Numerical solution of ODEs: A practical guide

Fruzsina Agocs

$F_\omega(\alpha + m)!$, 2022

FLATIRON INSTITUTE
Center for Computational Mathematics

Overview

What this talk is about

- Hopefully this talk will help you:
  - Assess whether a numerical solution is satisfactory,
  - Evaluate what accuracy you can reasonably demand,
  - Know what methods exist and where to look for an *implementation* and *further reading*.
- I'll also highlight some powerful methods developed by CCM colleagues
- Check out Alex Barnett's talk from $F_\omega(\alpha + m)!$ 2021 for computational preliminaries (finite-precision arithmetic, convergence/complexity of algorithms, etc.)
- Based on the books Corless and Fillion 2013; Butcher 2016; Press et al. 2007; Hairer et al. 1993a,b and many references therein

Notation and some definitions

- System of ODEs in **standard form** w.l.o.g. :

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) \tag{1}$$

  - Only one independent variable, t, therefore ODE
  - Solve on the solution interval $t \in [t_i, t_f]$
  - $\mathbf{y} : \mathbb{R} \to \mathbb{C}^n$ is *solution vector* with the dependent variables as its n components
  - $\mathbf{f} : \mathbb{R} \times \mathbb{C}^n \to \mathbb{C}^n$ is vector-valued function (RHS)
  - Can always write higher order ODE (containing $\frac{\mathrm{d}^m y}{\mathrm{d}t^n} := y^{(m)}$ with $m > 1$) in standard form: let $\mathbf{y}_i = y^{(i)}$
  - Can always write eq. (1) in *autonomous* form: $\mathbf{f}(t, \mathbf{y}(t)) = \mathbf{f}(\mathbf{y})$, by setting $\mathbf{y}_0 = t$

- Initial (IVP) or boundary value problem (BVP):

$$\mathbf{y}(t_i) = \mathbf{y}_i$$

e.g. $\mathbf{y}_0(t_i) = a, \quad \mathbf{y}_0(t_f) = b \quad$ for $\mathbf{y} = [y, y']$

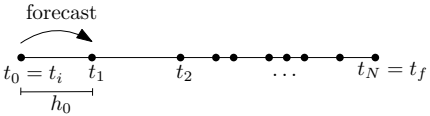
IVP: all conditions specified at one value of t,

BVP: conditions specified at different t-values

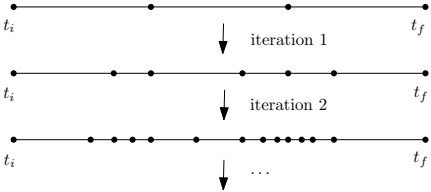
- Approximate numerical solution $\hat{\mathbf{y}}(t)$ and reference solution $\mathbf{y}(t)$. How close is $\hat{\mathbf{y}}(t)$ to $\mathbf{y}(t)$?

IVP v BVP solvers, in a nutshell

- In practice, similar to call and assess, but algorithms qualitatively different



- Timestepping / marching: increment indep variable, use ODE to compute associated increment in dep variables
- Size of steps, $h(t)$, determined by estimate of local error
- Controller adapts stepsize to keep local error beneath tolerance



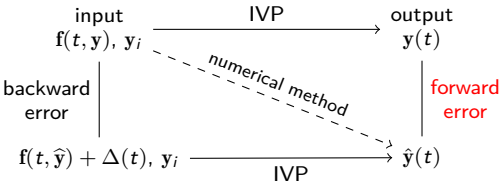
- No preferred direction for independent variable
- Mesh (usually) start off uniform
- Mesh computed simultaneously with the solution, refined iteratively

When to trust a numerical solution?

- Reference solution $\mathbf{y}(t)$ satisfies ODE exactly, $\mathbf{y}'(t) - \mathbf{f}(t, \mathbf{y}(t)) = 0$. **But $\mathbf{y}(t)$ is not known!**
- Instead, we have $\hat{\mathbf{y}}$, for which $\widehat{\mathbf{y}}'(t) - \mathbf{f}(t, \widehat{\mathbf{y}}(t)) = \Delta(t)$
  - $\Delta(t)$ is the *absolute residual*,
  - $\delta_i(t) = \frac{\widehat{\mathbf{y}}_i' - \mathbf{f}_i(t, \widehat{\mathbf{y}})}{\mathbf{f}_i(t, \widehat{\mathbf{y}})}$ is the *relative residual*,
  - We can always compute the residual[1]! It is a way to assess the quality of the solution $\widehat{\mathbf{y}}(t)$.
- $\widehat{\mathbf{y}}(t)$ is an exact solution of the modified, *nearby* problem

$$\widehat{\mathbf{y}}'(t) = \mathbf{f}(t, \widehat{\mathbf{y}}(t)) + \Delta(t)$$

  - How nearby? Controlled by the *tolerance* param supplied to the num method.
  - $\Delta(t)$ is a perturbation of $\mathbf{f}$, which is an input parameter to our IVP
  - Residual is therefore a *backward error*[2]
  - *Forward error* is the difference $\mathbf{y}(t) - \hat{\mathbf{y}}(t)$



[1] It's not free: one eval of $\Delta(t)$ costs one f-evaluation.
[2] Specifically, a perturbation in $\mathbf{f}$. But one may consider perturbing $\mathbf{y}_0$, or both.

Conditioning or sensitivity

- How are the backward and forward errors related? Via the **conditioning of the problem**.
  - Alternatively: how do perturbations in the input affect the output?

Definition 1.

The condition number, κ, expresses the sensitivity of the problem to perturbations in the input parameters.

- Forward error $\leq \kappa \times$ backward error $= \kappa \times \Delta$
- How to find κ? A crude practical estimate (Corless and Fillion 2013):
  1. Solve IVP with two, very different tolerance settings. Get $\hat{\mathbf{y}}_1(t), \hat{\mathbf{y}}_2(t)$. They satisfy $\mathbf{y}_1' = \mathbf{f}(t, \mathbf{y}_1) + \Delta_1$ and $\mathbf{y}_2' = \mathbf{f}(t, \mathbf{y}_2) + \Delta_2$.
  2. Calculate their residuals, Δ_1, Δ_2, and assume (check) $\Delta_1 \gg \Delta_2$.
  3. κ is then *at least* $||\hat{\mathbf{y}}_1(t) - \hat{\mathbf{y}}_2(t)||/||\Delta_1||$.
- Conditioning is a property of the IVP alone.
- The best accuracy you can hope to achieve is $\kappa \times \varepsilon_{\text{machine}}$, independent of the numerical method.

Stability and stiffness

- How can we be sure that the numerical method gives the best achievable accuracy? **By using a (backwards) stable algorithm.**[3]
- But, "stability" in IVP methods is used for a different phenomenon, in relation to *stiffness*.
  - No generally accepted rigorous definition
  - But there exist extremely well-conditioned problems which certain (*explicit*) methods take unexpectedly long to solve, while others (*implicit*) can handle
  - Stiff problems have smooth solutions but explicit methods have to reduce their stepsize, not for accuracy but to maintain "stability".
- Best shown by example: 🐍
- How to tell if your problem is stiff? **By trial and error.** Run a stiff (implicit) solver and see if it mops the floor with a nonstiff (explicit) method.

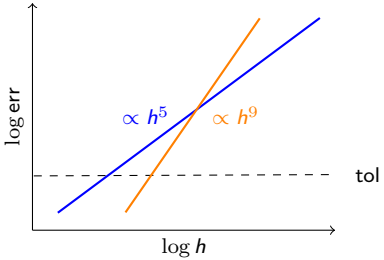
[3] See Alex's slides from last year!

Methods for IVPs: Euler and concepts

- Forecasting step $\mathbf{y}_{k+1} = \Phi(t_k; \mathbf{y}_{k+1}, \mathbf{y}_k, \mathbf{y}_{k-1}, \ldots, \mathbf{y}_0; h; \mathbf{f})$
- If Φ contains $\mathbf{y}_{k+1}$ then *implicit*, otherwise *explicit*
  - Implicit schemes will require solving a system of eqs $\rightarrow$ computational overhead, but robust against stiffness
  - Explicit schemes can generate the forecast $\mathbf{y}_{k+1}$ by iteration, $\rightarrow$ cheap
- Simplest case: (forward) Euler

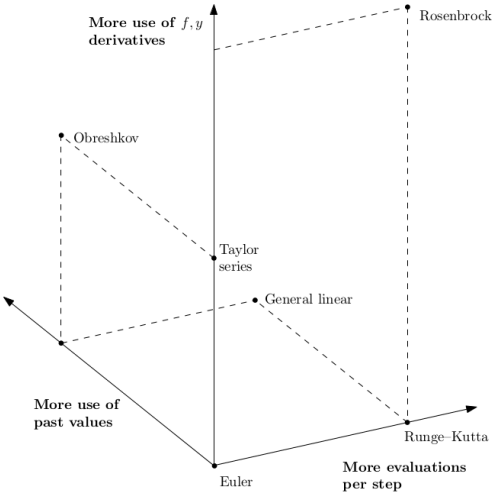
$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(t, \mathbf{y}_k)$$

- Standard analysis in terms of *local error*, err: error accumulated in a single step
  - If $\text{err}_i \sim \mathcal{O}(h^{p+1})$, it's a pth (convergence) order method
  - Euler has $\mathcal{O}(h^2)$ local error (compare with Taylor series) $\rightarrow$ first order
  - Why does order matter? **Determines how much the method needs to reduce its stepsize to match a given tolerance**
  - How to get higher order?



Methods for IVPs: Generalisations of Euler, fixed order schemes

- Common strategy in marching methods: Taylor expand $\mathbf{y}(t)$ around t_k to get $\mathbf{y}(t_k + h)$ up to the pth term
- This requires derivatives: $\mathbf{y}' = \mathbf{f}$, but $\mathbf{y}''$ and above not known
- Various strategies for getting derivative information:
  - **Runge–Kutta**: use intermediate $\mathbf{f}$-evaluations, $\mathbf{f}(t_k + c_i h, \mathbf{y}_i)$ for $i = 1, \ldots s$ for an s-stage method
  - use Jacobian, $J_{ij} = \partial \mathbf{f}_i / \partial \mathbf{y}_j$
  - **Linear multistep**: store and re-use old $\mathbf{y}$ (& $\mathbf{y}'$) evaluations: $\mathbf{y}_{k-1}, \mathbf{y}_{k-2}, \ldots$
- "Grand unified theory": general linear methods, see Butcher 2016



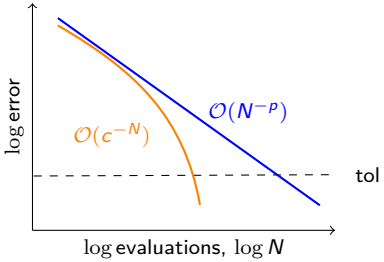
Methods for IVPs: Accelerated low order methods

- What if very high order is needed (and the system is large)? **Extrapolation, deferred correction methods.**
- General strategies:
  - Treat solution as analytic function of stepsize h. Probe at various values of h and extrapolate. $\rightarrow$ Richardson extrapolation (e.g. Bulirsch-Stoer)
  - Take a step with a low-order method. Write down ODE that its error (residual) satisfies, and solve it with the same method. Iterate. $\rightarrow$ Classical deferred correction.
- **Spectral deferred correction**: Dutt et al. 2000
  - *Spectral* accuracy, arbitrarily high order

Definition 2.

A spectral method is one whose convergence rate is as fast as the smoothness of the solution allows.

- Underlying low order method could be explicit or implicit
- Recommended for smooth, stiff, large systems

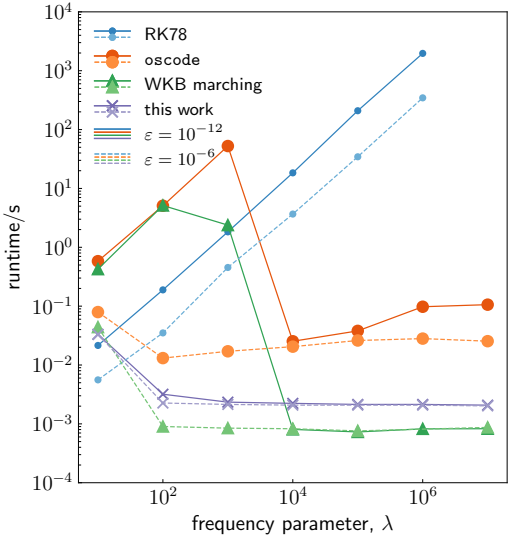


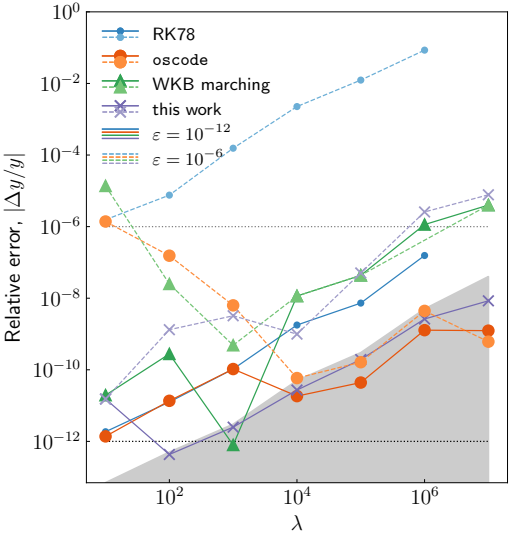
Special IVPs: oscillations /1

- Consider the IVP
$$y''(t) + 2\gamma(t)y'(t) + \omega^2(t)y(t) = 0, \quad t \in [t_i, t_f],$$
with $y(t_i) = y_i$, $y'(t_i) = y_i'$
- Even if $\omega(t)$, $\gamma(t)$ are smooth, *all* standard (read: polynomial-based) methods need $\mathcal{O}(\omega)$ steps/intervals/runtime $\rightarrow$ **slow**
- Methods based on *asymptotic*, non-convergent series may be efficient, offer $\mathcal{O}(1)$ runtime
  - *Asymptotic*: cannot keep adding terms to make series more accurate
  - But usually converges *for a while*. For these methods, $\Delta = \mathcal{O}(\omega^{-k})$, where k is # of terms
- Available methods with code:
  - fixed-order (fixed k) WKB expansion: Agocs et al. 2020 (`oscode`, `Python`, `C++`), Körner et al. 2022 (WKB-marching, `MATLAB`)
  - not asymptotics (but valid only at large ω): Bremer 2018 (phase function method, `Fortran90`)
  - adaptive, spectral asymptotics: Agocs and Barnett 2022 (`Python`)

Special IVPs: oscillations /2

$$y'' + \lambda^2 q(t) y = 0, \quad q(t) = 1 - t^2 \cos(3t), \quad t \in [-1, 1]$$



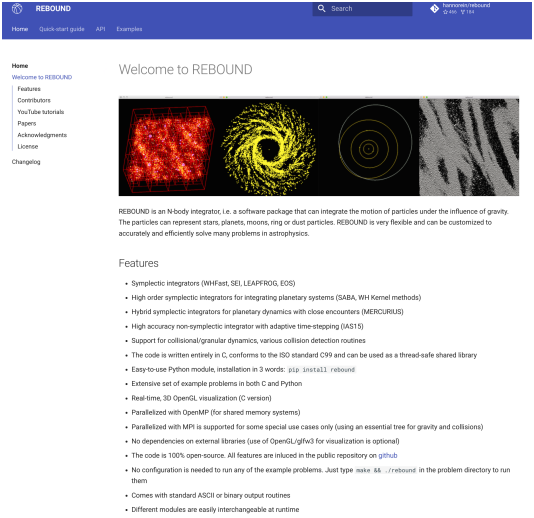


Special IVPs: Hamiltonian systems

- Special classes of problems: need e.g. conserved quantity or time-reversibility
- Common in physics: N-body systems, molecular dynamics, orbital dynamics, mechanical systems
- Objects involved obey Hamiltonian dynamics with $\mathbf{q}, \mathbf{p}$ are position and momentum

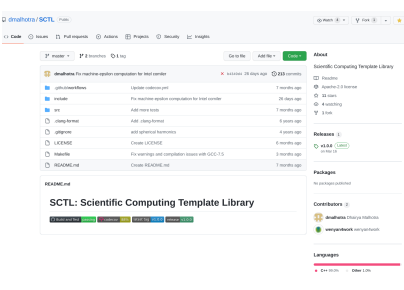
$$\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = -\frac{\partial H(\mathbf{p},\mathbf{q})}{\partial \mathbf{q}}, \quad \frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = \frac{\partial H(\mathbf{p},\mathbf{q})}{\partial \mathbf{p}}.$$

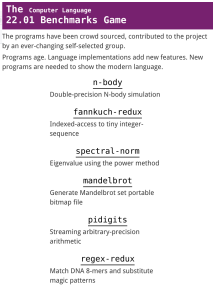
- Philosophy: output of numerical method is satisfactory if it solves a nearby *Hamiltonian* problem
- Methods: leapfrog, symplectic integrators
- Programming tools: REBOUND and references therein



Programming tools

- Scientific Computing Template Library (SCTL)
- Benchmarks game
- Comparison of methods callable from `Julia`
- Comparison of methods from `C++` libraries (`Sundials`, `GSL`, `boost`): 🐍
- Guide to available mathematical software (GAMS)
- `NodePy`





Comparison Of Differential Equation Solver Software

Subject/Item	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Hairer	OSDPACK/Netlib (NAG)	JNCODE	PyODEsol	FATODE	GSL	BOOST	Mathematica	Maple
Language	MATLAB	Python	R	Julia	C++ and Fortran	Fortran	Fortran	Python	Python	Fortran	C	C++	Mathematica	Maple
Selection of Methods for ODEs	Fair	Poor	Fair	Excellent	Good	Fair	Good	Poor	Poor	Fair	Poor	Fair	Fair	Fair
Efficiency*	Poor	Poor****	Poor***	Excellent	Excellent	Good	Good	Good	Good	Good	Fair	Fair	Fair	Good
Tweakability	Fair	Poor	Good	Excellent	Excellent	Good	Good	Fair	Fair	Fair	Fair	Fair	Good	Fair
Event Handling	Good	Good	Fair	Excellent	Good**	None	Good**	None	Fair	None	None	None	Good	Good
Symbolic Calculation of Jacobians and Autodifferentiation	None	None	None	Excellent	None	None	None	None	None	None	None	None	Excellent	Excellent
Complex Numbers	Excellent	Good	Fair	Good	None	None	None	None	None	None	None	Good	Excellent	Excellent
Arbitrary Precision Numbers	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	Excellent	Excellent
Control Over Linear/Nonlinear Solvers	None	Poor	None	Excellent	Excellent	Good	Depends on the solver	None	None	None	None	None	Fair	None
Built-in Parallelism	None	None	None	Excellent	Excellent	None	None	None	None	None	None	Fair	None	None
Differential-Algebraic Equation (DAE) Solvers	Good	None	Good	Excellent	Good	Excellent	Good	None	Fair	Good	None	None	Good	Good
Implicitly-Defined DAE Solvers	Good	None	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	None	Good	None
Constant-Lag Delay-Differential Equation (DDE) Solvers	Fair	None	Poor	Excellent	None	Good	Fair (via DDVERK)	Fair	None	None	None	None	Good	Excellent
State-Dependent DDE Solvers	Poor	None	Poor	Excellent	None	Excellent	Good	None	None	None	None	None	None	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	None	None	Excellent	None	None	None	Good	None	None	None	None	Fair	Poor

Methods for BVPs

- Beware: $\kappa_{\text{BVP}} \neq \kappa_{\text{IVP}}$!
- κ_{BVP} can be computed, see Corless and Fillion 2013.
- General ideas:
  - **Shooting:**
    1. Solve associated IVP instead from t_i from a guess of i.c. Compute solution at t_f.
    2. Embed in a root finding process until solution satisfies boundary conditions at t_f.
  - **Collocation:**
    1. Approximate the solution in a finite-dimensional space (e.g. space of some polynomials up to a given degree).
    2. Require that the ODE is satisfied exactly at a finite number of points (nodes). This gives a set of conditions.
    3. Solve to give the finite-dimensional representation of the solution.
    4. Compute residual; iterate (over dimensions of solution space) until satisfactory.
  - Tricky part: choosing the right *mesh* of nodes (e.g. Equispaced v Chebyshev) and refining the mesh.
- Example: Chebyshev spectral method[4]
  - Codes: `MATLAB`, `Python`, `Julia`: Chebfun, Chebop, Dedalus, Approxfun
  - Reading: Boyd 2001; Trefethen 2000, 2019

[4] See Dan Fortunato's talk from $F_\omega(\alpha + m)$! 2021.

References I

- R. M. Corless and N. Fillion (2013). "A graduate introduction to numerical methods". In: *AMC* 10, p. 12.
- J. C. Butcher (2016). *Numerical methods for ordinary differential equations*. John Wiley & Sons.
- W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.
- E. Hairer, S. P. Nørsett, and G. Wanner (1993a). *Solving Ordinary Differential Equations I*. Vol. 1. Springer Series in Computational Mathematics. Springer Berlin Heidelberg. ISBN: 978-3-642-08158-3.
- — (1993b). *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer Series in Computational Mathematics. Springer. ISBN: 9783540604525.
- A. Dutt, L. Greengard, and V. Rokhlin (2000). "Spectral deferred correction methods for ordinary differential equations". In: *BIT Numerical Mathematics* 40.2, pp. 241–266.
- F. J. Agocs, W. J. Handley, A. N. Lasenby, and M. P. Hobson (2020). "Efficient method for solving highly oscillatory ordinary differential equations with applications to physical systems". In: *Physical Review Research* 2.1, 013030 🐦.
- J. Körner, A. Arnold, and K. Döpfner (2022). "WKB-based scheme with adaptive step size control for the Schrödinger equation in the highly oscillatory regime". In: *Journal of Computational and Applied Mathematics* 404, p. 113905.
- J. Bremer (2018). "On the numerical solution of second order ordinary differential equations in the high-frequency regime". In: *Applied and Computational Harmonic Analysis* 44.2, pp. 312–349.

References II

F. J. Agocs and A. H. Barnett (2022). "An adaptive spectral method for oscillatory second-order linear ODEs with frequency-independent cost". In: *in prep.*

J. P. Boyd (2001). *Chebyshev and Fourier spectral methods*. Courier Corporation.

L. N. Trefethen (2000). *Spectral methods in MATLAB*. SIAM.

— (2019). *Approximation Theory and Approximation Practice, Extended Edition*. SIAM.