as PNG. Parameter sweep summaries retain case status; select a row to inspect that case or overlay runs.

| Saved record | What it preserves |
|---|---|
| .pdeq project | Model, settings, number style and overrides |
| Results JSON | Returned data and completed run snapshot |
| Results CSV | Returned numerical table; strings need careful import |
| Run report / events CSV | Readable diagnostics / located events |
| Panel PNG / CSV | Displayed chart or active numerical comparison table |

Animations do not recompute. Plots use display precision. A common colour range within one field does not make colours from separate runs quantitatively comparable. Force Stop can leave the previous completed snapshot visible.

**REFERENCE**

# 57 Scope and validation of this edition

This guide updates the attached 2.0.1 edition for Version 3.0.1 and numerical engine build 3.0.1.1. Existing algebraic, integer, ODE and BVP material is retained; changed PDE/integral guidance and the Version 3 reference chapters are integrated into the contents and bookmarks.

| Checked material | Used for |
|---|---|
| Attached guide | Retained exact-integer, IVP/BVP and Desktop workflows |
| VERSION3_GUIDE and field_v3 source | Boundary conventions, eigen/sweep scope and refinement |
| QUADRATURE_GUIDE and quadrature source | Syntax, controls, methods and acceptance semantics |
| Release audit / regression records | Completed checks and platform limits |
| Reported endpoint test | Open limitation and verified reformulations |

## Evidence and its limits

The numerical build record reports 69 quadrature cases per precision mode through CLI and DLL, 43 Version 3 cases per mode/interface, 104 field CLI cases, 16 field DLL cases, 15 root checks and 23 teaching examples. These are recorded engine tests, not Windows graphical acceptance.

Windows EXE/DLL files were cross-compiled and inspected. Windows execution, GUI interaction and account-specific activation remain acceptance tasks. No universal PDE solver, symbolic integration engine or certified global-error guarantee is claimed.

Review edition dated 1 November 2026. The release candidate is undergoing extended testing before public release. The reported x/sqrt(1-sqrt(x)) Windows failure remains open; this guide does not claim a fix.

Keep the package, guide, model and result together when reproducing a test. New correctness changes should be reflected in both examples and regression records.

**VERSION 3 NUMERICAL WORKFLOW**

# 58 What is new in Version 3.0.1

The focus is reliable numerical solutions, transparent accuracy checks and visual exploration. Existing algebraic and exact-integer features remain available. Chapters 87–88 cover multi-range quadrature, declared singularities, standalone singularity analysis and the new Desktop tools. This update also improves limit evaluation and adds numerical fallback and investigation steps.

| Addition | Practical value |
|---|---|
| Neumann and Robin PDE boundaries | Insulation, derivative data and convection |
| Automatic refinement and accuracy targets | Estimated error checked against a requested target |
| Convergence / exact comparisons | Observed grid behaviour, maximum and RMS errors |
| Numerical visualisation | Heat maps, contours, time playback and error curves |
| Sturm-Liouville eigenproblems | Natural frequencies, modes and eigenfunction plots |
| Reaction / advection-diffusion | Linear one-dimensional transport with sources or decay |
| Coupled Volterra systems | Linear second-kind systems with memory |
| Parameter sweeps | Repeated solves with summaries and comparison plots |
| Standalone quadrature | Known-function finite and improper definite integrals |

### A useful first session

Open V3_01_Heat_Neumann.txt, solve it and compare numerical and exact fields. Inspect its convergence table. Next open V3_11_Sturm_Liouville.txt and view a mode. Finally run V3_15_Quadrature_Endpoint.txt and read the scalar integral and estimated error.

Unit checking, fluid-property databases, general interpolation, regression, uncertainty analysis and optimisation were deliberately deferred. There is no separate Student edition described by this guide.

**VERSION 3 BOUNDARIES**

# 59 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.

**VERSION 3 BOUNDARIES**

# 60 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.

**VERSION 3 ACCURACY**

# 61 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.

**VERSION 3 ACCURACY**

# 62 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.

**VERSION 3 VISUALISATION**

# 63 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.

**VERSION 3 EIGENPROBLEMS**

# 64 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.

**VERSION 3 EIGENPROBLEMS**

# 65 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.

**VERSION 3 TRANSPORT**

# 66 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.

**VERSION 3 TRANSPORT**

# 67 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.

**VERSION 3 INTEGRAL SYSTEMS**

# 68 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.

**VERSION 3 PARAMETER STUDIES**

# 69 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.

**VERSION 3 QUADRATURE**

# 70 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. Use quad as a short name and see Chapter 87 for multiple ranges and known interior singularities.

### 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.

**VERSION 3 QUADRATURE**

# 71 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. Use range to enter several explicit intervals, or supply one range with a singularities list as shown in Chapter 87.

**VERSION 3 QUADRATURE**

# 72 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.

**VERSION 3 QUADRATURE**

# 73 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.

**VERSION 3 QUADRATURE**

# 74 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. Targeted fallback strategies can help in difficult cases. Read the final status and acceptance fields after any fallback.

**VERSION 3 QUADRATURE**

# 75 Quadrature reference examples and tail bounds

| Integrand | Interval | Reference |
|---|---|---|
| exp(-x^2) | [-inf,inf] | sqrt(pi) |
| exp(-x) | [0,inf] | 1 |
| exp(x) | [-inf,0] | 1 |
| 1/sqrt(x) | [0,1] | 2 |
| log(x) | [0,1] | -1 |
| 1/(1+x^2)^5 | [0,inf] | 35*pi/256 |

```
quadrature
integrand=1/(1+x^2)^5
variable=x
interval=[0,inf]
```

The corrected numerical engine build 3.0.1.1 accepts this original input at default settings. Its reference is approximately 0.42951462060797954432. Earlier builds could overflow the intermediate denominator even though the reciprocal is tiny.

For a finite cutoff R>0, the omitted right tail is less than 1/(9*R^9), because (1+x^2)^5>x^10. R=10^20 therefore gives a tail below approximately 1.12e-181. That bound concerns truncation only, not the quadrature or arithmetic error.

### Nested exponential example

```
quadrature
integrand=exp(-0.0009*t^2-t-0.5*exp(-t))
variable=t
interval=[-10,inf]
```

For this example the omitted left tail is below 10^-4782. Such a tiny mathematical tail does not imply hundreds of accurate computed digits: binary128 provides only about 34 significant decimal digits of arithmetic.

**VERSION 3 QUADRATURE**

# 76 Endpoint cancellation: an open test case

```
quadrature
integrand=x/sqrt(1-sqrt(x))
variable=x
interval=[0,1]
```

The exact value is 64/35, approximately 1.82857142857142857143. The endpoint singularity behaves like sqrt(2)/sqrt(1-x), so it is integrable.

### Reported Windows failure

A Windows run reported nonfinite_integrand: division by zero at an interior quadrature point. The last candidate was already within about 2.77e-17 of the reference. Cancellation in 1-sqrt(x) near x=1 is consistent with the failure. A native Linux rebuild accepted the original input; the Windows-specific failure has not yet been reproduced and fixed.

### Mathematically equivalent workaround

```
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.

Keep this original model as a release regression test. This guide does not claim that the current package automatically stabilises every cancellation or that the reported Windows defect is fixed.

**RELEASE-CANDIDATE TESTING**

# 77 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.

### 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.

**VERSION 3 ADDITIONS**

# 78 Numerical differentiation of expressions

Use diff or differentiate as the first line; expr and expression are aliases. Give one coordinate for every variable. wrt selects a partial derivative, while gradient and hessian request the complete vector or matrix.

```
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
```

For an ordinary partial derivative use wrt = x and order = 1 through 4; wrt = [x,z] denotes a mixed partial. grad/gradient and hess/hessian are equivalent. First derivatives use automatic differentiation; higher orders use extrapolated numerical differences and have heuristic, not certified, error estimates.

**VERSION 3 ADDITIONS**

# 79 Differentiation of tables and CSV measurements

```
diff
file = "tests/tabular_research_data.csv"
x_column = time
```

```
y_column = displacement
error_column = sigma
method = savitzky_golay
window = 7
degree = 2
at x = [3,5,7]
order = 1
uncertainty = true
output_csv = "derivatives.csv"
```

Alternatively, use data = [[0,0],[1,1],[2,4],[3,9],[4,16]] for inline pairs. Methods include auto/local_polynomial, finite_difference (an alias to the local fit), savitzky_golay and smoothing_spline. The smoothing spline uses lambda (default 1) and supports derivative orders 1–2; higher derivatives are not supported by that spline method. Local polynomial estimates support orders 1–4 when sufficiently sampled.

Input x values are sorted; repeated x values and unsupported extrapolation are rejected. error_column or inline [x,y,sigma] values represent positive standard deviations. Propagated uncertainty assumes independent input errors and excludes interpolation or physical-model bias.

**VERSION 3 ADDITIONS**

# 80 Interpolation and measured data

```
interp
data = [[0,0],[1,1],[2,4],[3,9],[4,16],[5,25]]
at x = [1.5,2.5,3.5]
method = local_polynomial
```

This quadratic data set gives 2.25, 6.25 and 12.25. interp and interpolate are aliases. Supported methods: linear, local_polynomial (also auto), pchip, cubic_spline, akima, barycentric and smoothing_spline. Local polynomial uses window and degree; smoothing spline uses lambda. Akima requires at least five points; barycentric is limited to 32 samples; local fitting limits the window to 15.

```
interpolate
file = "tests/interpolation_sample.csv"
x_column = x
y_column = y
method = pchip
at x = [1.5,2.5]
output_csv = "interpolation_results.csv"
```

Interpolation accepts irregular spacing and sorts x observations. Extrapolation is off by default and can be enabled only for linear and local_polynomial. uncertainty = true is supported with positive input sigmas for linear, local_polynomial, barycentric, cubic_spline and smoothing_spline, but not PCHIP or Akima. A smooth plotted curve does not establish a physical model or a certified error.

**VERSION 3 ADDITIONS**

# 81 Numerical limits and one sided behaviour

```
limit
expr = abs(x)/x
at x = 0
direction = both
```

limit and lim are aliases; expr/expression are accepted. For a single variable, at x = 0 or variable = x together with at = 0 is valid. direction = left, right or both selects the approach. at = inf, infinity, -inf or -infinity selects an infinite endpoint. Classifications are finite_limit, positive_infinity, negative_infinity, does_not_exist and undetermined.

```
lim
expr = (3*x^2+2*x+1)/(x^2+5)
at x = infinity
```

The rational example tends to 3. The preceding abs(x)/x example has right value 1, left value -1, hence no two-sided limit. The quotient (exp(x)-1)/sqrt(1-cos(x)) also has unequal sided limits at 0: +sqrt(2) from the right and -sqrt(2) from the left; its two-sided limit does not exist. A two-variable limit may reveal conflicting sampled paths, but a finite set of agreeing paths cannot prove existence. Limit diagnostics are numerical/structural evidence; not formal certification. The updated limit evaluator improves numerical handling. Use the reported evidence and the applicable one-sided results when deciding whether the conclusion is usable.

**VERSION 3 ADDITIONS**

# 82 Compute Functions and the Function Explorer

```
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
```

compute and func are aliases. Named assignments are evaluated in order and may use earlier scalar assignments. expr = gamma(5) is permitted for a single evaluation. The default arithmetic is binary128-class (Boost.Multiprecision cpp_bin_float_quad, 113 binary significant bits, about 34 decimal digits); precision = double is also available. This is a working-precision designation, not a guarantee of 34 accurate result digits.

### Function Explorer

```
compute
function = besselj(0,x)
range = [0,20]
points = 500
```

Samples include both endpoints. points may range from 2 to 20,000 (default 500). Undefined sample values are represented as unavailable samples; the desktop presents rows and a plot and supports CSV export. Explorer sampling is a numerical survey and cannot replace an analytic proof.

**VERSION 3 ADDITIONS**

# 83 Compute function catalogue and isolation

Compute supports the familiar elementary functions as well as cot, sec, csc, inverse and hyperbolic variants, sign, floor, ceil, round, trunc, frac, factorial/fact, root, logbase/logb, erf, erfc, gamma and lgamma. Additional function families are listed below. Names and argument order matter.

Bessel: besselj(n,x), bessely(n,x), besseli(n,x), besselk(n,x).

Airy: airy_ai(x), airy_bi(x).

Orthogonal polynomials: legendre(n,x), assoc_legendre(n,m,x), hermite(n,x), laguerre(n,x), assoc_laguerre(n,a,x), chebyshev_t(n,x), chebyshev_u(n,x).

Gamma and related: beta(a,b), digamma(x), polygamma(n,x).

Other special: zeta(x), lambertw(x[,branch]), ei(x), si(x), ci(x).

Elliptic: elliptic_k(m), elliptic_e(m), elliptic_f(phi,m), elliptic_einc(phi,m), elliptic_pi(n,m), elliptic_pi(phi,n,m).

Real-domain conventions: lambertw(x) is principal real branch, and lambertw(x,-1) is the second real branch where defined; elliptic m is the parameter $k^2$; polygamma supports orders 0–8 (x>0 for positive orders); factorial accepts integer 0–2000 in binary128 mode. Input domain errors are explicit. Compute-specific special-function dispatch is isolated from the equation, quad, diff, limit, ODE, BVP, PDE and integral-equation parsers. Do not assume a Compute-only function is accepted in those other models.

**VERSION 3 ADDITIONS**

# 84 Compute number theory and input ceilings

```
compute
precision = binary128
n = 5929
a = totient(n)
b = primepi(1000000)
c = primefactors(n)
d = primefactor(n,3)
e = gcd(42,77)
```

totient(5929) = 4620; primepi(1000000) = 78498; primefactors(5929) = [7,7,11,11]. primefactors returns a list and cannot feed subsequent scalar arithmetic; primefactor(n,k) selects a 1-based scalar factor.

Available exact-integer utilities: totient(n), gcd(a,b), lcm(a,b), isprime(n), primepi(n), mobius(n), divisorcount(n), divisorsum(n), primefactors(n), and primefactor(n,k). All require integer inputs in their documented nonnegative/positive domains. The strict maximum for primepi(n) is 100,000,000 inclusive; primepi(100000000) = 5,761,455. The other number-theory inputs are bounded by 10^12 in this release. This deliberate performance ceiling is not the mathematical limit of prime counting or factorisation.

**VERSION 3 ADDITIONS**

# 85 Exact linear Diophantine equations

```
int x, y
7*x + 11*y = 100
```

This recognises one linear equality in exactly two declared integers and uses the exact Extended Euclidean Algorithm. One complete family is x = 8 + 11*t and y = 4 - 7*t (integer t). Equivalently $x \equiv 8 \pmod{11}$, with y = (100 - 7*x)/11. The congruence must be paired with the dependent-variable formula.

```
int x, y
3*x + 5*y = 100
bound x = [0,100]
bound y = [0,100]
```

This bounded example has exactly seven integer solutions. If gcd(|a|,|b|) does not divide c, a*x+b*y=c has no integer solution; otherwise integer parameter bounds determine a finite or infinite family. Signed and zero coefficients are supported, with exact counts for finite families and enumeration capped by display settings. Extremely large integer coefficients remain exact. The timing field may use microseconds or nanoseconds when an exact shortcut completes too quickly to display meaningful seconds.

**RELEASE-CANDIDATE TESTING**

# 86 Release readiness and regression checks

Treat the guide as a source-level feature specification until the Windows build has been updated. For the MSIX candidate, rebuild the engine CLI, DLL and Desktop GUI from the same commit and bundle matching files. Run targeted regression suites for differentiation, tabular research, interpolation, limits,

compute functions, number theory and linear Diophantine equations, then the existing algebraic, integer, quadrature, ODE, BVP, PDE and integral-equation suites.

Validate at least one passing case and one malformed/out-of-domain case per newly added mode in the installed MSIX, not only in a developer command shell. Check desktop tables, CSV output, plot views, reports, binary128 precision metadata, filenames, license controls, hashes and package installation/uninstallation. A passing source test alone does not verify the Windows package.

Before release, review any previously documented open defect against the final rebuilt Windows executable; update its status only when reproduced and retested. Preserve the exact model, expected reference, output status, platform and version in the regression record.

**VERSION 3.0.1 NUMERICAL TOOLS**

# 87 Split integrals and investigate singularities

### Use explicit ranges or declare split points

Run the examples separately. In the first, quad evaluates the same expression on two intervals and combines the contributions. In the second, the solver splits [0, 1] at the declared interior 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]
```

The integrand is undefined at 0.2 but the two improper integrals converge. The reference value is $6/\sqrt{5} \approx 2.683281573$. Check Accepted integral and Target met before using the result. Splitting at a point still requires the relevant one-sided integrals to converge.

### Inspect an expression before choosing a split

```
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)]
```

Run with the comment in place to investigate the interval. Remove // to supply the two candidate points explicitly. The expression has finite corners at $\pm 1/\sqrt{2}$. At $\pm 1$ the formula is undefined, and the limit from inside the interval is 2 at each endpoint. Use the reported evidence to decide where to inspect or split the model.

### Read unresolved or ambiguous diagnostics

For ODEs, BVPs, PDEs and integral equations, bounded numerical investigation can provide an additional validation step. If the investigation remains unresolved or ambiguous, retain that status and inspect the model, supplied data and relevant diagnostics before relying on the result.

**VERSION 3.0.1 DESKTOP TOOLS**

# 88 Equation Generator and Fun with Mathematics

### Create a reproducible nonlinear system

Open Tools → Equation Generator. Select 1–20 variables, the function families and a complexity level. Add bounds if you want a restricted model, then generate the system. Record the seed and settings to recreate it later.

For a classroom exercise, copy the generated equations and any bounds into the editor, solve the model, then reveal the constructed solution and compare it with the returned roots. The constructed solution provides a reference point. The system may admit other roots.

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

### Choose an activity

Find Your Number in Pi. Enter a sequence of digits to look for it in π.

The King’s Puzzle. Find the smallest starting number that satisfies the sequence of divisibility conditions.

Number Puzzle. Arrange number tiles to satisfy the displayed arithmetic rules.

# 89 Run your first workflow in Desktop

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.

Paste this example into the model editor or save it as a text file and open it in Desktop.

This example gives quadrature a cap and runs differentiation using the overall workflow budget.

```
workflow
budget = 2
quad area within budget
    expr = x^2
    var = x
    range = [0, 3]
end
diff slope within 2
    expr = x^3
    at x = 2
    wrt = x
end
if area.accepted
    report area.value
else
    report area.status
end
report slope.result.requested_derivative
```

Expected reports are 9 and 12, plus one notice about the ignored differentiation cap.

### Available services

| Service | Header example | Where to find the result |
|---|---|---|
| Equation solving | solve roots within 2 | roots.roots, roots.status |
| Definite quadrature | quad area within 2 | area.value, area.estimated_absolute_error |
| Compute Functions | compute calc | calc.result.results for calculations |
| Differentiation | diff slope | slope.result.requested_derivative when requested |
| Limit | limit near_zero | near_zero.status, near_zero.value for a finite value |
| Interpolation | interpolate sample | sample.result.results |
| ODE initial-value problem | ode decay within 2 | decay.result.trajectory |
| Boundary-value problem | bvp curve within 2 | curve.result.trajectory |
| PDE | pde square within 2 | square.result.trajectory |
| Eigenvalue problem | eigen modes_found within 2 | modes_found.result.eigenpairs |
| Integral equation | integral fredholm within 2 | fredholm.result.trajectory |
| Singularity analysis | singularities poles within 2 | poles.result.points |

# 90 Write conditions loops and functions

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.

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.

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.

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

Expected result is approximately 2.928968253968254.

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.

# 91 Sweep parameters and combine services

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
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).

### Pass an algebraic root to quadrature

Read a root field in the workflow and use @root.x to pass its numeric value to a job. Inside a service block, write that service's model syntax. Use ordinary workflow names outside service blocks. A reusable model sees only its declared parameters for substitution.

```
workflow
solve scale
    x = 2
end
for root in scale.roots
    quad area
        expr = @root.x*t
        var = t
        range = [0,1]
    end
    if area.accepted
        report area.value
    else
        report area.status
    end
end
```

Expected area is 1. Open the two jobs in Run Report to inspect the solve and integration independently.

# 92 Set budgets and interpret workflow results

The eight headers shown with within enforce an optional per-job budget. In the workflow timing update, compute, diff, limit, and interpolate accept a valid within clause, run normally and record an informational notice that the cap was ignored. The same rule applies to reusable models.

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.

### Read the results

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.

quad requires an ordinary integral. A pole may make it divergent even when a Cauchy principal value exists. Run singularities to investigate the interval and inspect the job's reason when a result is unaccepted.

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.

### Run the same file from the command line

From Command Prompt, run the following with the executable on your PATH or from its installation directory.

```
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.