Scientific Computing: Ordinary Differential Equations

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

[1]Course MATH-GA.2043 or CSCI-GA.2112, Fall 2015

Nov 19th, 2015

Outline

1. Initial Value Problems
2. Numerical Methods for ODEs
3. Higher-Order Methods
4. Conclusions

Initial Value Problems

- We want to numerically approximate the solution to the **ordinary differential equation**

$$\frac{dx}{dt} = x'(t) = \dot{x}(t) = f\left[x(t), t\right],$$

with **initial condition** $x(t = 0) = x(0) = x_0$.

- This means that we want to generate an approximation to the **trajectory** $x(t)$, for example, a sequence $x(t_k = k\Delta t)$ for $k = 1, 2, \ldots, N = T/\Delta t$, where Δt is the **time step** used to discretize time.
- If f is independent of t we call the system **autonomous**.
- Note that second-order equations can be written as a **system** of first-order equations:

$$\frac{d^2x}{dt^2} = \ddot{x}(t) = f\left[x(t), t\right] \quad \equiv \quad \begin{cases} \dot{x}(t) = & v(t) \\ \dot{v}(t) = & f\left[x(t), t\right] \end{cases}$$

Relation to Numerical Integration

- If f is independent of x then the problem is equivalent to numerical integration
$$x(t) = x_0 + \int_0^t f(s)ds.$$
- More generally, we cannot compute the integral because it depends on the unknown answer $x(t)$:
$$x(t) = x_0 + \int_0^t f\left[x(s), s\right] ds.$$
- Numerical methods are based on approximations of $f\left[x(s), s\right]$ into the "future" based on knowledge of $x(t)$ in the "past" and "present".

Convergence

- Consider a trajectory numerically discretized as a sequence that **approximates** the exact solution at a **discrete** set of points:

$$x^{(k)} \approx x(t_k = k\Delta t), \quad k = 1, \ldots, T/\Delta t.$$

- A method is said to **converge with order** $p > 0$, or to have **order of accuracy** p, if for any finite T for which the ODE has a solution,

$$\left| x^{(k)} - x(k\Delta t) \right| = O(\Delta t^p) \text{ for all } 0 \leq k \leq T/\Delta t.$$

- All methods are recursions that compute a new $x^{(k+1)}$ from previous $x^{(k)}$ by evaluating $f(x)$ several times. For example, **one-step methods** have the form

$$x^{(k+1)} = G\left(x^{(k)}; f\right).$$

Consistency

- The **local trunction error** of a method is the amount by which the exact solution does not satisfy the numerical scheme:

$$e_k = x\left[(k+1)\,\Delta t\right] - G\left[x(k\Delta t); f\right]$$

- A method is **consistent** if the local truncation error vanishes as $\Delta t \to 0$.
- A method is **consistent with order** $q > 1$ if $|e_k| = O(\Delta t^q)$.
- The **global truncation error** is the sum of the local truncations from each time step.
- Note that the local truncation order must be at least 1, since if one makes an error $O(\Delta t^q)$ at each time step, the global error after $T/\Delta t$ time steps can become on the order of

$$\left| x^{(k)} - x(k\Delta t) \right| = O(\Delta t^q \cdot \frac{T}{\Delta t}) = O(\Delta t^{q-1}) = O(\Delta t^p),$$

and we must have $p > 0$ for convergence.

Zero Stability

- It turns out consistency is not sufficient for convergence: One must also examine how perturbations grow with time: **error propagation**.
- A method is called **zero-stable** if for all sufficiently small but finite Δt, introducing perturbations at each step (e.g., roundoff errors, errors in evaluating f) with magnitude less than some small ϵ perturbs the solution by at most $O(\epsilon)$.
- This simply means that errors do not increase but rather decrease from step to step, as we saw with roundoff errors in the first homework.
- A central theorem in numerical methods for differential equations is the **Lax equivalence theorem**:
  Any consistent method is convergent if and only if it is zero-stable, or
  $$\text{consistency} + \text{stability} = \text{convergence}.$$
- One-step methods can be shown to be zero-stable if f is well-behaved (Lipschitz continuous with respect to its second argument).

Euler's Method

- Assume that we have our approximation $x^{(k)}$ and want to move by one time step:

$$x^{(k+1)} \approx x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds.$$

- The simplest possible thing is to use a piecewise constant approximation:

$$f\left[x(s), s\right] \approx f(x^{(k)}) = f^{(k)},$$

  which gives the **forward Euler method**

$$x^{(k+1)} = x^{(k)} + f^{(k)} \Delta t.$$

- This method requires only one function evaluation per time step.

Euler's Method

$$\text{Scheme: } x^{(k+1)} - x^{(k)} - f^{(k)}\Delta t = 0$$

- The local trunction error is easy to find using a Taylor series expansion:

$$e_k = x\left[(k+1)\,\Delta t\right] - x\left(k\Delta t\right) - f\left[x\left(k\Delta t\right)\right]\Delta t =$$
$$\tilde{x}^{(k+1)} - \tilde{x}^{k} - f\left(\tilde{x}^{k}\right)\Delta t = \tilde{x}^{(k+1)} - \tilde{x}^{k} - \left[x'\left(k\Delta t\right)\right]\Delta t = \frac{x''(\xi)}{2}\Delta t^2,$$

  for some $k\Delta t \leq \xi \leq (k+1)\,\Delta t$.
- Therefore the order of the local truncation error is $O(\Delta t^2)$.
- The global truncation error, however, is of order $O(\Delta t)$, so this is a **first-order accurate** method.

Long-Time Stability

- Consider the model problem for $\lambda < 0$:

$$x'(t) = \lambda x(t)$$
$$x(0) = 1,$$

with an exact solution that **decays exponentially**, $x(t) = e^{\lambda t}$.

- Applying Euler's method to this model equation gives:

$$x^{(k+1)} = x^{(k)} + \lambda x^{(k)} \Delta t = (1 + \lambda \Delta t)\, x^{(k)} \quad \Rightarrow$$

$$x^{(k)} = (1 + \lambda \Delta t)^k$$

- The numerical solution will **decay** if the time step satisfies the **stability criterion**

$$|1 + \lambda \Delta t| \leq 1 \quad \Rightarrow \quad \Delta t < -\frac{2}{\lambda}.$$

- Otherwise, the numerical solution will blow up over a sufficiently long period!

Global Error

- Now assume that the stability criterion is satisfied, and see what the error is at time T:

$$x^{(k)} - e^{\lambda T} = (1 + \lambda \Delta t)^{T/\Delta t} - e^{\lambda T} =$$
$$= \left(1 + \frac{\lambda T}{N}\right)^{N} - e^{\lambda T}.$$

- In the limit $N \to 0$ the first term converges to $e^{\lambda T}$ so the error is zero (the method converges).
- Furthermore, the correction terms are:

$$\left(1 + \frac{\lambda T}{N}\right)^{N} = e^{\lambda T}\left[1 - \frac{(\lambda T)^2}{2N} + O(N^{-2})\right]$$
$$= e^{\lambda T}\left[1 - \frac{\lambda^2 T}{2}\Delta t + O(\Delta t^2)\right],$$

  which now shows that the relative error is $O(\Delta t)$ but generally grows with T.

Absolute Stability

- A method is called **absolutely stable** if for $\lambda < 0$ the numerical solution decays to zero, like the actual solution.
- The above analysis shows that Euler's method is **conditionally stable**, meaning it is stable if $\Delta t < 2/\left|\lambda\right|$.
- One can make the analysis more general by allowing λ to be a **complex number**. This is particularly useful when studying stability in numerical methods for PDEs...
- The theoretical solution decays if λ has a **negative real part**, $\mathrm{Re}(\lambda) < 0$.
- We call the **region of absolute stability** the set of complex numbers

$$z = \lambda \Delta t$$

for which the numerical solution decays to zero.

A-stable Methods

- For Euler's method, the stability condition is

$$|1 + \lambda \Delta t| = |1 + z| = |z - (-1)| \leq 1 \quad \Rightarrow$$

  which means that z must be in a unit disk in the complex plane centered at $(-1, 0)$:

$$z \in \mathcal{C}_1(-1, 0).$$

- An **A-stable** or **unconditionally stable method** is one that is stable for any choice of time-step if $\mathrm{Re}(\lambda) < 0$.
- It is not trivial to come up with methods that are A-stable but also as simple and efficient as the Euler method, but it is necessary in many practical situations.

Stiff Equations

- For a real "non-linear" problem, $x'(t) = f\left[x(t), t\right]$, the role of λ is played by
$$\lambda \longleftrightarrow \frac{\partial f}{\partial x}.$$
- Consider the following model equation:
$$x'(t) = \lambda\left[x(t) - g(t)\right] + g'(t),$$
where $g(t)$ is a nice (regular) function evolving on a time scale of order 1, and $\lambda \ll -1$ is a large negative number.
- The exact solution consists of a fast-decaying "irrelevant" component and a slowly-evolving "relevant" component:
$$x(t) = \left[x(0) - g(0)\right] e^{\lambda t} + g(t).$$
- Using Euler's method requires a time step $\Delta t < 2/\left|\lambda\right| \ll 1$, i.e., many time steps in order to see the relevant component of the solution.

Stiff Systems

- An ODE or a system of ODEs is called **stiff** if the solution evolves on widely-separated timescales and the fast time scale decays (dies out) quickly.
- We can make this precise for linear systems of ODEs, $\mathbf{x}(t) \in \mathbb{R}^n$:

$$\mathbf{x}'(t) = \mathbf{A}\left[\mathbf{x}(t)\right].$$

- Assume that $\mathbf{A}$ has an eigenvalue decomposition:

$$\mathbf{A} = \mathbf{X}\boldsymbol{\Lambda}\mathbf{X}^{-1},$$

and express $\mathbf{x}(t)$ in the basis formed by the eigenvectors $\mathbf{x}_i$:

$$\mathbf{y}(t) = \mathbf{X}^{-1}\left[\mathbf{x}(t)\right].$$

contd.

$$\mathbf{x}'(t) = \mathbf{A}\left[\mathbf{x}(t)\right] = \mathbf{X}\boldsymbol{\Lambda}\left[\mathbf{X}^{-1}\mathbf{x}(t)\right] = \mathbf{X}\boldsymbol{\Lambda}\left[\mathbf{y}(t)\right] \quad \Rightarrow$$

$$\mathbf{y}'(t) = \boldsymbol{\Lambda}\left[\mathbf{y}(t)\right]$$

- The different y variables are now **uncoupled**: each of the n ODEs is independent of the others:

$$y_i = y_i(0)e^{\lambda_i t}.$$

- Assume now that all eigenvalues are negative, $\boldsymbol{\lambda} < 0$, so each component of the solution decays:

$$\mathbf{x}(t) = \sum_{i=1}^{n} y_i(0)e^{\lambda_i t}\mathbf{x}_i \quad \rightarrow \quad 0 \text{ as } t \rightarrow \infty.$$

Stiffness

- If we solve the original system using Euler's method,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{A}\mathbf{x}^{(k)}\Delta t,$$

  the time step must be smaller than the smallest stability limit,

$$\Delta t < \frac{2}{\max_i |\mathrm{Re}(\lambda_i)|}.$$

- A system is stiff if there is a strong **separation of time scales** in the eigenvalues:

$$r = \frac{\max_i |\mathrm{Re}(\lambda_i)|}{\min_i |\mathrm{Re}(\lambda_i)|} \gg 1.$$

- For non-linear problems **A** is replaced by the Jacobian $\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{f}(\mathbf{x}, t)$.

Backward Euler

$$x^{(k+1)} \approx x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds.$$

- How about we use a piecewise constant-approximation, but based on the end-point:

$$f\left[x(s), s\right] \approx f(x^{(k+1)}) = f^{(k+1)},$$

  which gives the **backward Euler method**

$$x^{(k+1)} = x^{(k)} + f(x^{(k+1)})\Delta t.$$

- This method requires **solving a non-linear equation** at every time step.

Unconditional Stability

- Backward Euler is an **implicit method**, as opposed to an **explicit method** like the forward Euler method.
- The local and global truncation errors are basically the same as in the forward Euler method.
- But, let us examine the stability for the model equation $x'(t) = \lambda x(t)$:

$$x^{(k+1)} = x^{(k)} + \lambda x^{(k+1)} \Delta t \quad \Rightarrow \quad x^{(k+1)} = x^{(k)} / (1 - \lambda \Delta t)$$

$$x^{(k)} = x^{(0)} / (1 - \lambda \Delta t)^k$$

- This implicit method is thus **unconditionally stable**, since for any time step

$$|1 - \lambda \Delta t| > 1.$$

Implicit Methods

- This is a somewhat generic conclusion:
  Implicit methods are generally **more stable** than explicit methods, and solving stiff problems generally requires using an implicit method.
- The price to pay is solving a system of non-linear equations at every time step (linear if the ODE is linear):
  This is best done using **Newton-Raphson**'s method, where the solution at the previous time step is used as an initial guess.
- Trying to by-pass Newton's method and using a technique that looks like an explicit method (e.g., fixed-point iteration) will not work:
  One most **solve linear systems** in order to avoid stability restrictions.
- For PDEs, the linear systems become large and implicit methods can become very expensive...

Multistep Methods

$$x^{(k+1)} \approx x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds.$$

- Euler's method was based on a piecewise constant approximation (extrapolation) of $f(s) \equiv f\left[x(s), s\right]$.
- If we instead integrate the linear extrapolation

$$f(s) \approx f\left(x^{(k)},\, t^{(k)}\right) + \frac{f\left(x^{(k)},\, t^{(k)}\right) - f\left(x^{(k-1)},\, t^{(k-1)}\right)}{\Delta t}(s - t_k),$$

we get the second-order **two-step Adams-Bashforth** method

$$x^{(k+1)} = x^{(k)} + \frac{\Delta t}{2}\left[3f\left(x^{(k)},\, t^{(k)}\right) - f\left(x^{(k-1)},\, t^{(k-1)}\right)\right].$$

- This is an example of a **multi-step method**, which requires keeping previous values of f.

Runge-Kutta Methods

- Runge-Kutta methods are a powerful class of **one-step methods** similar to Euler's method, but more accurate.
- As an example, consider using a trapezoidal rule to approximate the integral

$$x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds \approx x^{(k)} + \frac{\Delta t}{2}\left[f\left(k\Delta t\right) + f\left(\left(k+1\right)\Delta t\right)\right],$$

$$x^{(k+1)} = x^{(k)} + \frac{\Delta t}{2}\left[f\left(x^{(k)},\ t^{(k)}\right) + f\left(x^{(k+1)},\ t^{(k+1)}\right)\right]$$

  which requires solving a nonlinear equation for $x^{(k+1)}$.
- This is the simplest **implicit Runge-Kutta method**, usually called the **trapezoidal method** or the **Crank-Nicolson method**.
- The local truncation error is $O(\Delta t^3)$, so the global error is **second-order accurate** $O(\Delta t^2)$, and the method is **unconditionally stable**.

Explicit Runge-Kutta Methods

$$x^{(k+1)} = x^{(k)} + \frac{\Delta t}{2}\left[f\left(x^{(k)},\ t^{(k)}\right) + f\left(x^{(k+1)},\ t^{(k+1)}\right)\right]$$

- In an explicit method, we would approximate $x^{\star} \approx x^{(k+1)}$ first using Euler's method, to get the simplest **explicit Runge-Kutta method**, usually called **Heun's method**

$$\begin{aligned} x^{\star} =& x^{(k)} + f\left(x^{(k)},\ t^{(k)}\right)\Delta t \\ x^{(k+1)} =& x^{(k)} + \frac{\Delta t}{2}\left[f\left(x^{(k)},\ t^{(k)}\right) + f\left(x^{\star},\ t^{(k+1)}\right)\right]. \end{aligned}$$

- This is still second-order accurate, but, being explicit, is **conditionally-stable**, with the same time step restriction as Euler's method.
- This is a representative of a powerful class of second-order methods called **predictor-corrector methods**:
  Euler's method is the predictor, and then trapezoidal method is the corrector.

Higher-Order Runge Kutta Methods

- The idea is to evaluate the function $f(x,t)$ several times and then take a time-step based on an average of the values.
- In practice, this is done by performing the calculation in **stages**: Calculate an intermediate approximation $x^{\star}$, evaluate $f(x^{\star})$, and go to the next stage.
- The most celebrated Runge-Kutta methods is a **four-stage** fourth-order accurate RK4 method based on Simpson's approximation to the integral:

$$x^{(k)} + \int_{k\Delta t}^{(k+1)\Delta t} f\left[x(s), s\right] ds \approx$$

$$x^{(k)} + \frac{\Delta t}{6}\left[f(x^{(k)}) + 4f(x^{(k+1/2)}) + f(x^{(k+1)})\right] =$$

$$x^{(k)} + \frac{\Delta t}{6}\left[f^{(k)} + 4f^{(k+1/2)} + f^{(k+1)}\right],$$

and we approximate $4f^{(k+1/2)} = 2f^{(k+1/2;1)} + 2f^{(k+1/2;2)}$.

RK4 Method

$$f^{(k)} = f\left(x^{(k)}\right), \quad x^{(k+1/2;1)}, = x^{(k)} + \frac{\Delta t}{2} f^{(k)}$$

$$f^{(k+1/2;1)} = f\left(x^{(k+1/2;1)},\ t^{(k)} + \Delta t/2\right)$$
$$x^{(k+1/2;2)} = x^{(k)} + \frac{\Delta t}{2} f^{(k+1/2;1)}$$

$$f^{(k+1/2;2)} = f\left(x^{(k+1/2;2)},\ t^{(k)} + \Delta t/2\right)$$
$$x^{(k+1;1)} = x^{(k)} + \Delta t\, f^{(k+1/2;2)}$$

$$f^{(k+1)} = f\left(x^{(k+1;1)},\ t^{(k)} + \Delta t\right)$$
$$x^{(k+1)} = x^{(k)} + \frac{\Delta t}{6}\left[f^{(k)} + 2f^{(k+1/2;1)} + 2f^{(k+1/2;2)} + f^{(k+1)}\right]$$

Adaptive Methods

- For many problems of interest the character of the problem changes with time, and it is not appropriate to use the same time step throughout.
- An **adaptive method** would adjust the time step to satisfy the stability criterion, for example

$$\Delta t_n < 2\alpha \left(\frac{\partial f}{\partial x} \right)_n, \text{ where } \alpha < 1,$$

  but it would also need to ensure some accuracy.
- Robust adaptive methods are usually based on Runge-Kutta methods: They increase or decrease Δt_k from step to step as deemed best based on error estimates.
- For example, a famous **RK45 method** cleverly combines a fifth stage with the prior four stages in order to estimate the error, similarly to what we did for adaptive integration (see notes by Goodman, for example).

Which Method is Best?

- As expected, there is no universally "best" method for integrating ordinary differential equations: It depends on the problem:
  - How stiff is your problem (may demand implicit method), and does this change with time?
  - How many variables are there, and how long do you need to integrate for?
  - How accurately do you need the solution, and how sensitive is the solution to perturbations (chaos).
  - How well-behaved or not is the function $f(x, t)$ (e.g., sharp jumps or discontinuities, large derivatives, etc.).
  - How costly is the function $f(x, t)$ and its derivatives (Jacobian) to evaluate.
  - Is this really ODEs or a something coming from a PDE integration (next lecture)?

In MATLAB

- In MATLAB, there are several functions whose names begin with
$$[\mathbf{t}, \mathbf{x}] = ode(f,\ [t_0, t_e],\ x_0,\ odeset(\dots)).$$
- *ode*23 is a second-order adaptive explicit Runge-Kutta method, while *ode*45 is a fourth-order version (try it first).
- *ode*23*tb* is a second-order implicit RK method.
- *ode*113 is a variable-order explicit multi-step method that can provide very high accuracy.
- *ode*15*s* is a variable-order implicit multi-step method.
- For implicit methods the Jacobian can be provided using the *odeset* routine.

Non-Stiff example

```
function dy = rigid(t,y) % File rigid.m
dy = zeros(3,1);     % a column vector
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
%-----------

options = odeset('RelTol',1e-3, 'AbsTol',[1e-4 1e-4 1e-5]);
[T,Y] = ode45(@rigid, [0 12], [0 1 1], options);

plot(T,Y(:,1),'o--r', T,Y(:,2),'s--b', T,Y(:,3),'d--g');
xlabel('t'); ylabel('y'); title('RelTol=1e-3');
```

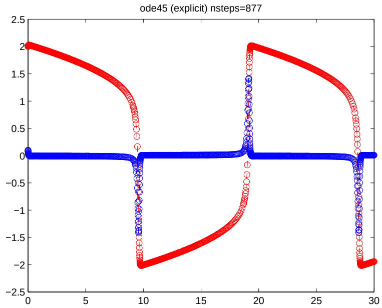
Stiff example

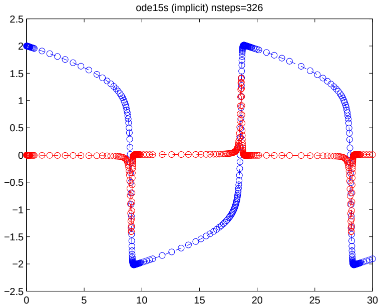
```
r=10; % Try r=100
f = @(t,y) [y(2); r*(1 - y(1)^2)*y(2) - y(1)];

figure(2); clf
[T,Y] = ode45(f,[0 3*r],[2 1]);
plot(T,Y(:,1),'o--r', T,Y(:,2)/r,'o--b')
title(['ode45␣(explicit)␣nsteps=', int2str(size(T,1))]);

figure(3); clf
[T,Y] = ode15s(f,[0 3*r],[2 0]);
plot(T,Y(:,1),'o--b', T,Y(:,2)/r,'o--r')
title(['ode15s␣(implicit)␣nsteps=', int2str(size(T,1))]);
```

Stiff van der Pol system ($r = 10$)





Conclusions/Summary

- Time stepping methods for ODEs are **convergent if and only if they are consistent and stable**.
- We distinguish methods based on their **order of accuracy** and on whether they are **explicit** (forward Euler, Heun, RK4, Adams-Bashforth), or **implicit** (backward Euler, Crank-Nicolson), and whether they are **adaptive**.
- **Runge-Kutta methods** require more evaluations of f but are more robust, especially if adaptive (e.g., they can deal with sharp changes in f). Generally the recommended first-try ($ode45$ or $ode23$ in MATLAB).
- **Multi-step methods** offer high-order accuracy and require few evaluations of f per time step. They are not very robust however. Recommended for well-behaved non-stiff problems ($ode113$).
- For **stiff problems** an **implicit method** is necessary, and it requires solving (linear or nonlinear) systems of equations, which may be complicated (evaluating Jacobian matrices) or costly ($ode15s$).