Numerical Methods for Geosciences

Instructors: Harsha S. Bhat and Xavier Perrot

Fridays 10:30–12:30, Sept 2026–Jan 2027 (Room E509)

This document is a compilation of handwritten notes, PDF's etc prepared by myself, Harsha S. Bhat, over the years. Claude was used to transcribe, collate and summarize some aspects of this document. Thus it is quite likely that errors have crept in and I am in the process of cleaning it up. I am solely responsible for these errors.

Contents

1 Overview and Roadmap

These notes cover the theoretical content of Session 1 of a twelve-session sequence whose goal is to build, from scratch, a working quasidynamic earthquake-cycle simulator on a one-dimensional fault. Every later session reuses the architecture introduced today: *assemble the governing physics, solve a nonlinear algebraic equation for the instantaneous state of the system, then step that state forward in time.* Today we build the two numerical tools that architecture depends on — a root-finder and a time integrator — and we build them in the context of the simplest possible earthquake-cycle model: a single fault patch loaded through a spring.

The session sequence, and where today sits within it, is as follows.

1. **Foundations** (today): root-finding via Newton–Raphson; explicit time integration via Euler's method and the classical fourth-order Runge–Kutta method.
2. **Finite differences (FD)**: the fault-perpendicular thermal pressurization diffusion problem, discretized by finite differences.
3. **Finite volumes (FV)**: the same diffusion problem in flux-conservative form, needed once the diffusivity varies spatially.
4. **Boundary element method (BEM)**: the elastic interaction matrix that couples slip on different patches of the fault.
5. **Coupling**: mechanics, thermal pressurization, and dilatancy assembled into one feedback loop, with adaptive time-stepping.
6. **Synthesis**: the full multi-patch assembly, benchmarked against analytic limits.

Two numerical themes recur across all six sessions and are worth stating up front, because they explain *why* we spend an entire session on what might look like undergraduate numerical analysis.

- **Nonlinear algebraic equations appear at every time step.** The friction law relates stress to slip velocity through a transcendental relationship; later, the coupled thermo-mechanical system will require an implicit relationship between effective stress and pore pressure. Newton's method, developed carefully today, is the tool we reach for every time this happens.
- **Time integration must be both accurate and, eventually, adaptive.** Euler's method is cheap but low-order; Runge–Kutta methods trade extra function evaluations per step for a much faster reduction in error as the step size shrinks. Once thermal pressurization and dilatancy are coupled into the system (Session 5), the dynamics become stiff and adaptive step-size control — built from an *embedded* pair of Runge–Kutta methods — becomes essential. We plant that idea today so it is not a surprise later.

2 The Physical Problem: A Single Fault Patch

2.1 Elastic loading

Consider the simplest conceivable model of a seismogenic fault: a single patch of finite area, embedded in an elastic medium, loaded remotely at a constant plate rate. We idealize the elastic interaction between the patch and its surroundings as a linear spring of stiffness k, loaded at the far-field rate V_{load}. If V denotes the instantaneous slip velocity of the patch, the shear stress τ acting on the patch evolves according to the elastic loading equation

$$\frac{\mathrm{d}\tau}{\mathrm{d}t} = k\,(V_{\mathrm{load}} - V)\,. \tag{1}$$

When the patch slips exactly at the loading rate ($V = V_{\mathrm{load}}$), stress is constant; when it slips more slowly than loaded ($V < V_{\mathrm{load}}$), stress accumulates; when it slips faster ($V > V_{\mathrm{load}}$), stress is relieved. This is, in miniature, the loading/unloading cycle of an earthquake: interseismic loading corresponds to $V \ll V_{\mathrm{load}}$, and coseismic slip corresponds to $V \gg V_{\mathrm{load}}$.

Equation (1) alone does not close the system: it relates $\mathrm{d}\tau/\mathrm{d}t$ to V, but says nothing about what sets V at a given τ. That relationship is supplied by friction.

2.2 Rate-and-state friction

Laboratory rock-friction experiments show that the frictional resistance of a sliding surface depends not just on the instantaneous slip velocity, but also on the recent sliding history of the contact, encoded in a state variable θ with dimensions of time. The rate-and-state friction law (Dieterich, 1979; Ruina, 1983) expresses the shear stress supported by the fault as

$$\tau = \sigma_n \left[\mu^* + a \, \ln\!\left(\frac{V}{V^*}\right) + b \, \ln\!\left(\frac{\theta V^*}{D_c}\right) \right], \tag{2}$$

where:

- σ_n is the effective normal stress clamping the fault;
- μ^* is a reference friction coefficient, attained when $V = V^*$ and $\theta = D_c/V^*$;
- V^* is a reference slip velocity;
- a is the *direct effect* coefficient: an instantaneous increase in V raises τ through the $a \ln(V/V^*)$ term, before the state variable has time to respond;
- b is the *evolution effect* coefficient, coupling friction to the state variable θ;
- D_c is the critical slip distance over which the state variable relaxes to a new steady value after a change in sliding conditions.

The sign of $a-b$ determines whether steady sliding is velocity-strengthening ($a > b$, stable) or velocity-weakening ($a < b$, potentially unstable, i.e. capable of hosting stick-slip and hence earthquake nucleation). Velocity-weakening patches are the ones that can produce the fast, runaway slip we associate with dynamic rupture.

2.3 State evolution

The state variable θ evolves according to one of several proposed laws; we adopt the *aging law*,

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = 1 - \frac{V\theta}{D_c}. \tag{3}$$

At steady state ($\mathrm{d}\theta/\mathrm{d}t = 0$) this gives $\theta_{ss} = D_c/V$: the state variable is inversely proportional to slip rate, consistent with the interpretation of θ as a measure of the average age of asperity contacts on the sliding surface — contacts that slide faster renew (and thus age) more quickly. Away from steady state, Eq. (3) describes state relaxing toward D_c/V over a slip distance of order D_c, which is why D_c is called the *critical slip distance*: it is simultaneously the memory length scale of the friction law and (as we will see in Session 5) the length scale that sets the porosity relaxation of dilatant gouge.

Remark 2.1 *Notice the structural similarity between Eq. (3) and the porosity relaxation equation you will meet in Session 5, $\mathrm{d}\phi/\mathrm{d}t = -(\phi - \phi_{ss}(V))/D_c \cdot V$ in essence: both are first-order relaxation ODEs toward a velocity-dependent steady state, over the same length scale D_c. Recognizing this pattern now will make the coupling in Session 5 considerably less mysterious.*

2.4 Typical parameter values and orders of magnitude

A point worth internalizing before we ever write a line of code: slip velocity in an earthquake cycle spans an enormous dynamic range. Interseismic creep rates are of order $V \sim 10^{-11}$–10^{-9} m/s (comparable to plate rates of a few cm/yr); coseismic slip rates reach $V \sim 1$ m/s. That is roughly *nine to ten orders of magnitude* of variation in V over a single earthquake cycle. Typical laboratory-constrained values are $a, b \sim 0.005$–0.02, $D_c \sim$ 1–100 μm in the lab (and highly uncertain, likely much larger, at field scale), and σ_n of order tens to hundreds of MPa at seismogenic depths. We will return to this dynamic range in Section 4.5, because it directly motivates how we should — and should not — set up the Newton iteration for V.

2.5 An alternative state law, and a nondimensional form

The aging law, Eq. (3), is not the only evolution law used in the literature; the **slip law**,

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\frac{V\theta}{D_c}\ln\left(\frac{V\theta}{D_c}\right), \tag{4}$$

shares the same steady state $\theta_{ss} = D_c/V$ but predicts different transient behavior after a sudden velocity step (in particular, no evolution at all while $V = 0$, unlike the aging law's $\mathrm{d}\theta/\mathrm{d}t = 1$ at $V = 0$) and is often preferred for matching certain laboratory observations. We will use the aging law throughout this course for concreteness, but the numerical machinery built today — Newton's method for the algebraic friction relation, RK4 for the time evolution — applies unchanged to either choice: only the right-hand side of one ODE in the coupled system changes.

It is often convenient, and a standard habit worth forming early, to work in nondimensional variables rather than SI units directly. Scaling V by V^*, θ by D_c/V^*, and τ by σ_n collapses the friction law, Eq. (2), to

$$\tilde{\tau} = \mu^* + a\ln\tilde{V} + b\ln\tilde{\theta}, \qquad \frac{\mathrm{d}\tilde{\theta}}{\mathrm{d}\tilde{t}} = 1 - \tilde{V}\tilde{\theta}, \tag{5}$$

where tildes denote nondimensional quantities and $\tilde{t} = tV^*/D_c$. This removes several parameters from explicit view (they are absorbed into the choice of scales) and, more importantly for numerical work, tends to make the resulting quantities $\mathcal{O}(1)$ rather than spanning the 10^{-11}–1 range of dimensional V in SI units — reducing (though, as discussed in Section 4.5, not eliminating) the floating-point and step-scaling difficulties that motivate the log-velocity reparametrization.

3 The Nonlinear Algebraic Problem

3.1 The residual

At any instant, force balance on the fault patch requires that the friction law, Eq. (2), and the applied stress be consistent. Given a current value of the state variable θ and a current applied stress τ_{applied} (supplied, in the coupled system of Section 8, by the elastic loading equation), we must find the slip velocity V that satisfies

$$f(V) \;\equiv\; \tau_{\text{friction}}(V,\theta) - \tau_{\text{applied}} \;=\; \sigma_n\left[\mu^* + a\ln\left(\frac{V}{V^*}\right) + b\ln\left(\frac{\theta V^*}{D_c}\right)\right] - \tau_{\text{applied}} \;=\; 0. \tag{6}$$

This is the *residual equation* we must solve, at every time step of every simulation we build this semester, however many patches or however much additional physics is layered on top.

Remark 3.1 (θ is frozen during this solve) *It is worth being precise about the role θ plays in Eq. (6). At the moment we solve for V, θ is a* known, fixed number *— its current value, carried over from the state at this instant — not a second unknown being solved for simultaneously. The algebraic problem here is one equation in one unknown, V; θ only enters as a parameter of that equation. θ's own evolution, Eq. (3), is handled separately and later, by the time integrator (Section 8), which updates θ* using *the V this root-find just produced. Keeping this division of labor straight — Newton solves for V given θ; the time-stepper updates θ (and τ) given V — is essential once the two are wired together in Section 8.2.*

3.2 Why this is hard

Equation (6) is transcendental in V: V appears inside a logarithm, and there is no algebraic manipulation that isolates it. There is, in general, no closed-form solution. We must instead use an *iterative* method: start from a guess V_0, and refine it until the residual is acceptably close to zero. This is the subject of Section 4.

It is worth pausing to note two features of f that will matter for what follows.

- f is a smooth (in fact, real-analytic away from $V = 0$) function of V, so classical derivative-based root-finders apply directly.

- $f'(V) = \sigma_n a/V > 0$ for $V > 0$: the residual is strictly monotonically increasing in V (given $a > 0$), so a root, if it exists, is unique. This is reassuring — we are not at risk of Newton's method converging to the "wrong" root — but as we will see, uniqueness of the root says nothing about how easy it is to *find*.

4 Newton's Method

4.1 Derivation from Taylor's theorem

Suppose we have a smooth function f and a current guess V_n for where it crosses zero. Near V_n, f looks almost like a straight line — its tangent line at V_n. Newton's method's whole idea is: *since f itself is hard to solve exactly, just solve the easy, straight-line version instead, and use the answer as your next guess.*

Formally, Taylor's theorem tells us that close to V_n,

$$f(V_n + \Delta V) = f(V_n) + f'(V_n)\,\Delta V + \mathcal{O}\big(\Delta V^2\big)\,, \tag{7}$$

i.e. f really does look like a straight line (slope $f'(V_n)$) plus a small correction. Dropping that small correction and asking what step ΔV sends the straight-line part to zero,

$$f(V_n) + f'(V_n)\,\Delta V \approx 0 \quad \Longrightarrow \quad \Delta V = -\frac{f(V_n)}{f'(V_n)}, \tag{8}$$

gives the Newton update, $V_{n+1} = V_n + \Delta V$:

$$\boxed{V_{n+1} = V_n - \frac{f(V_n)}{f'(V_n)}}\,. \tag{9}$$

Geometrically, V_{n+1} is simply where the tangent line to f at V_n crosses the axis — Figure 1 shows this happening twice in a row, each tangent line landing closer to the true root.

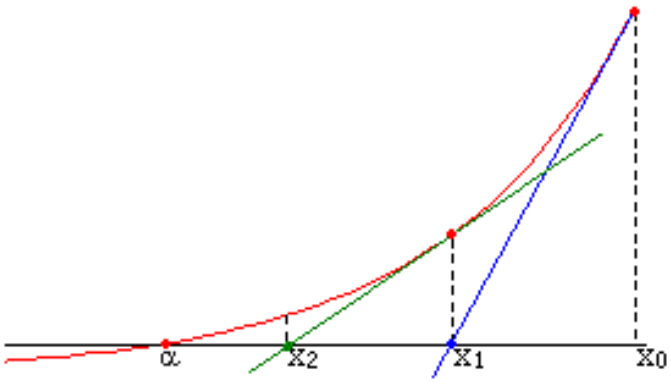

Figure 1: Newton's method: the tangent line at each guess x_n crosses zero at the next guess x_{n+1}, converging toward the root. Source: Wikipedia

Each iteration replaces the (generally curved, transcendental) function f by its local straight-line approximation and solves *that* exactly — which is why Newton's method is sometimes described as "linearize and solve, repeatedly."

For our residual, Eq. (6), the derivative needed at every iteration is

$$f'(V) = \frac{\mathrm{d}\tau_{\text{friction}}}{\mathrm{d}V} = \frac{\sigma_n a}{V}, \tag{10}$$

where the derivative is taken holding θ fixed, consistent with the remark in Section 3.1: throughout the entire Newton iteration, θ sits still at its current value, and only V moves from one iterate to the next. Because f' is available in closed form here, each Newton iteration costs one evaluation of f and one of f' — both $\mathcal{O}(1)$ operations.

4.2 Algorithm

Newton–Raphson for $f(V) = 0$

1. Choose an initial guess V_0 and a tolerance ε.
2. For $n = 0, 1, 2, \ldots$:
   (a) Evaluate $f(V_n)$ and $f'(V_n)$.
   (b) Update: $V_{n+1} = V_n - f(V_n)/f'(V_n)$.
   (c) If $|V_{n+1} - V_n| < \varepsilon$ (or $|f(V_{n+1})| < \varepsilon$), stop and accept V_{n+1}.
3. If a maximum iteration count is reached without convergence, flag failure (see Section 4.6).

4.3 Convergence theory

The appeal of Newton's method is not just that it works, but that — under the right conditions — it works *very* fast.

Here is the intuition, without the formalism. Look again at Figure 1: each tangent-line step landed noticeably closer to the root than the step before. That is not a coincidence of this particular curve — it is a general feature of using a *curved* function's tangent line as an approximation. The further you are from the root, the more the curve bends away from its tangent line, so a tangent-line step from far away is a poor approximation and can land almost anywhere. But once you are *close* to the root, the curve is almost indistinguishable from its own tangent line over that short remaining distance, so the tangent-line step is almost exactly right. The error left over is only due to how much the curve bends (its curvature, f'') over that short remaining distance — and that leftover error is proportional to the *square* of how far off you already were. Being twice as close to the root, roughly speaking, makes the next step four times closer still.

This is the entire content of what a textbook would call Newton's "quadratic convergence" guarantee: *once you are close enough to a root*, the number of correct decimal digits roughly *doubles* every iteration. If V_n is off by about 10^{-1}, expect V_{n+1} off by about 10^{-2}, V_{n+2} off by about 10^{-4}, and V_{n+3} off by about 10^{-8} — machine precision, in three or four more steps. This is why, in practice, well-posed Newton iterations for smooth friction laws converge to machine precision in 3–5 iterations.

The catch is the phrase *once you are close enough*: nothing above promises fast (or any) convergence starting from far away, especially wherever the curve is close to flat — there, the tangent line barely tilts, and the "obvious" step can rocket past the root instead of homing in on it. We come back to exactly this failure mode, with real numbers, in Problems 1–3 of the practice problems at the end of these notes.

4.4 A worked example

Take, for illustration, $\sigma_n = 100$ MPa, $a = 0.010$, $b = 0.015$, $V^* = 10^{-6}$ m/s, $\mu^* = 0.6$, $D_c = 10$ μm, and suppose at some instant $\theta = D_c/V^*$ (i.e. steady state at V^*) and $\tau_{\text{applied}} = 61$ MPa. The residual is

$$f(V) = 100\left[0.6 + 0.010\ln\!\left(\frac{V}{10^{-6}}\right) + 0.015\ln(1)\right] - 61 = 100\left[0.6 + 0.010\ln\!\left(\frac{V}{10^{-6}}\right)\right] - 61, \tag{11}$$

in MPa, with V in m/s. Starting from $V_0 = 10^{-6}$ m/s: $f(V_0) = 100(0.6) - 61 = -1$ MPa, $f'(V_0) = \sigma_n a/V_0 = 100 \times 0.010/10^{-6} = 10^6$ MPa·s/m. One Newton step gives

$$V_1 = 10^{-6} - \frac{-1}{10^6} = 10^{-6} + 10^{-6} = 2 \times 10^{-6}\ \text{m/s}. \tag{12}$$

Evaluating $f(V_1) = 100[0.6 + 0.010\ln 2] - 61 \approx 100(0.6069) - 61 \approx -0.31$ MPa: the residual has shrunk by roughly a factor of three, consistent with rapid convergence once close to a root of a well-scaled problem. A second iteration would reduce the residual by a comparable factor again, and the iteration converges to $V^* \approx 2.72 \times 10^{-6}$ m/s (i.e. $\ln(V^*/10^{-6}) = 1$) within a handful of steps.

4.5 Practical reparametrization: solving in $\ln V$

The worked example above was gentle: the initial guess was already within one order of magnitude of the root. In an earthquake-cycle simulation, this is *not* generally true — recall from Section 2.4 that V can range over nine or ten orders of magnitude over a single cycle, and the value of V from the previous time step (the natural initial guess for the current step) is a poor predictor of the new V precisely during the most interesting part of the cycle, the nucleation and coseismic phases, when V is changing fastest.

Solving directly for V has two related problems:

1. **Unphysical overshoot.** Nothing in the Newton update, Eq. (9), prevents V_{n+1} from being negative. Since $f'(V) = \sigma_n a/V \to \infty$ as $V \to 0^+$, a poorly scaled step near small V can easily overshoot past zero into $V < 0$, where the friction law itself is undefined (the logarithm has no real argument).
2. **Badly scaled steps across decades.** $f'(V) \propto 1/V$, so the "natural" Newton step size $f(V_n)/f'(V_n)$ scales like V_n itself. A step appropriate near $V \sim 1$ m/s is absurdly large near $V \sim 10^{-9}$ m/s, and vice versa.

The standard fix is to solve for the *logarithm* of velocity rather than velocity itself. Define

$$x \equiv \ln\left(\frac{V}{V^*}\right), \qquad V = V^* e^x. \tag{13}$$

Substituting into Eq. (6) gives a new residual, linear in x inside the friction term:

$$g(x) \equiv \sigma_n \left[\mu^* + a\,x + b \ln\left(\frac{\theta V^*}{D_c}\right)\right] - \tau_{\text{applied}} = 0, \tag{14}$$

with derivative $g'(x) = \sigma_n a$, a *constant*. Two benefits follow immediately.

- $V = V^* e^x$ is positive for *any* real x: reparametrizing in log-space makes unphysical negative velocities structurally impossible, regardless of how large a step Newton takes.
- Because g' no longer depends on V, Newton steps in x-space are naturally well-scaled across the entire dynamic range of V: a step of a given size in x corresponds to a *multiplicative* change in V, which is exactly the kind of change we expect as V ranges over decades.

In this particular residual g is in fact exactly linear in x once θ is held fixed, so a single Newton step in x would converge exactly; the value of the reparametrization becomes more pronounced once this root-find is embedded inside the fully coupled system of Section 8 and, later, the thermal-pressurization-coupled system of Session 5, where θ (or the effective normal stress) also depends on V and the combined residual is no longer exactly linear in any single variable.

4.6 Failure modes and remedies

Even with the log-V reparametrization, Newton's method can fail to converge, and it is worth knowing the standard symptoms and remedies, because a simulation that silently returns a garbage V is far worse than one that visibly crashes.

Remark 4.1 (A derivative-free alternative) *Newton's method requires f' at every iteration; Eq. (10) gave us this in closed form, but for more complicated residuals later in the course (once thermal pressurization and dilatancy are coupled in) an analytic derivative may be inconvenient or unavailable. The* ***secant method*** *replaces $f'(V_n)$ with a finite-difference approximation built from the two most recent iterates,*

$$V_{n+1} = V_n - f(V_n)\,\frac{V_n - V_{n-1}}{f(V_n) - f(V_{n-1})}, \tag{15}$$

trading Newton's quadratic convergence rate for the slightly slower superlinear rate $|e_{n+1}| \sim C|e_n|^\varphi$ with $\varphi = (1+\sqrt{5})/2 \approx 1.618$ (the golden ratio), in exchange for needing only one new function evaluation per step rather than one function and *one derivative evaluation. We will use Newton's method throughout this course, since our residuals will always have accessible analytic derivatives, but it is worth knowing the secant method exists for the day you encounter a residual that does not.*

- **Poor initial guess.** If x_0 is far from the root, the local convergence theorem of Section 4.3 gives no guarantee; the iteration can wander before (if ever) entering the basin of quadratic convergence. *Remedy:* use the previous time step's converged V as the initial guess (warm starting), and fall back to a bisection or bracketing step if Newton diverges.
- **Oscillation or divergence.** Near an inflection point or where f' is small, Newton steps can overshoot repeatedly without converging. *Remedy:* step damping (take a fraction of the Newton step, e.g. $V_{n+1} = V_n + \alpha\Delta V$ with $\alpha < 1$, increasing α toward 1 as the residual shrinks) or a line-search that only accepts a step if it reduces $|f|$.
- **Stagnation at machine precision.** Once $|f(V_n)|$ is comparable to floating-point round-off in the evaluation of f, further iterations do not reduce the error and may even increase it. *Remedy:* stop when the residual stops decreasing meaningfully, not only when it is exactly below a fixed tolerance.

5 Ordinary Differential Equations and Explicit Euler

5.1 The initial value problem

Newton's method solves an algebraic equation at a single instant. But τ and θ in our spring-slider system evolve continuously in time, governed by the coupled ODEs, Eqs. (1) and (3). In general we consider an initial value problem (IVP)

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y), \qquad y(t_0) = y_0, \tag{16}$$

where y may be a scalar or, as in our case, a vector of state variables (τ and θ together). We seek a numerical approximation $y_n \approx y(t_n)$ at a sequence of discrete times $t_0 < t_1 < t_2 < \dots$, typically with a fixed step $h = t_{n+1} - t_n$ (though adaptive step sizing, previewed in Section 6.9, will vary h).

5.2 Euler's method, from a Taylor expansion

The simplest numerical integrator follows from truncating a Taylor expansion of y about t_n:

$$y(t_n + h) = y(t_n) + h\,y'(t_n) + \tfrac{1}{2}h^2 y''(t_n) + \mathcal{O}\big(h^3\big) = y(t_n) + h\,f(t_n, y(t_n)) + \mathcal{O}\big(h^2\big)\,. \tag{17}$$

Dropping the $\mathcal{O}\big(h^2\big)$ remainder and replacing the (unknown) exact solution $y(t_n)$ by its numerical approximation y_n defines the **explicit (forward) Euler method**:

$$\boxed{y_{n+1} = y_n + h\,f(t_n, y_n)}\,. \tag{18}$$

It is the numerical-integration analogue of Newton's method: a local, first-order (linear) approximation of the true dynamics, applied repeatedly. In plain terms: at every step, look at which way the solution is currently heading (its slope, $f(t_n, y_n)$), and walk in a straight line in that direction for a distance h — then look again, and repeat. Figure 2 shows what this looks like against the true, curving solution: because the true solution keeps curving away from whatever straight line you're currently walking along, each step drifts a little further off track than the last.

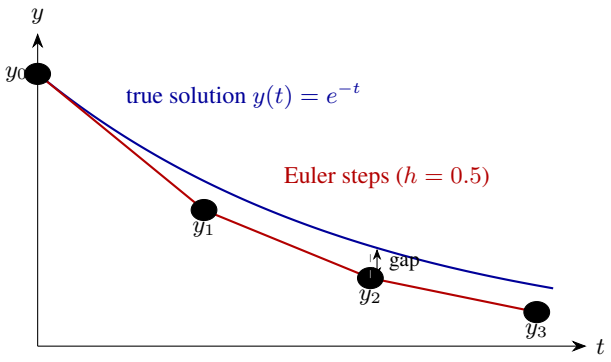

Figure 2: Euler's method walks in a straight line along the *current* slope for one full step (red segments), while the true solution (blue) keeps curving. Each step's straight-line assumption is only exactly right at its starting point, so the numerical path drifts progressively further from the truth — here for $y' = -y$, $y(0) = 1$, with a deliberately large $h = 0.5$ to make the drift visible.

5.3 Local and global truncation error

Two distinct notions of error matter for any time-stepping scheme, and it is worth being clear about the difference between them, even without belaboring a formal proof.

Local truncation error is the error made by a *single* step, assuming that step started from the exact solution. For Euler's method, this is exactly the piece of the Taylor expansion we truncated in Section 5.2:

$$\tau_n \equiv y(t_{n+1}) - [y(t_n) + h\, f(t_n, y(t_n))] = \tfrac{1}{2}h^2 y''(\xi_n) = \mathcal{O}\big(h^2\big)\,. \tag{19}$$

A method whose local error is $\mathcal{O}\big(h^{p+1}\big)$ is said to have **order** p; Euler's method, with local error $\mathcal{O}\big(h^2\big)$, has order $p = 1$.

Global error is the accumulated discrepancy between the numerical and exact solutions after many steps, $E_n \equiv y_n - y(t_n)$, once we actually run the method for $N \sim (T - t_0)/h$ steps rather than just one. Global error is not simply N times the local error added up; each step's local error also gets carried forward and reshaped by the dynamics over every subsequent step. The standard result — true under mild conditions on f, and something we will confirm numerically rather than prove here (Section 8.4) — is that this accumulation costs exactly *one power of* h: a method with local error $\mathcal{O}\big(h^{p+1}\big)$ has global error $\mathcal{O}(h^p)$.

Remark 5.1 *This one-power-of-h rule of thumb is why we always describe a method by its (global)* ***order*** *p rather than by the order of its local error: it is the global error, accumulated over the whole simulated time interval, that we actually care about. Euler's method, order $p = 1$, has global error $\mathcal{O}(h)$: halving the step size only halves the error, and achieving one additional digit of accuracy requires roughly a tenfold reduction in h (hence a tenfold increase in the number of, admittedly cheap, steps). This is the number to keep in mind when Section 6 introduces methods with $p = 4$, where the same one-digit improvement costs far fewer extra steps.*

5.4 Linear stability

Accuracy is not the only concern; a numerical method must also be *stable* for a given step size, or errors will grow rather than merely accumulate. Here is the idea in the plainest possible setting before we generalize it.

Suppose y decays exponentially, $y' = \lambda y$ with λ a negative real number, so the true solution $y(t) = y_0 e^{\lambda t}$ shrinks steadily toward zero. Apply one Euler step:

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)\, y_n. \tag{20}$$

Every step just multiplies the current value by the same fixed number, $(1 + h\lambda)$. That number is the whole story: if $|1 + h\lambda| < 1$, repeated multiplication shrinks y_n toward zero, matching the true decaying solution — good. But if $|1 + h\lambda| > 1$, repeated multiplication makes $|y_n|$ *grow without bound*, even though the true solution is decaying — the numerical scheme has manufactured an instability that has nothing to do with the real physics, purely because h was too large relative to $|\lambda|$. And if $1 + h\lambda$ happens to be negative (which happens once $h|\lambda| > 1$), the sign flips every step too, so an unstable run doesn't just grow, it *oscillates* while it grows — exactly the wild zig-zag you will compute by hand in Problem 4.

Write $z = h\lambda$ and $R(z) \equiv 1 + z$ for this multiplier (the **amplification factor**). The condition for stability, $|R(z)| \le 1$, traces out a disk in the complex z-plane — Euler's **region of absolute stability** — shown in Figure 3. For our purposes λ is real and negative (a diffusive or frictionally-damped relaxation rate, not an oscillation), so all that matters is the real-axis slice of that disk: stability requires

$$h \le \frac{2}{|\lambda|}. \tag{21}$$

The step size is bounded by the fastest-decaying mode in the system — the more strongly damped a process is, the smaller a step Euler needs to keep up with it.

Remark 5.2 *This is exactly the concern flagged in Session 5: once frictional heating and thermal pressurization couple into the spring-slider system, the effective $|\lambda|$ of the fastest mode grows sharply once slip accelerates, forcing Euler's step size down catastrophically at precisely the moment the simulation is most expensive to run. This is why the coupled system will eventually need either an implicit method or, at minimum, a higher-order explicit method with adaptive step control — the subject of the rest of this section and Section 6.9.*

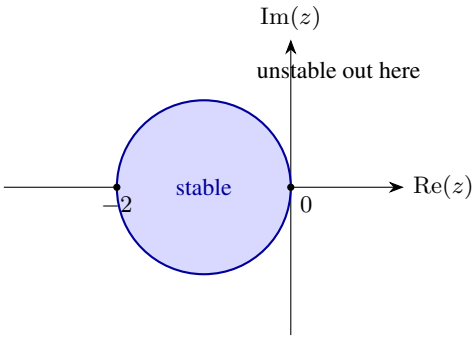

Figure 3: Euler's region of absolute stability: the shaded disk $|1 + z| \leq 1$, $z = h\lambda$. For real, negative λ, this is exactly the segment $-2 \leq z \leq 0$, i.e. $h \leq 2/|\lambda|$.

A short digression: why stiffness, and why eigenvalues?

Problems 4–6 all turn on the same underlying idea, so it's worth pausing to build it properly before diving in — not just naming it, but seeing where it actually comes from.

The question a coupled system raises. Section 5.4 analyzed exactly one scalar equation, $y' = \lambda y$, and got a clean stability bound: $h \leq 2/|\lambda|$. But our real system isn't scalar — τ and θ push on each other, and soon (Session 5) temperature and pore pressure will too. The natural question is: where does a single number like λ come from once the equations are coupled? Is there even such a number?

Locally, a coupled system isn't really coupled. The resolution is the same move used everywhere else in these notes: linearize. Near any given state $\mathbf{y}_0$, a nonlinear system $\mathbf{y}' = \mathbf{f}(\mathbf{y})$ looks, for small deviations $\delta\mathbf{y} = \mathbf{y} - \mathbf{y}_0$, like the linear system

$$\delta\mathbf{y}' \approx J\,\delta\mathbf{y}, \qquad J \equiv \left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_{\mathbf{y}_0}, \tag{22}$$

the Jacobian of $\mathbf{f}$ evaluated at the current state — exactly the multivariable version of the single derivative $f'(V_n)$ that drove Newton's method in Section 4. And a linear system, however coupled it looks written out in the original variables, is secretly just several *independent* scalar equations wearing a disguise: change coordinates to the eigenvectors of J, and each component z_i of $\delta\mathbf{y}$ in that basis evolves completely on its own,

$$z_i' = \lambda_i z_i, \tag{23}$$

one copy of Section 5.4's scalar test equation per eigenvalue λ_i of J. Nothing here is a linear-algebra abstraction bolted onto the problem — the eigenvalues are the decay (or growth) rates of the system's own natural, hidden, independent directions. A system with m state variables generically has m such directions, each ticking at its own rate, and you only see them once you look along the right axes.

Why this controls the step size. A numerical method has no idea this hidden decomposition exists — it just applies the same update rule to the whole vector $\mathbf{y}$ every step. But because the decomposition is linear, the method's action on $\delta\mathbf{y}$ splits the same way: it is, invisibly, applying the scalar Euler (or RK4) update to *every* mode z_i at once, superimposed. Stability of the whole system requires stability on *every* mode simultaneously, and a chain is only as strong as its weakest link: the step-size bound is set by whichever λ_i has the largest $|\mathrm{Re}(\lambda_i)|$ —

$$h \lesssim \frac{2}{\max_i |\mathrm{Re}(\lambda_i)|}, \tag{24}$$

even if that particular mode is a minor, fast-decaying, physically-uninteresting-by-now piece of the solution. The slow modes you actually care about resolving are just along for the ride, throttled by a clock that has nothing to do with them.

A semi-formal definition. This gives a working criterion for **stiffness**. Define the **stiffness ratio**

$$S = \frac{\max_i |\mathrm{Re}(\lambda_i)|}{\min_i |\mathrm{Re}(\lambda_i)|} \tag{25}$$

of the Jacobian J (restricting to eigenvalues with $\mathrm{Re}(\lambda_i) < 0$, the decaying modes relevant to diffusive and frictional relaxation). A system is called stiff, for a given explicit method, when *both*:

(i) $S \gg 1$ — the decay rates are widely separated, *and*

(ii) the interval you need to integrate over is comparable to the *slow* mode's own timescale, $1/\min_i |\mathrm{Re}(\lambda_i)|$ — i.e. the fast mode isn't what you actually need resolved; it may have already settled down.

When both hold, stability — not accuracy — is what's throttling your step size, for a mode you've already stopped caring about. That is the actual bite of stiffness: it is not a statement about how hard a problem is to get right, but about which constraint (stability or accuracy) is binding.

Remark 5.3 *This eigenvalue-ratio criterion is a* working *definition, useful and standard, but not a universally agreed rigorous one — stiffness has famously resisted a single clean formalization since the term was coined. Curtiss and Hirschfelder's original 1952 paper defines it almost operationally: a problem is stiff if explicit methods need impractically small steps to stay stable, while implicit methods (Section 6.11) do not. The eigenvalue ratio S is the standard way to make that operational statement checkable in advance, not a competing definition.*

Where we stand today. Right now, with only τ and θ as state variables, there is really just one relaxation clock in play — θ's own rate $|\lambda| \approx V/D_c$, the quantity you compute in Problem 6 — so S isn't yet meaningfully defined; a *ratio* needs at least two rates to compare. The moment Session 5 adds a diffusive temperature/pressure state with its own decay rate, the Jacobian becomes genuinely multi-eigenvalue, and that is the first point at which "eigenvalues," plural, stops being overkill notation and becomes the actual object governing the step size.

A quick practical diagnostic

- Estimate h_{acc}: the step size accuracy alone would ask for, to resolve the solution's actual features.
- Estimate $h_{\mathrm{stab}} \sim 2/\max_i |\mathrm{Re}(\lambda_i)|$: the step size your method's stability region permits, using the *fastest* decaying mode anywhere in the system.
- If $h_{\mathrm{stab}} \ll h_{\mathrm{acc}}$, the problem is stiff for that method: you're forced into far more steps than the solution's own behavior would ever require, purely to keep the fast mode from blowing up numerically.

Problem 6 is the version of this that matters for the rest of the course: the friction state variable relaxes at a rate $|\lambda| \approx V/D_c$ that is itself part of the solution, and swings across nine orders of magnitude between the interseismic and coseismic phases of a single earthquake cycle. The system is mild (or not stiff at all) while creeping, and becomes extremely stiff the moment slip accelerates — which is exactly why a single fixed step size can never be both safe and efficient across a full cycle, and why the coupled model of Session 5 will need either adaptive step control (Section 6.10) or an implicit method (Section 6.11), rather than the fixed-step RK4 that suffices for today's dry spring-slider.

6 Runge–Kutta Methods

6.1 Motivation

Euler's method uses the derivative $f(t_n, y_n)$ evaluated at a single point — the start of the step — and extrapolates linearly across the whole interval $[t_n, t_{n+1}]$. If f varies substantially across that interval (which it will, whenever the dynamics are fast relative to h), a single-point estimate of the slope is a poor representative of the average slope needed to get from y_n to y_{n+1} accurately. This is exactly the drift you saw in Figure 2: the slope Euler commits to at the start of the step is already stale by the end of it.

The idea behind Runge–Kutta (RK) methods is simply: sample the slope at *several* points across the step instead of just one, and combine the samples into a better estimate of the average slope. Figure 4 previews what this looks like for the

specific 4-point scheme we derive in Section 6.6: sample once at the start of the step, twice near the middle (each time with a better guess of where the solution actually is by then), and once at the end, then take a weighted average of all four. Critically, this is achieved *without* needing derivatives of f itself (unlike, say, a Taylor-series method that would require f, $f_t + f_y f$, and so on); only evaluations of f at various (t, y) pairs are needed, which is exactly what makes RK methods practical for the nonlinear, closed-form-derivative-free right-hand sides we will build later in the semester.

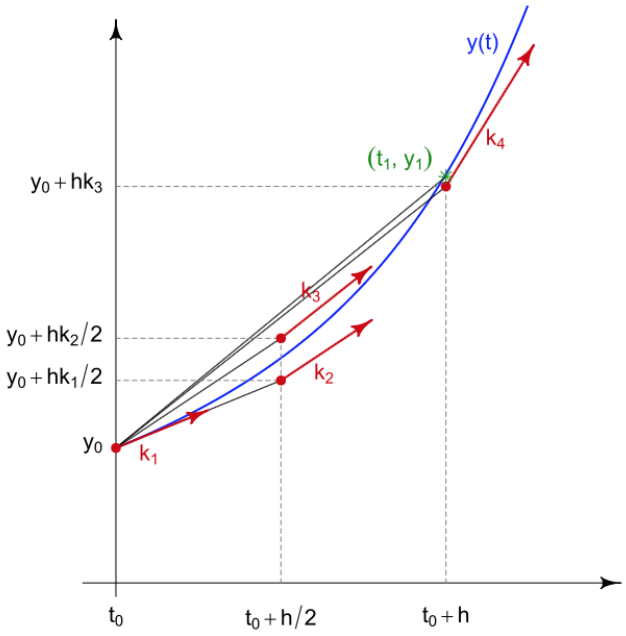

Figure 4: Where RK4 samples the slope within one step: once at the start, twice near the midpoint (using progressively better predictions of where the solution is by then), and once at the end. The final update (Section 6.6) averages these four, weighting the two midpoint samples twice as heavily as the two endpoint samples. Source: Wikipedia

6.2 The general explicit s-stage formula

An explicit Runge–Kutta method with s **stages** computes s intermediate slope estimates $k_1, \ldots, k_s$, each depending only on *previously computed* stages (hence "explicit"), and combines them into the update:

$$k_i = f\left(t_n + c_i h,\; y_n + h\sum_{j=1}^{i-1} a_{ij}\, k_j\right), \qquad i = 1, \ldots, s, \tag{26}$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i\, k_i. \tag{27}$$

The coefficients a_{ij} (for $j < i$), b_i, and c_i fully specify the method. Different choices of these coefficients, subject to certain consistency and order conditions (Section 6.4), give different named methods — Euler's method is the degenerate $s = 1$ case (a empty, $b_1 = 1$, $c_1 = 0$); the classical fourth-order method of Section 6.6 is the $s = 4$ case most students associate with the name "Runge–Kutta."

6.3 The Butcher tableau

The coefficients of an RK method are conventionally organized into a compact array called the **Butcher tableau** (Butcher, 1964), named for the numerical analyst John C. Butcher:

$$\begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b}^{\mathsf{T}} \end{array} \quad = \quad \begin{array}{c|cccc} c_1 & & & & \\ c_2 & a_{21} & & & \\ \vdots & \vdots & \ddots & & \\ c_s & a_{s1} & \cdots & a_{s,s-1} & \\ \hline & b_1 & \cdots & b_{s-1} & b_s \end{array} \tag{28}$$

where $A = (a_{ij})$ is a *strictly lower triangular* $s \times s$ matrix for an explicit method (the entries a_{ij} for $j \geq i$ are simply absent, or equivalently zero, which is precisely what makes the stage computation (26) explicit rather than requiring a nonlinear solve at each stage), $\mathbf{b} = (b_1, \ldots, b_s)^{\mathsf{T}}$ are the stage weights that appear in the final update (27), and $\mathbf{c} = (c_1, \ldots, c_s)^{\mathsf{T}}$ are the fractional locations, within the step, at which each stage's slope is evaluated.

The tableau is not merely a bookkeeping convenience: every property of the method we will discuss below — consistency, order, and linear stability — can be read directly off $(A, \mathbf{b}, \mathbf{c})$, which is why the tableau, rather than the explicit stage formulas, is the standard way numerical analysts communicate and compare RK methods.

6.4 Consistency and order conditions

For the method to make sense at all, the stage evaluated at fractional time c_i needs to be evaluated using a y-estimate appropriate to that fractional time — you can't sample the slope at the midpoint of the step using a y-value that only advanced a quarter of the way there. This bookkeeping requirement, the **row-sum condition**,

$$c_i = \sum_{j=1}^{i-1} a_{ij}, \qquad i = 1, \ldots, s, \tag{29}$$

is automatically satisfied by every tableau below, and it's standard practice to simply read c_i off the row sums of A rather than specify it independently.

Beyond that bookkeeping, getting a method to actually be accurate to a given order takes more: the coefficients must satisfy a handful of algebraic conditions, the **order conditions**, obtained by writing out the true solution and the numerical update as power series in h and demanding they agree term by term, as far as you want the order to reach. The first three are:

$$\text{order 1 (consistency):} \quad \sum_{i=1}^{s} b_i = 1, \tag{30}$$

$$\text{order 2 :} \quad \sum_{i=1}^{s} b_i c_i = \frac{1}{2}, \tag{31}$$

$$\text{order 3 :} \quad \sum_{i=1}^{s} b_i c_i^2 = \frac{1}{3}, \qquad \sum_{i,j} b_i a_{ij} c_j = \frac{1}{6}. \tag{32}$$

Order 1 (Eq. (30)) just says the weights b_i average to a full step: applied to the trivial ODE $y' = 1$, the method must reproduce the exact answer $y_{n+1} = y_n + h$ — the bare minimum for calling something a numerical integrator at all. Euler's method ($s = 1$, $b_1 = 1$) satisfies this but not Eq. (31) (since $c_1 = 0$ gives $b_1 c_1 = 0 \neq \frac{1}{2}$), consistent with it being only first order.

Where the order-2 condition comes from (optional derivation). It's worth seeing this worked out once, so the conditions above stop looking like they fell from the sky. For a general 2-stage method, write out k_2's Taylor expansion for small h, substitute into the update y_{n+1}, and compare against the true solution's own Taylor expansion (using $y' = f$ and $y'' = f_t + f f_y$ from the chain rule). Matching the h^1 terms of the two expansions gives $b_1 + b_2 = 1$; matching the h^2 terms gives $b_2 c_2 = \frac{1}{2}$ — exactly Eqs. (30)–(31). This is the general recipe behind every order condition at every order:

expand both sides in powers of h, and force them to agree.

The number of these conditions grows fast — 1 at order 1, 1 more at order 2, 2 more at order 3, 4 more at order 4 (8 total), 17 by order 5 — simply because there are more and more distinct ways for the chain rule to combine f and its derivatives the further out you expand. Fully organizing and generating them at high order is the subject of a nice piece of mathematics (Butcher's "rooted tree" theory) that we won't need here; we will simply use tableaus, like the ones below, that are already known to satisfy all the conditions up to the stated order. The one fact worth remembering is that it is possible to satisfy all 8 order-4 conditions with only 4 stages — no wasted effort — which is exactly what makes the classical RK4 method of Section 6.6 so effective for its cost.

6.5 Second-order methods

We build intuition with two of the simplest RK methods beyond Euler, both using $s = 2$ stages and both satisfying the order-1 and order-2 conditions, Eqs. (30)–(31), but not order 3.

Explicit midpoint method. Evaluate the slope once at t_n to estimate y at the *midpoint* of the interval, then use the slope *at that midpoint* for the full step:

$$k_1 = f(t_n, y_n), \qquad k_2 = f\big(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2}k_1\big)\,, \qquad y_{n+1} = y_n + h\,k_2. \tag{33}$$

Its Butcher tableau is

$$\begin{array}{c|cc} 0 & & \\ 1/2 & 1/2 & \\ \hline & 0 & 1 \end{array} \tag{34}$$

Check: $\sum b_i = 0+1 = 1$ (order 1, Eq. (30)); $\sum b_i c_i = 0\cdot 0 + 1\cdot \frac{1}{2} = \frac{1}{2}$ (order 2, Eq. (31)). Both are satisfied, confirming this is a second-order method built from only two function evaluations.

Heun's method (explicit trapezoidal / improved Euler). Evaluate the slope at both endpoints of the interval — using a preliminary (first-order accurate) Euler prediction for the endpoint value of y — and average the two slopes:

$$k_1 = f(t_n, y_n), \qquad k_2 = f(t_n + h,\, y_n + hk_1), \qquad y_{n+1} = y_n + \tfrac{h}{2}(k_1 + k_2). \tag{35}$$

Its Butcher tableau is

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array} \tag{36}$$

Check: $\sum b_i = \frac{1}{2} + \frac{1}{2} = 1$; $\sum b_i c_i = \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 1 = \frac{1}{2}$. Again both order conditions are satisfied. Both the midpoint method and Heun's method are second order and cost 2 function evaluations per step, but they are *different* methods (different tableaus), with different stability properties and different behavior on any given nonlinear problem — a first illustration of the fact, developed further below, that a given order does not determine a method uniquely.

The one-parameter family of second-order, 2-stage methods. In fact, the midpoint method and Heun's method are just two members of an entire family. For a general 2-stage explicit method with $c_2 = a_{21}$ free, the order-1 and order-2 conditions, $b_1 + b_2 = 1$ and $b_2 c_2 = \frac{1}{2}$, are *two* equations in *three* unknowns (b_1, b_2, c_2), so they determine a one-parameter family of solutions,

$$b_2 = \frac{1}{2c_2}, \qquad b_1 = 1 - \frac{1}{2c_2}, \qquad c_2 \neq 0 \text{ free}, \tag{37}$$

parametrized by the choice of $c_2 \in (0, 1]$ (values outside this range are possible but rarely useful). Setting $c_2 = \frac{1}{2}$ recovers the midpoint method ($b_1 = 0, b_2 = 1$); setting $c_2 = 1$ recovers Heun's method ($b_1 = b_2 = \frac{1}{2}$); setting $c_2 = \frac{2}{3}$ gives *Ralston's method* ($b_1 = \frac{1}{4}, b_2 = \frac{3}{4}$), which minimizes a bound on the leading third-order error term among this family and is, in that specific sense, the "best" second-order 2-stage method. This is the general pattern seen again at order 4 (Sections 6.6–6.7): the order conditions constrain but do not uniquely determine a method, and different members of a given order/stage family trade off differently on stability, error-constant size, and other secondary properties.

6.6 The classical fourth-order method (RK4)

The method usually meant when people say simply "Runge–Kutta," and the one you will implement in the coding session, uses $s = 4$ stages:

$$k_1 = f(t_n,\, y_n), \tag{38}$$
$$k_2 = f\big(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2}k_1\big)\,, \tag{39}$$
$$k_3 = f\big(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2}k_2\big)\,, \tag{40}$$
$$k_4 = f(t_n + h,\, y_n + h\,k_3)\,, \tag{41}$$
$$y_{n+1} = y_n + \frac{h}{6}\,(k_1 + 2k_2 + 2k_3 + k_4)\,. \tag{42}$$

Read as a narrative rather than a formula: k_1 is the slope at the start of the step; k_2 uses that slope to step halfway, and re-evaluates the slope there; k_3 re-evaluates the slope at the midpoint *again*, but now using the (presumably better) estimate of y at the midpoint supplied by k_2; and k_4 uses k_3 to step all the way to the end of the interval and evaluates the slope there. The final update is a weighted average of these four slope estimates, with the two midpoint estimates (k_2, k_3) weighted twice as heavily as the two endpoint estimates (k_1, k_4) — reminiscent of Simpson's rule for numerical quadrature, which weights function values at the midpoint of an interval four times as heavily as the endpoints, for exactly analogous reasons (both are constructed to be exact for cubic polynomials).

Its Butcher tableau is

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array} \tag{43}$$

As a partial check of the order-4 claim, verify the order-1 and order-2 conditions directly: $\sum b_i = \frac{1}{6} + \frac{1}{3} + \frac{1}{3} + \frac{1}{6} = 1$; and with $c = (0, \frac{1}{2}, \frac{1}{2}, 1)$, $\sum b_i c_i = \frac{1}{6}(0) + \frac{1}{3}(\frac{1}{2}) + \frac{1}{3}(\frac{1}{2}) + \frac{1}{6}(1) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}$. Both hold, as required; verifying the remaining six order-4 conditions is a mechanical (if tedious) exercise in the same spirit, and is the content of the classical derivation due to Kutta (1901).

RK4 costs 4 function evaluations per step (versus 1 for Euler) but achieves local truncation error $\mathcal{O}\big(h^5\big)$ and global error $\mathcal{O}\big(h^4\big)$ (versus $\mathcal{O}\big(h^2\big)$ and $\mathcal{O}(h)$ for Euler, Section 5.3) — for smooth problems, a dramatically better accuracy-per-step-size trade, which is why it is the default workhorse explicit integrator in scientific computing, and why it is the integrator you will build and use for the spring-slider model in Section 8.

6.7 A fully worked RK4 step

To make the abstract stage formulas concrete, work through one complete step by hand on a test problem with a known exact solution: $y' = -2ty$, $y(0) = 1$, whose exact solution is $y(t) = e^{-t^2}$. Take $h = 0.2$ and step from $t_0 = 0$, $y_0 = 1$ to $t_1 = 0.2$.

$$k_1 = f(t_0, y_0) = -2(0)(1) = 0, \tag{44}$$
$$k_2 = f\big(t_0 + \tfrac{h}{2},\, y_0 + \tfrac{h}{2}k_1\big) = f(0.1,\, 1 + 0.1 \times 0) = -2(0.1)(1) = -0.2, \tag{45}$$
$$k_3 = f\big(t_0 + \tfrac{h}{2},\, y_0 + \tfrac{h}{2}k_2\big) = f(0.1,\, 0.98) = -2(0.1)(0.98) = -0.196, \tag{46}$$
$$k_4 = f(t_0 + h,\, y_0 + h\,k_3) = f(0.2,\, 0.9608) = -2(0.2)(0.9608) = -0.38432, \tag{47}$$
$$y_1 = y_0 + \frac{h}{6}\,(k_1 + 2k_2 + 2k_3 + k_4) = 1 + \frac{0.2}{6}\big(0 - 0.4 - 0.392 - 0.38432\big) \approx 1 - 0.03881 = 0.96119. \tag{48}$$

Compare against the exact value $y(0.2) = e^{-0.04} \approx 0.96079$: the RK4 estimate is accurate to about 4×10^{-4} after a single step of size $h = 0.2$ — for comparison, one Euler step from the same starting point gives $y_1^{\text{Euler}} = y_0 + h\,k_1 = 1 + 0.2 \times 0 = 1$, an error of about 4×10^{-2}, roughly *one hundred times* worse, from a single function evaluation instead of four. This hundredfold gap for a hundredfold-cheaper method is, admittedly, an unusually dramatic illustration (it occurs here because $k_1 = 0$ exactly, so Euler's first step captures none of the local curvature at all), but it is directionally exactly

what Sections 5.3 and 6.6 predict, and it is the calculation you should reproduce as a first sanity check on your own RK4 implementation in the coding session: pick any test problem with a known solution, do one step by hand, and compare against your code's output before trusting it on the friction-coupled system.

6.8 Kutta's 3/8 rule: a second 4th-order method

It is worth seeing at least one other 4-stage, order-4 method, if only to underline that RK4 is not the unique way to achieve fourth-order accuracy with four stages — the order conditions of Section 6.4 constrain the coefficients but do not pin them down uniquely. Kutta's 3/8 rule (also due to Kutta, 1901) is

$$\begin{array}{c|cccc} 0 & & & & \\ 1/3 & 1/3 & & & \\ 2/3 & -1/3 & 1 & & \\ 1 & 1 & -1 & 1 & \\ \hline & 1/8 & 3/8 & 3/8 & 1/8 \end{array} \tag{49}$$

This method also satisfies all 8 order-4 conditions and so also achieves global error $\mathcal{O}(h^4)$, but with different intermediate stage locations $c = (0, \frac{1}{3}, \frac{2}{3}, 1)$ (evenly spaced across the interval, rather than clustered at the midpoint twice) and, as a consequence, a different stability region (Section 6.8) and different behavior on stiff or oscillatory problems. In practice the classical RK4 tableau above is used far more often, largely because its stability region is slightly larger, but the existence of the 3/8 rule is a useful reminder that "order 4" describes a whole family of methods, not a single one.

6.9 Linear stability of Runge–Kutta methods

The same idea from Section 5.4 carries over directly: apply an RK method to $y' = \lambda y$, and every step again just multiplies the current value by some fixed number $R(z)$, $z = h\lambda$ — only now $R(z)$ is a higher-degree polynomial instead of Euler's simple $1 + z$, because there are more stages contributing to the update. For the classical RK4 method, working through the four stages on this test problem collapses everything down to

$$R_{\mathrm{RK4}}(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}, \tag{50}$$

which is simply the first five terms of the Taylor series for e^z — a nice consistency check, since matching e^z to more and more terms is exactly what being higher order means (Section 6.4).

As with Euler, stability requires $|R(z)| \leq 1$. Figure 5 compares the two methods along the real axis, the case relevant to a purely decaying mode like ours: Euler's stable range ends at $z = -2$, while RK4's extends further, to about $z \approx -2.78$. In practical terms, RK4 tolerates a step size roughly $1.4\times$ larger than Euler for the same decay rate — a welcome bonus given that RK4 was already going to be used for its accuracy, not its stability. But notice it is still a *finite* range: push h far enough and RK4 becomes unstable too, exactly like Euler. No explicit method escapes this; only an implicit method (Section 6.10) can be stable for *every* step size.

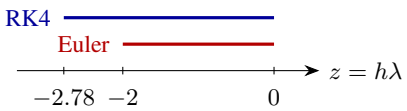

Figure 5: Where each method stays stable along the real axis (a purely decaying mode). RK4's stable range reaches further left than Euler's — about $1.4\times$ further — but both are bounded ranges, not the whole axis.

6.10 Embedded pairs and adaptive step size: a preview

Everything above uses a fixed step size h, chosen once and used throughout. In practice, and especially once thermal pressurization and dilatancy are coupled into the spring-slider system (Session 5), the natural time scale of the dynamics changes dramatically over an earthquake cycle: interseismic loading evolves over years, while coseismic slip can accelerate over milliseconds. A fixed h small enough to resolve the coseismic phase would make the interseismic phase

prohibitively (and pointlessly) expensive to simulate. What we want instead is a step size that adapts itself, growing when the solution is calm and shrinking when it isn't.

Getting two error estimates for the price of one. The standard solution is an **embedded Runge–Kutta pair**: two RK methods of different order, say p and $p+1$, constructed to *share* the same stage evaluations $k_1, \dots, k_s$ (differing only in the weights $\mathbf{b}$ used to combine them), so that both a p-th order and a $(p+1)$-th order estimate of y_{n+1} can be obtained from a single set of s function evaluations:

$$y_{n+1}^{(p)} = y_n + h \sum_{i=1}^{s} b_i k_i, \qquad y_{n+1}^{(p+1)} = y_n + h \sum_{i=1}^{s} b_i^* k_i, \tag{51}$$

using the *same* $k_1, \dots, k_s$ in both sums — only the weights $\mathbf{b}$ versus $\mathbf{b}^*$ differ. The difference between the two estimates,

$$\widehat{e}_{n+1} \;=\; y_{n+1}^{(p+1)} - y_{n+1}^{(p)} \;=\; h \sum_{i=1}^{s} (b_i^* - b_i)\, k_i, \tag{52}$$

is then a cheap, practical estimate of the local truncation error of the lower-order method — cheap because it costs no extra function evaluations beyond the s we were already computing.

Turning the error estimate into a step size. Recall from Section 5.3 that a p-th order method has local error $\mathcal{O}\big(h^{p+1}\big)$, i.e. $|\widehat{e}_{n+1}| \approx C h^{p+1}$ for some constant C that depends on the local behavior of the solution but not on h. If we specify a tolerance tol we want the local error to stay below, we can solve this relationship for the step size that *would have* hit that tolerance exactly, and use it as our next attempt. Writing the scalar error ratio

$$\mathrm{err}_{n+1} \;=\; \frac{|\widehat{e}_{n+1}|}{\mathrm{tol}}, \tag{53}$$

(with $\mathrm{tol} = \mathrm{atol} + \mathrm{rtol}\,|y_n|$ a mix of absolute and relative tolerance, in practice), the standard step-size update is

$$\boxed{h_{\mathrm{new}} \;=\; h \cdot S \left(\frac{1}{\mathrm{err}_{n+1}} \right)^{1/(p+1)}} \tag{54}$$

where $S \approx 0.9$ is a **safety factor**, pulling back slightly from the theoretical optimum so the next step is more likely to succeed too, rather than landing right on the edge of the tolerance. In practice h_{new} is also clamped to a modest range around h (e.g. never grow by more than a factor of 5 or shrink by more than a factor of 10 in one step), so that a single wild error estimate can't send the step size to an extreme in one jump.

Accept or reject. Unlike the fixed-step algorithm of Section 8, an adaptive step is provisional until checked:

- If $\mathrm{err}_{n+1} \le 1$ (the error came in under tolerance), **accept** the step — advance to $y_{n+1}^{(p+1)}$ (the higher-order estimate; this free upgrade is called **local extrapolation**) — and use h_{new} from Eq. (54) for the *next* step.
- If $\mathrm{err}_{n+1} > 1$ (the error exceeded tolerance), **reject** the step: discard y_{n+1} entirely, and retry the *same* step from y_n using the smaller h_{new} that Eq. (54) already computed.

Either way, no work is wasted: the same s function evaluations that produced the (accepted or rejected) step also told us what step size to try next.

One adaptive step, embedded RK pair

1. Compute stages $k_1, \dots, k_s$ once (Eq. (26)).
2. Form both estimates $y_{n+1}^{(p)}$, $y_{n+1}^{(p+1)}$ and the error estimate $\widehat{e}_{n+1}$, Eq. (52).
3. Compute $\mathrm{err}_{n+1} = |\widehat{e}_{n+1}|/\mathrm{tol}$ and h_{new} from Eq. (54).
4. If $\mathrm{err}_{n+1} \le 1$: accept, advance to $y_{n+1}^{(p+1)}$, move on with step size h_{new}.
5. If $\mathrm{err}_{n+1} > 1$: reject, retry the same step from y_n with step size h_{new}.

The best-known embedded pair is the Dormand–Prince method (RK45, using a 5th- and a 4th-order estimate from 6 or 7 stages), which underlies the default adaptive ODE solvers in most scientific software (e.g. `scipy.integrate.solve_ivp`'s `RK45`, MATLAB's `ode45`). We will not derive an embedded pair today, but the coupling session (Session 5) will build directly on the fixed-step RK4 method developed here, adding exactly this accept/reject/resize machinery of Eq. (54) once the coupled dynamics make a fixed step size impractical.

6.11 A note on implicit methods and stiffness

Every method presented above is *explicit*: each stage k_i depends only on previously computed stages, so the strictly lower-triangular structure of A lets us evaluate $k_1, k_2, \dots$ one at a time, in order, with no algebraic solve beyond the friction-law root-find already embedded in each stage's right-hand side. This is what keeps the method cheap.

The alternative is an **implicit** Runge–Kutta method, in which A is full (or at least not strictly lower triangular), so that k_i depends on k_j for $j \ge i$ as well as $j < i$ — meaning *all* the stages must, in general, be solved for *simultaneously*, as a coupled system of nonlinear equations (via, unsurprisingly, a multivariate generalization of Newton's method). The simplest example is the **backward (implicit) Euler** method,

$$y_{n+1} = y_n + h\, f(t_{n+1}, y_{n+1}), \tag{55}$$

whose Butcher tableau is the trivial 1×1 array $c_1 = 1$, $a_{11} = 1$, $b_1 = 1$ — note a_{11} is on the diagonal, not below it, which is exactly what makes the method implicit. Applying it to the linear test equation $y' = \lambda y$ gives the amplification factor $R(z) = 1/(1-z)$, whose stability region $\{z : |1/(1-z)| \le 1\}$ is the *entire left half of the complex plane*: backward Euler is stable for *any* step size $h > 0$ whenever $\mathrm{Re}(\lambda) \le 0$ (a property called **A-stability**), in sharp contrast to the bounded stability disks of every explicit method discussed above (Sections 5.4 and 6.8).

A closely related, and frequently used, alternative is the **trapezoidal rule** (also called Crank–Nicolson in the PDE literature), which averages the explicit and implicit Euler right-hand sides:

$$y_{n+1} = y_n + \frac{h}{2}\Big[f(t_n, y_n) + f(t_{n+1}, y_{n+1})\Big], \tag{56}$$

with Butcher tableau

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}, \tag{57}$$

an implicit 2-stage method (note the nonzero $a_{22} = 1/2$ on the diagonal of the second row) that is second order (it satisfies Eqs. (30)–(31) by the same check as Heun's method, whose tableau it resembles but for that one diagonal entry) *and* A-stable, with amplification factor $R(z) = (1 + z/2)/(1 - z/2)$. It is, in a sense, the implicit counterpart to Heun's method, trading Heun's conditional stability for unconditional stability at second, rather than first, order — a strictly better stability-for-accuracy trade than backward Euler, at the cost of the same per-stage nonlinear solve.

This unconditional stability is purchased at the price of a nonlinear solve, of comparable difficulty to the friction-law root-find of Section 4, *at every single stage* rather than merely embedded in evaluating an explicit right-hand side. Whether that price is worth paying is precisely a question of **stiffness**: how large is $|\lambda|$ (the fastest decay or growth rate present in the system) relative to the time scale of the dynamics you actually care about resolving? Today's dry spring-slider is not stiff, and RK4 is the right tool. Once thermal pressurization couples frictional heating to pore pressure to weakening to yet faster slip (Session 5), the fastest mode in the system accelerates sharply during the coseismic phase, and the

choice between a very small explicit step and an implicit (or, more practically, a semi-implicit / adaptive-explicit) method becomes a live design decision rather than an academic one. We flag it here so that when it resurfaces in Session 5, it is a familiar idea being applied, not a new one being introduced under time pressure.

7 From Loops to Vectorized Code

7.1 Why this matters, twice more, later

Everything above is naturally expressed, and will be implemented in the coding session, as scalar Python functions acting on scalar (or, once we track τ and θ together, small-vector) state. That implementation effort pays for itself twice more this semester: in Sessions 3–4, assembling and repeatedly multiplying an $N \times N$ elastic stiffness matrix (one entry per pair of fault patches) is only tractable if done with vectorized array operations rather than explicit Python loops; and in Sessions 5–6, evaluating the friction law, the diffusion update, and the dilatancy relaxation together, at every one of possibly thousands of time steps, has the same requirement. Shaky vectorization habits formed now compound into real performance and readability costs later, which is why we spend a short amount of time on it today, even though it is not, strictly, numerical analysis.

7.2 Arrays, elementwise operations, and broadcasting

A NumPy array supports elementwise arithmetic directly: if `V` is an array of slip velocities, `k * V` multiplies every entry by the scalar `k` without an explicit loop, and `V1 - V2` subtracts two same-shaped arrays elementwise. This is not merely syntactic convenience; vectorized operations are implemented in compiled C code under the hood and are typically one to two orders of magnitude faster than the equivalent explicit `for` loop in pure Python, for array sizes of practical interest.

Broadcasting is the rule that lets NumPy combine arrays of different (but compatible) shapes without the user manually replicating data: a scalar combines with an array of any shape; more generally, two array shapes are compatible if, comparing dimensions from the right, each pair of dimensions is either equal or one of them is 1. This is the same rule, scaled up, that will let us write the N-patch elastic interaction as a single matrix–vector product `K @ slip` in Session 4, rather than a double loop over patch pairs.

7.3 Writing the friction law and residual as vectorized functions

The residual function, Eq. (6), and its derivative, Eq. (10), should be written once as plain functions of `V` (and the other state), which then work correctly whether `V` is a single float (a single patch, today's model) or a full array (many patches, Session 6 onward) — provided every operation inside the function is itself elementwise (as `np.log`, unlike Python's `math.log`, is). Writing code this way from the outset, rather than special-casing the scalar and vector cases, is the single highest-leverage habit from today's numpy discussion.

8 Assembling the Spring-Slider Model

8.1 The governing equations, together

We now have every ingredient needed to simulate the single-patch spring-slider system introduced in Section 2. The state we integrate forward in time is the pair (τ, θ); the governing ODEs are the elastic loading equation, Eq. (1), and the state evolution equation, Eq. (3):

$$\frac{\mathrm{d}\tau}{\mathrm{d}t} = k\left(V_{\mathrm{load}} - V\right), \tag{58}$$

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = 1 - \frac{V\theta}{D_c}. \tag{59}$$

Crucially, V is *not* itself a state variable that we integrate: it is a *diagnostic* quantity, recovered at every instant (and, as we will see, at every RK stage) from the current (τ, θ) by solving the friction-law residual, Eq. (6), via Newton's method.

This is the sense in which the system is a genuinely coupled algebraic-differential problem, not a plain ODE system: the right-hand side of Eqs. (58)–(59) cannot be evaluated without first solving a nonlinear equation.

8.2 One coupled RK4 step, in full

Write the state vector as $\mathbf{y} = (\tau, \theta)$ and the right-hand side as $\mathbf{F}(\mathbf{y}) = \big(k(V_{\text{load}} - V(\mathbf{y})),\ 1 - V(\mathbf{y})\theta/D_c\big)$, where $V(\mathbf{y})$ denotes "solve the friction residual, Eq. (6), for V given the τ and θ components of $\mathbf{y}$, via Newton's method." A single RK4 step, Eqs. (26)–(27) specialized to $s = 4$, then reads:

$$\mathbf{k}_1 = \mathbf{F}(\mathbf{y}_n), \qquad \text{(Newton-solve for } V \text{ at } \mathbf{y}_n\text{, then RHS)} \tag{60}$$

$$\mathbf{k}_2 = \mathbf{F}\big(\mathbf{y}_n + \tfrac{h}{2}\mathbf{k}_1\big)\,, \qquad \text{(Newton-solve at the midpoint predictor)} \tag{61}$$

$$\mathbf{k}_3 = \mathbf{F}\big(\mathbf{y}_n + \tfrac{h}{2}\mathbf{k}_2\big)\,, \qquad \text{(Newton-solve at the refined midpoint)} \tag{62}$$

$$\mathbf{k}_4 = \mathbf{F}\big(\mathbf{y}_n + h\,\mathbf{k}_3\big)\,, \qquad \text{(Newton-solve at the endpoint predictor)} \tag{63}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right). \tag{64}$$

Notice that a single RK4 step of the coupled system requires *four separate Newton solves* for V, one per stage, each nested inside the RK4 stage evaluation. This nested structure — root-find inside a time-integration stage, repeated at every stage of every step — is the architectural pattern flagged in Section 1, and it will recur, essentially unchanged in outline, all the way through the multi-patch, multi-physics system of Session 6, where each RK stage will instead require a Newton solve (or, later, a full coupled diffusion-plus-friction solve) at *every one of N patches simultaneously*.

8.3 Pseudocode

One coupled time step of the spring-slider model

1. **Input:** current state (τ_n, θ_n), step size h, parameters (k, V_{load}, σ_n, a, b, D_c, μ^*, V^*).
2. **Define** `rhs(tau, theta)`:
   (a) Solve $f(V) = \tau_{\text{friction}}(V, \theta) - \tau = 0$ for V by Newton–Raphson (Section 4), warm-started from the previous accepted V, ideally in $\log V$ (Section 4.5).
   (b) Return $\big(k(V_{\text{load}} - V),\ 1 - V\theta/D_c\big)$.
3. Compute $\mathbf{k}_1 = \texttt{rhs}(\tau_n, \theta_n)$.
4. Compute $\mathbf{k}_2 = \texttt{rhs}\big(\tau_n + \tfrac{h}{2}k_{1,\tau},\ \theta_n + \tfrac{h}{2}k_{1,\theta}\big)$.
5. Compute $\mathbf{k}_3 = \texttt{rhs}\big(\tau_n + \tfrac{h}{2}k_{2,\tau},\ \theta_n + \tfrac{h}{2}k_{2,\theta}\big)$.
6. Compute $\mathbf{k}_4 = \texttt{rhs}(\tau_n + h\,k_{3,\tau},\ \theta_n + h\,k_{3,\theta})$.
7. Update: $(\tau_{n+1}, \theta_{n+1}) = (\tau_n, \theta_n) + \tfrac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$.
8. **Output:** $(\tau_{n+1}, \theta_{n+1})$; repeat for the next step.

This is precisely what you will implement, and then run on a baseline (dry, undamped) spring-slider, in the coding session: build the Newton solver and friction-law wrapper first, validate it in isolation against a known root, then wrap it in the RK4 integrator above and confirm that $V(t)$ shows the expected interseismic-loading / coseismic-slip alternation of a simple stick-slip cycle.

8.4 Validating the integrator: a numerical convergence check

Before trusting RK4 on a problem as nonlinear as the friction-coupled spring-slider, it is standard — and will be the first thing you do in the coding session — to validate the integrator on a problem with a known exact solution, and check that the *observed* rate of convergence matches the *theoretical* order derived in Sections 5.3 and 6.4. Take the simplest possible

test problem, $y' = -y$, $y(0) = 1$, with exact solution $y(t) = e^{-t}$. Integrate to $t = 1$ with both Euler and RK4, at a sequence of step sizes h, and record the error $|y_N - e^{-1}|$ at the final time.

h	Euler error	RK4 error	
0.1	$\sim 1.7 \times 10^{-2}$	$\sim 2 \times 10^{-6}$	
0.05	$\sim 8.7 \times 10^{-3}$	$\sim 1.3 \times 10^{-7}$	
0.025	$\sim 4.4 \times 10^{-3}$	$\sim 8 \times 10^{-9}$	
observed ratio on halving h		$\approx 2\times$	$\approx 16\times$

The pattern to check for is the *ratio* of successive errors as h is halved, not the absolute error values (which depend on the particular test problem). Because Euler is order 1, halving h should roughly halve the error ($2^1 = 2$); because RK4 is order 4, halving h should reduce the error by a factor of roughly $2^4 = 16$. Observing ratios close to these values, computed from your own implementation on exactly this test problem, is the standard first check that a from-scratch integrator has been implemented correctly — a coding bug that still "basically works" will often nonetheless fail to reproduce the correct *order* of convergence, making this check considerably more diagnostic than simply eyeballing whether $y(t)$ looks qualitatively reasonable.

Remark 8.1 *The same check, applied instead to the Newton solver of Section 4, is to fix a residual f with a known root, start from a sequence of initial guesses at varying distances from the root, and confirm that $|e_{n+1}|/|e_n|^2$ approaches a constant (Section 4.3) rather than $|e_{n+1}|/|e_n|$ — i.e. confirm quadratic, not merely linear, convergence. Both checks are cheap, take only a few lines of code, and are worth doing before ever running the coupled model.*

9 Summary and Where We're Headed

Today's session built exactly two numerical tools, in enough depth to trust them for the rest of the semester:

- **Newton–Raphson**, derived from a first-order Taylor expansion, converges quadratically once near a simple root (Section 4.3), but requires care — a good initial guess, and ideally a log-velocity reparametrization — to behave well across the many orders of magnitude of slip velocity relevant to an earthquake cycle.
- **Explicit Runge–Kutta methods**, and the classical fourth-order method (RK4) in particular, sample the right-hand side of an ODE at several points per step and combine them via a Butcher tableau $(A, \mathbf{b}, \mathbf{c})$ to achieve much higher accuracy per step than Euler's method, at the cost of more function evaluations per step — a cost that is essentially free once each function evaluation is itself just a fast Newton solve.

These two tools compose: a single RK4 step of the coupled spring-slider system requires four Newton solves for slip velocity, one per stage (Section 8.2). This root-find-inside-a-time-step pattern is the architecture underlying every remaining session of this course — only the physics evaluated inside the root-find and inside the RHS will grow, from a single patch with dry friction today, to N patches with diffusive thermal pressurization and dilatant gouge by Session 6.

In the coding session immediately following, you will implement: (i) the Newton solver and friction-law wrapper, validated against a known root; (ii) the RK4 integrator, validated against a problem with a known analytic solution; and (iii) the coupled baseline spring-slider model above, run three ways in later sessions — dry, with thermal pressurization, and with thermal pressurization plus dilatancy — to see directly how each additional piece of physics reshapes the cycle.

A Notation

Symbol	Meaning
V	Slip velocity (diagnosed from the friction law, not integrated directly)
V_{load}	Far-field (plate) loading rate
τ	Shear stress on the fault patch
θ	State variable (rate-and-state friction), dimensions of time
σ_n	Effective normal stress
μ^*	Reference friction coefficient
V^*	Reference slip velocity
a, b	Direct and evolution-effect rate-and-state coefficients
D_c	Critical slip distance
k	Elastic (spring) stiffness
$f(V)$	Residual of the friction force balance, Eq. (6)
h	Time step size
s	Number of stages in a Runge–Kutta method
$A, \mathbf{b}, \mathbf{c}$	Butcher tableau: stage coupling matrix, weights, and abscissae
k_i	i-th stage slope estimate of an RK method (context: not the spring stiffness)
$R(z)$	Linear stability (amplification) function, $z = h\lambda$

B Suggested reading

- Dieterich, J. H. (1979). Modeling of rock friction: 1. Experimental results and constitutive equations. *J. Geophys. Res.*, 84(B5), 2161–2168.
- Ruina, A. (1983). Slip instability and state variable friction laws. *J. Geophys. Res.*, 88(B12), 10359–10370.
- Butcher, J. C. (2016). *Numerical Methods for Ordinary Differential Equations*, 3rd ed. Wiley. — the standard modern reference for Runge–Kutta theory and Butcher tableaus.
- Hairer, E., Nørsett, S. P., & Wanner, G. (1993). *Solving Ordinary Differential Equations I: Nonstiff Problems*, 2nd ed. Springer. — Chapter II covers Runge–Kutta methods and order conditions in full rigor.
- Kelley, C. T. (2003). *Solving Nonlinear Equations with Newton's Method*. SIAM. — a compact, practically minded treatment of Newton's method and its failure modes.

C Practice Problems: Numerical Pitfalls

The problems below are deliberately chosen so that the "obvious" numerical approach *fails* — either Newton's method diverges or cycles, or Euler's method blows up — so that the failure itself, not just the successful cases in the main text, becomes part of what you understand about these methods. Work each one with pencil, a calculator, or a few lines of Python before checking the solution; the point is to watch the failure happen in real numbers, not just to know abstractly that it can.

Problem 1 — Newton's method can cycle forever, even for a smooth polynomial

Let $f(x) = x^3 - 2x + 2$, which has a single real root at $x^* \approx -1.7693$.

(a) Starting from $x_0 = 0$, compute x_1 and x_2 by hand using Newton's method, $x_{n+1} = x_n - f(x_n)/f'(x_n)$.

(b) Compute x_3 and x_4. What do you notice?

(c) Explain, using a rough sketch of f near $x = 0$ and $x = 1$ (tangent lines are enough — no need to plot precisely), why the tangent line at $x = 0$ points to $x = 1$, and the tangent line at $x = 1$ points back to $x = 0$.

Solution. $f'(x) = 3x^2 - 2$.

(a) $f(0) = 2$, $f'(0) = -2$, so $x_1 = 0 - \dfrac{2}{-2} = 1$. Then $f(1) = 1 - 2 + 2 = 1$, $f'(1) = 3 - 2 = 1$, so $x_2 = 1 - \dfrac{1}{1} = 0$.
(b) $x_3 = x_1 = 1$ and $x_4 = x_0 = 0$: the iteration repeats $0, 1, 0, 1, \ldots$ forever, never approaching x^*.
(c) The tangent line at $x = 0$ is $y = f(0) + f'(0)(x - 0) = 2 - 2x$, which crosses zero at $x = 1$. The tangent line at $x = 1$ is $y = f(1) + f'(1)(x - 1) = 1 + (x - 1) = x$, which crosses zero at $x = 0$. The two tangent lines exactly swap 0 and 1, producing a perfect 2-cycle.

Pitfall to notice: The convergence picture from Section 4.3 only guarantees fast convergence *once you are already close to a root*. It says nothing about what happens far away — and here, $x_0 = 0$ is far enough from $x^* \approx -1.77$ that Newton's method doesn't converge at all: it falls into a perfect 2-cycle and repeats forever. A convergence check based on $|x_{n+1} - x_n|$ would loop indefinitely; a robust implementation needs a maximum iteration count and a fallback (Section 4.6).

Problem 2 — Newton's method on the friction law: raw V overshoots into $V < 0$

Take the friction-law residual with $\sigma_n = 100$ MPa, $a = 0.01$, $V^* = 10^{-6}$ m/s, $\mu^* = 0.6$, and θ held at its steady-state value so that the $b \ln(\theta V^*/D_c)$ term vanishes:

$$f(V) = 100 \left[0.6 + 0.01 \ln\left(\frac{V}{10^{-6}} \right) \right] - \tau_{\text{applied}}, \qquad f'(V) = \frac{1}{V} \text{ (MPa·s/m, with } V \text{ in m/s).}$$

Suppose the fault is very strongly loaded down, $\tau_{\text{applied}} = 40$ MPa, so that the true root is extremely small: solving $0.6 + 0.01 \ln(V^*_{\text{root}}/10^{-6}) = 0.4$ gives $V^*_{\text{root}} = 10^{-6} e^{-20} \approx 2.06 \times 10^{-15}$ m/s.

(a) Starting from the "reasonable-looking" guess $V_0 = 10^{-6}$ m/s $= V^*$, compute $f(V_0)$ and $f'(V_0)$, and take one Newton step to find V_1.

(b) What is wrong with V_1? (Hint: check its sign, and check whether f is even defined there.)

(c) Now repeat the solve in $x = \ln(V/V^*)$: the residual becomes $g(x) = 20 + x$ (verify this), with $g'(x) = 1$. Starting from $x_0 = 0$ (i.e. the same initial guess $V_0 = V^*$), take one Newton step in x and convert back to $V_1 = V^* e^{x_1}$. Compare to the true root above.

Solution.

(a) $f(V_0) = 100(0.6) - 40 = 20$ MPa, and $f'(V_0) = 1/10^{-6} = 10^6$ MPa·s/m. One Newton step: $V_1 = 10^{-6} - \dfrac{20}{10^6} = 10^{-6} - 2 \times 10^{-5} = -1.9 \times 10^{-5}$ m/s.
(b) $V_1 < 0$: the friction law's logarithm has no real argument there, so $f(V_1)$ isn't even defined — the raw-V solver has crashed, not merely converged slowly.
(c) $g(x) = 100(0.6 + 0.01x) - 40 = 60 + x - 40 = 20 + x$, confirming the stated form. With $x_0 = 0$: $g(0) = 20$, $g'(0) = 1$, so $x_1 = 0 - 20/1 = -20$. Converting back, $V_1 = 10^{-6} e^{-20} \approx 2.06 \times 10^{-15}$ m/s — exactly the true root, in a *single* step, because g is exactly linear in x here.

Pitfall to notice: The raw-V Newton step overshoots by more than the entire distance from V_0 down to zero, landing on a negative number where the friction law's logarithm is undefined — the solver doesn't just converge slowly, it crashes. The $\log V$ reparametrization doesn't merely avoid the crash: because the reparametrized residual is *exactly linear* in x here, a single Newton step lands exactly on the root. This is the sharpest possible illustration of why Section 4.5 insists on solving in $\log V$ once the true root can be many orders of magnitude from a typical initial guess.

Problem 3 — Newton's method can diverge even when f is smooth everywhere

Let $f(x) = \arctan(x)$, with a single root at $x^* = 0$ and $f'(x) = 1/(1 + x^2)$, which is smooth, bounded, and nonzero for every x — none of the obvious failure conditions apply.

(a) Starting from $x_0 = 2$, compute $f(x_0)$, $f'(x_0)$, and x_1. (Use $\arctan(2) \approx 1.1071$.)

(b) Compute x_2 using $\arctan(-3.536) \approx -1.294$.

(c) Compute $|x_2|$ and compare it to $|x_1|$ and $|x_0|$. What is the trend?

Solution.

(a) $f(2) = \arctan(2) \approx 1.1071$, $f'(2) = 1/(1+4) = 0.2$. $x_1 = 2 - \dfrac{1.1071}{0.2} = 2 - 5.5355 = -3.5355$.

(b) $f(x_1) = \arctan(-3.5355) \approx -1.294$, $f'(x_1) = 1/(1+3.5355^2) = 1/13.50 \approx 0.0741$. $x_2 = -3.5355 - \dfrac{-1.294}{0.0741} \approx -3.5355 + 17.47 = 13.94$.

(c) $|x_0| = 2$, $|x_1| \approx 3.54$, $|x_2| \approx 13.94$: the iterates grow, roughly by a factor of 3–4 each step, diverging away from the root rather than approaching it.

Pitfall to notice: Far from the root, the tangent line at a point where $|f'(x)|$ is small (here, $f'(x) \to 0$ as $|x| \to \infty$) is nearly flat, so the Newton step $-f(x_n)/f'(x_n)$ is huge — the iterate rockets past the root to a point even farther out, where the derivative is smaller still, and the process runs away to $\pm\infty$. (One can show this happens for any $|x_0|$ larger than about 1.39.) The lesson: smoothness and a nonzero derivative *everywhere* are not enough to guarantee convergence from an arbitrary starting guess — only *local* behavior near the root controls that, exactly as Section 4.3 describes.

Problem 4 — Euler's method can blow up on a problem that is decaying

Consider $y' = -100y$, $y(0) = 1$, with exact solution $y(t) = e^{-100t}$ — a rapidly *decaying* function.

(a) Using Euler's method with $h = 0.025$, compute y_1, y_2, y_3, y_4 by hand, using $y_{n+1} = (1 - 100h)y_n$.

(b) Compare the magnitude of y_4 to $y_0 = 1$. Is this remotely close to the true solution, which has decayed to essentially zero well before $t = 4h = 0.1$?

(c) Now redo the calculation with $h = 0.005$. Does the qualitative behavior (growing/oscillating vs. decaying) change? Using the stability condition $h \le 2/|\lambda|$ from Section 5.4, find the exact largest stable h for this problem, and check which of your two step sizes violates it.

Solution.

(a) Amplification factor $1 - 100(0.025) = -1.5$. So $y_1 = -1.5$, $y_2 = (-1.5)^2 = 2.25$, $y_3 = (-1.5)^3 = -3.375$, $y_4 = (-1.5)^4 = 5.0625$.

(b) $|y_4| \approx 5.06$, while the true solution at $t = 0.1$ is $e^{-10} \approx 4.5 \times 10^{-5}$, essentially zero. The numerical solution has grown to over five times its starting value, oscillating in sign along the way — the complete opposite of decay.

(c) With $h = 0.005$: factor $= 1 - 100(0.005) = 0.5$, so $y_1 = 0.5$, $y_2 = 0.25$, $y_3 = 0.125$, $y_4 = 0.0625$ — smoothly decaying, qualitatively correct. The exact stability bound is $h \le 2/|\lambda| = 2/100 = 0.02$. The first step size, $h = 0.025$, violates this ($0.025 > 0.02$); the second, $h = 0.005$, satisfies it comfortably.

Pitfall to notice: With $h = 0.025$, the amplification factor is $1 + h\lambda = 1 - 2.5 = -1.5$, outside Euler's stability disk $|1 + z| \le 1$: the numerical solution *oscillates in sign and grows in magnitude by 50% every step*, diverging to infinity, even though the true solution is monotonically decaying to zero. This is not a small accuracy error — it is a qualitative failure, and it would be easy to mistake the wild oscillation for a real physical instability rather than a numerical artifact, if you didn't know to check the stability condition first.

Problem 5 — RK4 buys a larger stability region, but not an unlimited one

Using the same $\lambda = -100$ as Problem 4, and the RK4 stability function $R(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}$:

(a) Evaluate $R(z)$ at $z = h\lambda = -2.5$ (i.e. $h = 0.025$, the same step size that broke Euler in Problem 4). Is $|R(-2.5)| \le 1$? What does this tell you about whether RK4 would have handled Problem 4(a) safely?

(b) Now evaluate $R(z)$ at $z = -3.0$ (i.e. $h = 0.03$). Is RK4 stable here?

(c) Using $|R(z)| \leq 1$ numerically or by trial and error, estimate the largest h for which RK4 remains stable at $\lambda = -100$, to the nearest 0.001. Compare it to Euler's stability bound from Problem 4(c).

Solution.

(a) $R(-2.5) = 1 - 2.5 + \dfrac{6.25}{2} - \dfrac{15.625}{6} + \dfrac{39.0625}{24} = 1 - 2.5 + 3.125 - 2.604 + 1.628 \approx 0.648$. Since $|0.648| \leq 1$, RK4 *is* stable at this step size — it would have integrated Problem 4(a) correctly, with no blow-up, using four times the function evaluations per step but a qualitatively correct answer instead of a diverging one.
(b) $R(-3.0) = 1 - 3 + 4.5 - 4.5 + 3.375 = 1.375$. Since $|1.375| > 1$, RK4 is unstable here.
(c) The real-axis stability boundary for RK4 is at $z \approx -2.785$, so $h_{\max} = 2.785/100 \approx 0.028$. Euler's bound was $h \leq 0.02$; RK4's is about $1.4\times$ larger ($0.028/0.02 \approx 1.39$).

Pitfall to notice: RK4 tolerates a step roughly $1.4\times$ larger than Euler before going unstable ($|z| \lesssim 2.78$ vs. Euler's $|z| \leq 2$) — genuinely useful, and free, since RK4 was already going to be used for its accuracy. But it is not a cure for stiffness: push h far enough (part (b)) and RK4 diverges too. *Every explicit method has a finite stability region.* This is exactly why Section 6.11 flags implicit methods, not just higher-order explicit ones, as the eventual fix once the coupled system (Session 5) becomes genuinely stiff.

Problem 6 — Why one fixed step size cannot serve the whole earthquake cycle

The state-evolution equation $\mathrm{d}\theta/\mathrm{d}t = 1 - V\theta/D_c$, linearized about its steady state $\theta_{ss} = D_c/V$, relaxes at an effective rate $|\lambda| \approx V/D_c$. Take $D_c = 10\ \mu\mathrm{m} = 10^{-5}\ \mathrm{m}$.

(a) During interseismic loading, $V \sim 10^{-9}\ \mathrm{m/s}$. Estimate $|\lambda|$, and use Euler's stability bound $h \leq 2/|\lambda|$ (Section 5.4) to estimate the *largest* stable step size, in seconds. Convert to a more natural unit (hours or days) if it helps intuition.

(b) During the coseismic phase, $V \sim 1\ \mathrm{m/s}$. Estimate $|\lambda|$ and the corresponding largest stable step size, this time in microseconds.

(c) Compute the ratio of the two step sizes from (a) and (b). What does this number tell you about trying to simulate an entire earthquake cycle with one fixed h, chosen to be safe everywhere?

Solution.

(a) $|\lambda| \approx V/D_c = 10^{-9}/10^{-5} = 10^{-4}\ \mathrm{s}^{-1}$. Largest stable step: $h \leq 2/10^{-4} = 2 \times 10^4\ \mathrm{s} \approx 5.6$ hours.
(b) $|\lambda| \approx 1/10^{-5} = 10^5\ \mathrm{s}^{-1}$. Largest stable step: $h \leq 2/10^5 = 2 \times 10^{-5}\ \mathrm{s} = 20\ \mu\mathrm{s}$.
(c) Ratio: $\dfrac{2 \times 10^4\ \mathrm{s}}{2 \times 10^{-5}\ \mathrm{s}} = 10^9$. A single fixed step size safe for the coseismic phase is nine orders of magnitude smaller than what the interseismic phase would actually need — using it throughout would take roughly a billion times more steps than necessary during the quiet part of the cycle.

Pitfall to notice: The required step size changes by roughly a factor of 10^9 between the two regimes of a single earthquake cycle. A fixed h small enough to remain stable during the coseismic phase would take on the order of a billion times more steps than necessary to cover the interseismic phase — computationally hopeless for a multi-cycle simulation. This is precisely the gap that adaptive step-size control (Section 6.10), built from an embedded Runge–Kutta pair, is designed to close: take large, cheap steps when $|\lambda|$ is small, and automatically shrink h only when and where the dynamics demand it.
