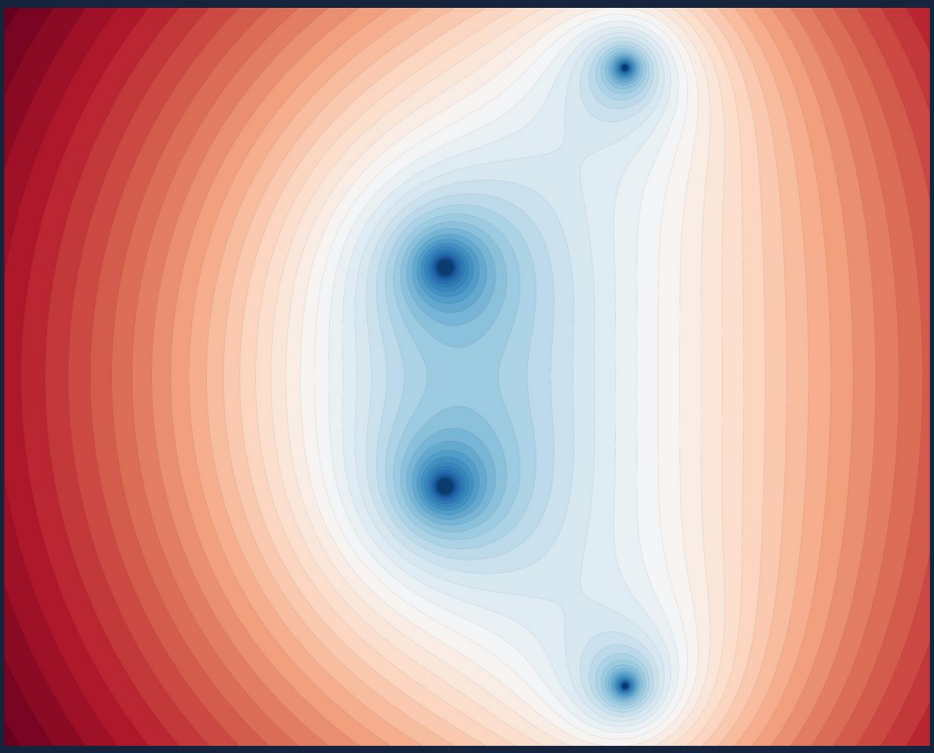

The Runge-Kutta stability region: a stiff spectrum lies outside it.

Solving Differential Equations in Julia I

Mass Matrices, DAEs, and the Stiff Solver Stack

Petra Vuković

ODIN PRESS

Solving Differential Equations in Julia I

Paperback version, First edition: 2026.
ISBN: forthcoming (Bowker allocation pending)
Trim Size: 6×9 inch. Paper color: White.

Published by Odin Press.
`https://odinpress.org`
This book was typeset in LaTeX by Odin Press from a manuscript submitted by the author.

Printed and Bound in the United States of America.

9 8 7 6 5 4 3 2 1 0

Preface

There are two complaints about a differential equation solver that are constantly mistaken for one another, and the whole of this book turns on keeping them apart. The first is that a tool is slow: it computes the right answer at a price one would rather not pay, and the remedy is an argument about hardware. The second is that a tool cannot state the problem at all — the model one has does not fit through the interface, at any price. The first is an engineering inconvenience. The second is a structural fact, and it is the only one of the two that settles an argument. Most of the writing in this field blurs the distinction, and a reader can smell it. I have tried to earn the claim of impossibility everywhere I make it, and to refuse it wherever only slowness is true.

The occasion for the distinction is the reference initial-value interface of scientific Python, which is genuinely excellent and which I recommend without hedging on the regime where it is enough — the small, non-stiff, first-order system given as an explicit derivative. But a discretised evolution equation does not arrive that way. It arrives as a mass matrix multiplying the derivative, with that matrix fixed by the physics — a capacitance, an inertia, the Gram matrix of a basis — and the reference interface has no argument position for it. That is not slowness. It is an empty cell in a capability matrix, and it recurs: a singular mass matrix that is really a constraint, a differential-algebraic index that must be reduced before any solver can touch the system, a second-order form that flattening doubles, a boundary condition that is a functional of the whole solution. The gap between slowness and impossibility is the subject, and the Julia SciML stack is on the page because it expresses what the reference interface cannot.

This is the first of two volumes, and the split between them is not arbitrary: the empty cells fall along two axes. This volume takes the axis of structure — the deterministic system whose form the standard interface cannot state: the mass matrix that reweights a derivative and, when

singular, becomes a constraint; the differential-algebraic index that must be reduced before any solver can touch it; the symbolic-numeric layer that composes a model acausally and reduces that index automatically; the split that pays for a stiff term alone; the second-order form flattening needlessly doubles; the boundary value problem where nothing marches. The second volume takes the other axis — not the equation's structure but its dynamics, and the workflows built over a solver: stochastic, delay, and hybrid systems; continuation and bifurcation; performance, large systems, and GPU ensembles; the differentiation of the solver itself and the universal differential equation; and the verification discipline that governs every number in both. Together they make the complete case — that across structure and across stochasticity, memory, discontinuity, scale, and differentiability, the Julia stack expresses problems the reference interface cannot, not more slowly but at all. The reader who works both learns to look at an equation, decide whether the standard tool is slow or simply cannot, and verify the solver chosen for it. The rule of evidence set here is the rule the second volume runs on, and its first chapter restates this framework before extending it.

I have held every comparison to one rule of evidence. The competing tool is given its best available configuration before a single conclusion is drawn — the right solver, an analytical Jacobian, exposed sparsity, a tuned tolerance — because a rigged benchmark is worth less than no benchmark, and the concessions are what buy the reader's trust where the verdict is close. No timing appears without the tolerance it was run at. Where a claim is quantitative there is a work-precision diagram carrying it, because faster is not a claim; it is a diagram, or it is nothing. And no solution is reported that I have not verified against a manufactured one, watched converge at its observed order, and checked for a conserved quantity that should not drift. A solution I have not verified is a plot, not a result.

The argument climbs a ladder, each rung removing one more thing the reference interface can express. Chapter 1 measures the wall: an explicit integrator whose step is set by stability, not accuracy, and whose cost grows with the square of the mesh. Chapter 2 explains it with the durable mathematics of stiffness — the stability function, A- and L-stability, order reduction — and retires the stiffness ratio. Chapter 3 draws the honest landscape, conceding what Python, the SUNDIALS wrappers, and the differentiable JAX suite each do well, so every later claim is made against the strongest opponent. Chapters 4 and 5 take the mass matrix and the differential-algebraic index as first-class problems, up to the index-three pendulum in Cartesian coordinates. Chapter 6 is where the claim is strongest and

narrowest: a symbolic-numeric layer with no Python counterpart, and an honest concession to Modelica. Chapters 7 and 8 split a stiff problem to pay for its stiff part alone, and integrate a second-order system without flattening it. Chapter 9 changes the problem class entirely, to trajectories pinned at both ends, and closes by conceding a failure of the native stack itself.

Every numerical result was produced by the code printed in Appendix A and is reproducible from a pinned environment; where a computed quantity contradicted the claim I intended, I changed the claim. I report each method with its boundary and each timing with its tolerance, and I have named the places where the method I recommend does not merely slow down but stops. That boundary is not a gap in the argument. It is part of it.

— Petra Vuković, 2026

Contents

Preface vii

List of Figures xv

List of Tables xxi

1 The Million-Step Wall 1
- 1.1 The system as it actually arrives . . . 2
- 1.2 Stiffness: the step that stability dictates . . . 8
- 1.3 The interface that cannot state the problem . . . 15
- 1.4 Sparsity is the whole game . . . 17
- 1.5 The same problem, solved . . . 21
- 1.6 What this book is . . . 28

2 What Stiffness Actually Is 31
- 2.1 The Jacobian spectrum and the linear test equation . . . 32
- 2.2 The stability function and its region . . . 34
- 2.3 A-stability and what it costs . . . 39
- 2.4 L-stability and the stiffly damped modes . . . 42
- 2.5 Stage order, weak stage order, and order reduction . . . 47
- 2.6 What stiffness is, and what it is not . . . 51

3 The Honest Landscape 57
- 3.1 The reference interface and what it is for . . . 58
- 3.2 Wrapping the industrial solvers . . . 60
- 3.3 The modern challenger . . . 64
- 3.4 Where each ceiling sits . . . 68
- 3.5 The rule of evidence . . . 72
- 3.6 Where the standard tool is enough . . . 76

4 Mass Matrices 79

4.1 Where mass matrices come from . . . 80
4.2 The Rosenbrock-W stage with a mass matrix . . . 83
4.3 Order reduction and why Rodas5P exists . . . 88
4.4 When M is singular . . . 91
4.5 Sparsity and the stage matrix in practice . . . 95
4.6 Choosing a mass-matrix solver . . . 97

5 Differential-Algebraic Systems 101
5.1 What the index measures . . . 102
5.2 Structural analysis of the incidence pattern . . . 105
5.3 Index reduction and dummy derivatives . . . 109
5.4 Why high index defeats a solver . . . 112
5.5 The residual form . . . 116
5.6 Reduce, or solve the residual . . . 119

6 The Symbolic-Numeric Layer: ModelingToolkit 123
6.1 The model as a graph, not a function . . . 124
6.2 Acausal composition from physical components . . . 127
6.3 Structural simplification and tearing . . . 130
6.4 Automatic index reduction . . . 135
6.5 Generated Jacobians and sparsity . . . 138
6.6 What the other ecosystems have instead . . . 144

7 Split Systems: IMEX and Additive Runge-Kutta 149
7.1 The split: what the stiff half is, and who decides . . . 150
7.2 Additive Runge-Kutta: two tableaux, one step . . . 152
7.3 Order conditions and the coupling conditions . . . 153
7.4 Stability of the coupled scheme . . . 157
7.5 When splitting beats full implicitness . . . 160
7.6 What an additive method requires, and who else provides it 164

8 Second-Order Systems: Nystrom Methods and the Cost of Flattening 167
8.1 Newton Wrote It Second Order: The Form the Interface Cannot Take . . . 168
8.2 Flattening: The Doubled State, and the Half of It That Is an Identity . . . 169
8.3 Nystrom Methods: Stages in the Position, Not in the Velocity . . . 172

8.4 Order Conditions, and Where the Stage Count Actually Falls . . . 173
8.5 Matched Accuracy, Not Matched Tolerance: The Only Honest Comparison . . . 176
8.6 Separable and Partitioned Systems, and the Boundary of the Claim . . . 181

9 Boundary Value Problems: Pinned at Both Ends 185
9.1 Nothing Marches: The Boundary Value Problem as One Algebraic System . . . 186
9.2 Shooting, and the Exponential Sensitivity That Defeats It 187
9.3 Collocation, MIRK, and the Mesh That Refines Itself . . 190
9.4 The Boundary Condition Is a Functional, Not Two Endpoint Values . . . 193
9.5 Eigenvalue Problems, Where the Normalisation Constraint Is Not Optional . . . 196
9.6 What solve_bvp Does Well, and Exactly Where It Stops 199

A Source Code 203
A.1 Chapter 1 — The Million-Step Wall . . . 203
A.2 Chapter 2 — What Stiffness Actually Is . . . 217
A.3 Chapter 3 — The Honest Landscape . . . 227
A.4 Chapter 4 — Mass Matrices . . . 254
A.5 Chapter 5 — Differential-Algebraic Systems . . . 274
A.6 Chapter 6 — The Symbolic-Numeric Layer: Modeling-Toolkit . . . 289
A.7 Chapter 7 — Split Systems: IMEX and Additive Runge-Kutta . . . 309
A.8 Chapter 8 — Second-Order Systems: Nystrom Methods and the Cost of Flattening . . . 327
A.9 Chapter 9 — Boundary Value Problems: Pinned at Both Ends . . . 340

References 355

Index 367

About the Author 371

Chapter 1

The Million-Step Wall

A spatial discretisation of an evolution equation does not hand you a derivative you can integrate. It hands you a linear relation between a derivative and a state, $Mu'(t) = f(t, u(t))$, in which the matrix M on the left is as much a part of the problem as the operator on the right. This chapter takes one such system — a one-dimensional diffusion problem discretised by piecewise-linear finite elements — and asks the only two questions that matter about a solver: can it state the problem, and, if it can, what sets the cost of the answer.

The two questions are not the same, and the distance between them is the subject of the whole book. A tool can be slow: it computes the right answer at a price you would rather not pay, and the remedy is an argument about hardware. A tool can also be unable to state the problem at all, at any price, and no amount of tuning reaches that. The reference initial-value interface in scientific Python is in the second position with respect to the mass matrix, not the first, and the demonstration that it is — measured, not asserted — is the chapter's deliverable.

The argument runs in six moves. Section 1.1 assembles the system and shows why M is not the identity and cannot be cleared away for free. Section 1.2 reads the stiffness off the spectrum and converts it into a step count. Section 1.3 shows that the reference interface cannot express M at all. Section 1.4 measures what the forced rearrangement costs. Section 1.5 states the same problem to a solver that accepts M and runs the fair comparison. Section 1.6 says what the remaining chapters are.

1.1 The system as it actually arrives

Let $u(t) \in \mathbb{R}^n$ collect the nodal values of a field that evolves in time, obtained by discretising a partial differential equation in space while leaving time continuous. The result is a finite-dimensional system in which a matrix multiplies the time derivative.

Definition 1.1 (Semi-discrete system)**.** A *semi-discretisation* of an evolution problem is a finite-dimensional system

$$M\,u'(t) = f(t, u(t)), \qquad u(t) \in \mathbb{R}^n, \tag{1.1}$$

where $M \in \mathbb{R}^{n \times n}$ is the *mass matrix* assembled from the spatial discretisation and f collects the discrete spatial operator together with the load and boundary data. The system is an ordinary differential equation in time; the matrix M is part of its statement, not an artefact of it.

The name of M is not decoration. It is the discrete image of the L^2 inner product on the basis that represents the field, and its structure is decided entirely by that basis.

Definition 1.2 (Mass matrix)**.** The *mass matrix* of a Galerkin discretisation with basis functions $\{\phi_i\}_{i=1}^{n}$ is the Gram matrix $M_{ij} = \langle \phi_i, \phi_j \rangle$. It is the identity only when the basis is orthonormal. A *lumped mass matrix* replaces M by a diagonal matrix, by construction; a *consistent mass matrix* keeps the true Gram matrix and is sparse, symmetric positive definite, and *not* diagonal.

Everything that follows turns on the word *not* in that last clause. A diagonal M is inverted at the cost of n divisions and changes nothing structural; the whole difficulty is that a consistent M is sparse but its inverse is not.

Proposition 1.3 (The inverse of a consistent mass matrix is dense)**.** Let M be an irreducible tridiagonal symmetric positive-definite matrix. Then every entry of M^{-1} is nonzero. Consequently, for the semi-discrete system the explicit form $u' = M^{-1} f(u)$ has a dense right-hand-side operator even though M itself is sparse.

Proof. An invertible tridiagonal matrix has a *semiseparable* inverse: there exist vectors $p, q \in \mathbb{R}^n$ such that $(M^{-1})_{ij} = p_i q_j$ for $i \le j$, and $(M^{-1})_{ij} = p_j q_i$ for $i > j$, by symmetry. The entries of p and q are ratios of leading and

trailing principal minors of M. For an irreducible tridiagonal matrix every principal minor in these sequences is nonzero — for a symmetric positive-definite M they are in fact strictly positive — so no p_i or q_j vanishes. Hence $(M^{-1})_{ij} = p_{\min(i,j)} q_{\max(i,j)} \neq 0$ for all i, j, and M^{-1} is full. Multiplying the full matrix M^{-1} by any matrix with no zero column, in particular by a nonsingular sparse J_f, yields a matrix with no zero entries. The operator $M^{-1} f$ therefore couples every degree of freedom to every other, whatever the sparsity of f. □

The consistent mass matrix is sparse and its inverse is full: this is the first structural fact of the book, and the one from which the chapter's cost follows.

Corollary 1.4 (When the difficulty disappears)**.** If the basis is orthonormal, or if the mass matrix is lumped to a diagonal $\widehat{M}$, then M^{-1} (respectively $\widehat{M}^{-1}$) is itself diagonal, and $u' = M^{-1} f(u)$ preserves the sparsity of f. The obstruction of this chapter is precisely the price of a *consistent* mass matrix and applies to no other.

Proof. If the basis is orthonormal then $M = I$ and $M^{-1} = I$. If the mass matrix is lumped to a diagonal $\widehat{M}$, then $\widehat{M}^{-1}$ is the diagonal matrix of reciprocal diagonal entries; a diagonal matrix multiplied into a sparse matrix scales its rows and preserves its sparsity pattern exactly. In either case the operator $M^{-1} f$ inherits the sparsity of f, and the dense fill-in of the preceding proposition does not arise. The obstruction is therefore confined to the consistent, non-diagonal case. □

This corollary is a promise about scope. A reader whose mass matrix is diagonal by construction is not the reader this chapter is addressed to, and it is worth saying so plainly rather than letting the difficulty be assumed universal.

We fix a concrete system and keep it for the rest of the chapter.

Example 1.5 (The running diffusion system)**.** Discretise the heat equation $u_t = u_{xx}$ on $(0, 1)$ with homogeneous Dirichlet data by continuous piecewise-linear (P_1) finite elements on a uniform mesh of n interior nodes, spacing $h = 1/(n+1)$. Assembly gives the tridiagonal stiffness matrix K with the stencil $h^{-1}(-1,\ 2,\ -1)$ and the tridiagonal consistent mass matrix M with the stencil $\frac{h}{6}(1,\ 4,\ 1)$, and the system

$$M\,u'(t) = -K\,u(t), \qquad u(0) = u_0. \tag{1.2}$$

Chapter 3

The Honest Landscape

A book that argues a tool cannot do something owes the reader a fair account of what that tool can do. This chapter is that account. It maps, capability by capability, what Python can and cannot do for differential equations — the reference interface of the standard scientific stack, the SUNDIALS wrappers that reach past it, and the JAX-based challenger that is the serious modern rival — and it gives each of them its best available configuration before drawing a single conclusion. The reward for that discipline is that every later claim of structural impossibility in this book is made against the strongest opponent available, not the most convenient one.

The chapter concedes before it claims. Section 3.1 measures the regime in which the reference interface is fully adequate and states plainly that on that regime this book recommends no change. Section 3.2 credits the SUNDIALS wrappers with the residual form, which expresses Chapter 1's mass-matrix system exactly, so that the obstruction of that chapter is correctly located at the reference interface rather than at Python. Section 3.3 gives the differentiable JAX suite its full due on the one axis where it is strongest. Section 3.4 assembles the capability matrix that is the map of the rest of the book. Section 3.5 states the rule of evidence that governs every measurement to come, and demonstrates a benchmark reversing under a careless protocol. Section 3.6 tells the reader, in both directions, when to change tools and when not to.

The result is a boundary drawn honestly on both sides. Most differential equations solved in practice fall on the near side, where the standard tool is enough. Everything in the remaining chapters lives on the far side, and

the credibility of those chapters is bought here, by conceding the near side without hedging.

3.1 The reference interface and what it is for

The entry point most scientists reach for takes an explicit derivative and a handful of scalar settings, and for the problem it was designed for it is a competent, well-tested tool.

Definition 3.1 (The reference interface)**.** The *reference interface* is the initial-value entry point of the standard Python scientific stack: it takes an explicit derivative $f(t, y)$, an integration interval, an initial state, a method name, and tolerances, and returns the computed trajectory.

Proposition 3.2 (Six methods, sufficient for the common case)**.** The reference interface offers six methods — RK45, RK23 and DOP853 for non-stiff problems, and Radau, BDF and LSODA for stiff ones — and for the overwhelming majority of small, non-stiff, explicitly stated systems this is entirely sufficient [8].

Proof. The interface's `method` argument enumerates exactly these six integrators, three tuned for non-stiff problems and three for stiff ones [8]. A small non-stiff system stated as an explicit derivative is precisely the input each of the non-stiff three was designed to integrate, and the measurements of this section confirm that they reach every tolerance asked of them on such a system. The claim is not that the interface is universal but that its menu covers the common case, and that is what the enumeration and the measurement jointly establish. □

Adequacy is the word this section will earn by measurement, and it deserves a precise definition.

Definition 3.3 (The adequacy regime)**.** The *adequacy regime* of a tool is the class of problems on which it returns the right answer at acceptable cost. Naming a tool's adequacy regime honestly is a precondition of criticising it outside that regime.

On the non-stiff class the interface is adequate, and the honest way to establish that is to concede the cost gap it does carry.

Proposition 3.4 (The interface is adequate, not competitive, on the non-stiff class)**.** For a non-stiff system of modest size stated with an explicit

derivative, the reference interface attains every tolerance it is asked for, down to the limits of double precision. It is not within a small constant factor of a compiled implementation: driving the integrator from Python costs one to two orders of magnitude in wall time, and the gap does not close as the system grows, because it is the cost of the interpreted right-hand side, not of the algorithm. It is conceded anyway, because on this class the absolute cost is milliseconds, and the tool is *adequate*.

Proof. Table 3.1 measures the non-stiff head-to-head. The reference interface reaches every requested tolerance: RK45 achieves 3.09×10^{-3} at a loose tolerance and 2.00×10^{-12} at a tight one, DOP853 9.82×10^{-5} down to 8.49×10^{-13}. At a matched tolerance of 10^{-10} the interface's DOP853 costs 5.66×10^{-3} s against the compiled suite's 1.07×10^{-4} s — a factor of 53, which does not shrink at larger n because it is the per-evaluation cost of an interpreted derivative, incurred once per stage regardless of the algorithm. The factor is real and is conceded; the absolute time is milliseconds, so on this class it buys the reader nothing, and adequacy — the right answer at acceptable cost — holds. Competitiveness is not claimed, and would be false. □

Table 3.1: The non-stiff case, conceded: achieved error and wall time, small non-stiff system

method	**err at tol** 10^{-4}	**time (s)**	**err at tol** 10^{-12}	**time (s)**
SciPy RK45	3.09×10^{-3}	1.17×10^{-3}	2.00×10^{-12}	3.17×10^{-2}
SciPy DOP853	9.82×10^{-5}	1.50×10^{-3}	8.49×10^{-13}	9.02×10^{-3}
Julia Tsit5	4.39×10^{-4}	8.73×10^{-5}	2.23×10^{-13}	3.65×10^{-4}
Julia Vern9	4.33×10^{-6}	8.85×10^{-5}	2.54×10^{-13}	1.21×10^{-4}

Corollary 3.5 (The gap is the interpreter, not the algorithm)**.** The one-to-two-order wall-time gap on the non-stiff class is a property of the interpreted right-hand side, not of the integration algorithm, and it therefore does not close as the system size grows.

Proof. An adaptive Runge–Kutta step evaluates the right-hand side a fixed number of times per step, independent of the implementation language; the algorithm and its step sequence are the same. What differs is the cost of one evaluation: an interpreted derivative pays per-call dispatch and boxing

There are two complaints about a solver that are constantly mistaken for one another: that it is slow, and that it cannot state your problem at all. The first is an argument about hardware. The second is a structural fact, and only the second is worth a book.

Most differential equations are solved perfectly well by the standard scientific-Python stack, and this book says so plainly. But a discretised evolution equation arrives as a mass matrix multiplying a derivative — fixed by the physics, not chosen for convenience — and the reference interface has no place to put it.

This is the first of two volumes. Volume I takes the axis of structure: singular mass matrices that are really constraints, high-index differential-algebraic systems, the symbolic-numeric layer that reduces that index automatically, split stiffness, second-order mechanics, and boundary value problems pinned at both ends. Volume II takes the other axis — stochastic, delay, and hybrid dynamics, sensitivities, universal differential equations, and scale. Together they map where the Julia SciML stack states what Python cannot.

Every comparison gives the rival its best configuration. Every timing carries its tolerance. Every claim rests on a work-precision diagram, and every solution is verified against a manufactured one. Slow is an inconvenience; cannot is a theorem. These books keep them apart.

ISBN 13 978-258-77-1942-1
Paperback
9 782587 719421