CS 450: Numerical Anlaysis[1]

Initial Value Problems for Ordinary Differential Equations

University of Illinois at Urbana-Champaign

[1] *These slides have been drafted by Edgar Solomonik as lecture templates and supplementary material for the book "Scientific Computing: An Introductory Survey" by Michael T. Heath (slides).*

Ordinary Differential Equations

- An *ordinary differential equation (ODE)* usually describes time-varying system by a function $\boldsymbol{y}(t)$ that satisfies a set of equations in its derivatives. *The general implicit form is*

$$\boldsymbol{g}(t, \boldsymbol{y}, \boldsymbol{y}', \boldsymbol{y}'', \ldots, \boldsymbol{y}^{(k)}) = \boldsymbol{0},$$

but we restrict focus on the explicit form, $\boldsymbol{y}^{(k)} = \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{y}', \boldsymbol{y}'', \ldots, \boldsymbol{y}^{(k-1)})$.

- An ODE of any *order* k can be transformed into a first-order ODE,

$$\boldsymbol{u}' = \begin{bmatrix} \boldsymbol{u}_1' \\ \vdots \\ \boldsymbol{u}_{k-1}' \\ \boldsymbol{u}_k' \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_k \\ \boldsymbol{f}(t, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_k) \end{bmatrix} \quad \textit{where} \quad \boldsymbol{u}_i(t) = \boldsymbol{y}^{(i-1)}(t).$$

Consequently, we restrict our focus to systems of first-order ODEs. Of particular importance are linear ODEs, which have the form $\boldsymbol{y}' = \boldsymbol{A}(t)\boldsymbol{y}$, *whose coefficients are said to be constant if* $\boldsymbol{A}(t) = \boldsymbol{A}$ *for all* t.

Example: Newton's Second Law

- $F = ma$ corresponds to a second order ODE,

$$F = my''(t),$$
$$y''(t) = F/m.$$

- We can transform it into a first order ODE in two variables:

$$\boldsymbol{u} = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix},$$

$$\begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \boldsymbol{u}' = \boldsymbol{f}(t, \boldsymbol{u}) = \begin{bmatrix} u_2 \\ F/m \end{bmatrix}.$$

Initial Value Problems

- Generally, a first order ODE specifies only the derivative, so the solutions are non-unique. An *initial condition* addresses this:

$$\boldsymbol{y}(t_0) = \boldsymbol{y}_0$$

This condition yields an initial value problem (IVP), which is the simplest example of a boundary condition.

- Given an initial condition, an ODE must satisfy an integral equation for any given point t:

$$\boldsymbol{y}(t) = \boldsymbol{y}_0 + \int_{t_0}^{t} \boldsymbol{f}(s, \boldsymbol{y}(s))ds,$$

In the special case that $\boldsymbol{y}' = \boldsymbol{f}(t)$, the integral can be computed directly by numerical quadrature to solve the ODE.

Existence and Uniqueness of Solutions

- For an ODE to have a unique solution, it must be defined on a closed domain D and be *Lipschitz continuous*:

$$\forall \boldsymbol{y}, \hat{\boldsymbol{y}} \in D, \quad ||\boldsymbol{f}(t, \hat{\boldsymbol{y}}) - \boldsymbol{f}(t, \boldsymbol{y})|| \leq L||\hat{\boldsymbol{y}} - \boldsymbol{y}||,$$

i.e. the rate of change of the ODE should itself change continuously. Any differentiable function $\boldsymbol{f}$ is Lipschitz continuous with

$$L = \max_{(t,\boldsymbol{y}) \in D} ||\boldsymbol{J}_{\boldsymbol{f}}(t, \boldsymbol{y})||,$$

where $\boldsymbol{J}_{\boldsymbol{f}}$ is Jacobian of $\boldsymbol{f}$ with respect to $\boldsymbol{y}$.

- The solutions of an ODE can be stable, unstable, or asymptotically stable:
  Perturbation to the input causes a perturbation to the solution that
  - *has bounded growth for a stable ODE,*
  - *unbounded growth for an unstable ODE, and*
  - *shrinks for an asymptotically stable ODE.*

Stability of 1D ODEs

- The solution to the scalar ODE $y' = \lambda y$ is $y(t) = y_0 e^{\lambda t}$, with stability dependent on λ:

$$\lim_{t\to\infty} y(t) = \begin{cases} \infty & : \lambda > 0 \textit{ (unstable)} \\ y_0 & : \lambda = 0 \textit{ (stable)} \\ 0 & : \lambda < 0 \textit{ (asymptotically stable)} \end{cases}$$

- A constant-coefficient linear ODE has the form $\boldsymbol{y}' = \boldsymbol{A}\boldsymbol{y}$, with stability dependent on the real parts of the eigenvalues of $\boldsymbol{A}$:
  - *At a point $(t, \boldsymbol{y})$, any ODE can be approximated by a linear ODE of the form $\boldsymbol{z}' = \boldsymbol{J_f}(t, \boldsymbol{y})\boldsymbol{z}$.*
  - *For general ODEs, stability can be ascertained locally by considering the eigenvalues of $\boldsymbol{J_f}(t, \boldsymbol{y})$.*

Numerical Solutions to ODEs

Demo: Forward Euler stability

- Methods for numerical ODEs seek to approximate $\boldsymbol{y}(t)$ at $\{t_k\}_{k=1}^m$.

  Compute $\boldsymbol{y}_k$ for $k \in \{1, \ldots, m\}$ so as to approximate $\boldsymbol{y}(t_k) \approx \boldsymbol{y}_k$. For an IVP, typically form $\boldsymbol{y}_{k+1}$ using $\boldsymbol{y}_k$ or additionally (for multistep methods) $\boldsymbol{y}_{k-1}, \ldots$

- Euler's method provides the simplest method (attempt) for obtaining a numerical solution:

  Approximation solution to ODE at $t_k + h$ by linear segment from $(t_k, \boldsymbol{y}_k)$ with slope $\boldsymbol{f}(t_k, \boldsymbol{y}_k)$,

  $$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h_k \boldsymbol{f}(t_k, \boldsymbol{y}_k).$$

  This approximation is the first order form of various models (Taylor series, finite differences, interpolation, quadrature, undetermined coefficients).

Error in Numerical Methods for ODEs

- Truncation error is typically the main quantity of interest, which can be defined *globally* or *locally*:
  - *Global error is measured at all points*

$$\boldsymbol{e}_k = \boldsymbol{y}_k - \boldsymbol{y}(t_k).$$

  - *Local error measures the deviation from the exact solution $\boldsymbol{u}_{k-1}(t)$ passing through the previous point $(t_{k-1}, \boldsymbol{y}_{k-1})$,*

$$\boldsymbol{l}_k = \boldsymbol{y}_k - \boldsymbol{u}_{k-1}(t_k).$$

- The *order of accuracy* of a given method is one less than than the order of the leading order term in the local error $\boldsymbol{l}_k$:
  - *Accuracy is of order p if $\boldsymbol{l}_k = O(h_k^{p+1})$, for forward Euler $p = 1$ since*

$$\boldsymbol{y}(t_{k+1}) = \boldsymbol{y}(t_k) + h_k \boldsymbol{f}(t_k, \boldsymbol{y}(t_k)) + O(h_k^2),$$

  so $\boldsymbol{l}_k = O(h_k^2)$.

Accuracy and Taylor Series Methods

- By taking a degree-r Taylor expansion of the ODE in t, at each consecutive $(t_k, \boldsymbol{y}_k)$, we achieve rth order accuracy.

  We can bound the local approximation error as the error in the Taylor expansion,

  $$\boldsymbol{y}(t_k + h) = \boldsymbol{y}(t_k) + \boldsymbol{y}'(t_k)h + \cdots + \boldsymbol{y}^{(r)}(t_k)h^{r-1}/r!$$

  which is $O(h^{r+1})$, leading to $O(h^r)$ accuracy in the approximation to $\boldsymbol{f}(t, \boldsymbol{y})$. Euler's method is a first-order Taylor series method.

- Taylor series methods require high-order derivatives at each step:
  - *Analytic differentiation is expensive, so seek to approximate.*
  - *Can perform a finite-differencing approximation by evaluating at points near $(t_k, \boldsymbol{y}_k)$ (multi-stage methods) or simply using previous points, e.g. $(t_{k-1}, \boldsymbol{y}_{k-1})$ (multi-step methods).*

Growth Factors and Stability Regions

Demo: *Stability regions*
Demo: *Stiffness*

- Stability of an ODE method discerns whether local errors are amplified, deamplified, or stay constant:
  - *A method is stable if the growth factor of the error is less than or equal to one.*
  - *The stability region for a method describes the step-size conditions necessary for stability in terms of*
    - *the step size h (assuming its constant) and*
    - *eigenvalues λ, usually as a function of $h\lambda$.*
- Basic stability properties follow from analysis of linear scalar ODE, which serves as a local approximation to more complex ODEs.
  - *Consider forward Euler for the ODE $y' = \lambda y$, where*

$$y_{k+1} = y_k + h\lambda y_k = \underbrace{(1 + h\lambda)}_{\text{growth factor}} y_k.$$

  - *Euler's method requires $|1 + h\lambda| \leq 1$ to be stable, which implies $-2 \leq h\lambda \leq 0$*
  - *The global error then satisfies*

$$e_k = l_k + (1 + h\lambda)e_{k-1}.$$

Stability Region for Forward Euler

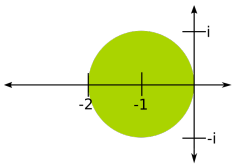
- The stability region of a general ODE constrains the eigenvalues of $h\boldsymbol{J_f}$
  - *The Mean Value Theorem implies that*

$$\exists \boldsymbol{\xi},\ \textit{such that } \boldsymbol{f}(t_k, \boldsymbol{y}_k) - \boldsymbol{f}(t_k, \boldsymbol{y}(t_k)) = \boldsymbol{J_f}(t_k, \boldsymbol{\xi})(\boldsymbol{y}_k - \boldsymbol{y}(t_k))$$

    and we know $||\boldsymbol{J_f}(t_k, \boldsymbol{\xi})||_2$ *is bounded by Lipschitz continuity.*
  - *Consequently the growth factor for Forward Euler is* $\boldsymbol{I} + h_k \boldsymbol{J_f}(t_k, \boldsymbol{\xi})$.
  - *The global error then satisfies*

$$\boldsymbol{e}_k = (\boldsymbol{I} + h_k \boldsymbol{J_f}(t_k, \boldsymbol{\xi}))\boldsymbol{e}_{k-1} + \boldsymbol{l}_k.$$

  - *Forward Euler is asymptotically stable if the spectral radius of the growth factor is less than one, which implies that the eigenvalues of* $h_k \boldsymbol{J_f}(t_k, \boldsymbol{\xi})$ *must always lie within a stability region that is a circle on the complex plane centered at* -1 *with radius* 1.



Backward Euler Method

Demo: *Backward Euler stability*
Activity: *Backward Euler Method*

- Implicit methods for ODEs form a sequence of solutions that satisfy conditions on a local approximation to the solution:

  The most basic implicit method is the backward Euler method

  $$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \boldsymbol{h}_k \boldsymbol{f}(t_{k+1}, \boldsymbol{y}_{k+1}),$$

  which solves for $\boldsymbol{y}_{k+1}$ so that a linear approximation of the solution at t_{k+1} passes through the point $(t_k, \boldsymbol{y}_k)$. Just like forward Euler, first-order accuracy is achieved by the linear approximation.

- The stability region of the backward Euler method is the left half of the complex plane:

  Such a method is called unconditionally stable. Note that the growth factor can be derived via

  $$y_{k+1} = y_k + h\lambda y_{k+1} = \frac{1}{1 - h\lambda} y_k,$$

  and satisfies $|1/(1 - h\lambda)| \leq 1$ so long as $h\lambda \leq 0$.

Trapezoid Method

- A second-order accurate implicit method is the *trapezoid method*

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \boldsymbol{h}_k(\boldsymbol{f}(t_k, \boldsymbol{y}_k) + \boldsymbol{f}(t_{k+1}, \boldsymbol{y}_{k+1}))/2,$$

  - *This method takes the average of the backward and forward Euler steps.*
  - *Its growth factor is* $\frac{1+h\lambda/2}{1-h\lambda/2}$.
  - *Since* $\left|\frac{1+h\lambda/2}{1-h\lambda/2}\right| \leq 1$ *for any* $\lambda < 0$*, the method is unconditionally stable.*
- Generally, methods can be derived from quadrature rules:
  - *Evaluate or approximate* $\boldsymbol{f}$ *at a set of points near* $(t_k, \boldsymbol{y}_k)$.
  - *Use weights from a given quadrature rule to approximate solution to local integral equation.*
  - *Finding appropriate quadrature nodes is hard, implicit methods in effect solve for them.*

Multi-Stage Methods

- *Multi-stage methods* construct $\boldsymbol{y}_{k+1}$ by approximating $\boldsymbol{y}$ between t_k and t_{k+1}:
  - *Runge-Kutta methods are the most well-known family of these, simple example is Heun's method,*

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h\left[\underbrace{\boldsymbol{f}(t_k, \boldsymbol{y}_k)}_{\boldsymbol{v}_1}/2 + \boldsymbol{f}\Big(t_k + h, \boldsymbol{y}_k + h\underbrace{\boldsymbol{f}(t_k, \boldsymbol{y}_k)}_{\boldsymbol{v}_1}\Big)/2\right].$$

  - *We can think of the above method as employing the trapezoid quadrature rule.*
  - *The difference between Heun's method and the (implicit) trapezoid method is that we evaluate at $\boldsymbol{f}(t_k + h, \boldsymbol{y}_k + h\boldsymbol{v}_1)$ rather than working with the implicit value of $\boldsymbol{f}(t_k + h, \boldsymbol{y}_{k+1})$.*
- The 4th order Runge-Kutta scheme is particularly popular:

  This scheme uses Simpson's rule,

$$\begin{aligned}
\boldsymbol{y}_{k+1} &= \boldsymbol{y}_k + (h/6)(\boldsymbol{v}_1 + 2\boldsymbol{v}_2 + 2\boldsymbol{v}_3 + \boldsymbol{v}_4) \\
\boldsymbol{v}_1 &= \boldsymbol{f}(t_k, \boldsymbol{y}_k), & \boldsymbol{v}_2 &= \boldsymbol{f}(t_k + h/2, \boldsymbol{y}_k + (h/2)\boldsymbol{v}_1), \\
\boldsymbol{v}_3 &= \boldsymbol{f}(t_k + h/2, \boldsymbol{y}_k + (h/2)\boldsymbol{v}_2), & \boldsymbol{v}_4 &= \boldsymbol{f}(t_k + h, \boldsymbol{y}_k + h\boldsymbol{v}_3).
\end{aligned}$$

Runge-Kutta Methods

- Runge-Kutta methods evaluate $\boldsymbol{f}$ at $t_k + c_i h$ for $c_0, \ldots, c_r \in [0, 1]$,

$$\boldsymbol{u}_k(t_{k+1}) = \boldsymbol{y}_k + \int_{t_k}^{t_k+h} \boldsymbol{f}(s, \boldsymbol{y}(s))ds \quad \approx \quad \boldsymbol{y}_k + h \sum_{i=0}^{r-1} w_i \boldsymbol{f}(t_k + c_i h, \hat{\boldsymbol{y}}_{ki}),$$

where $\{(c_i, w_i)\}_{i=0}^{r}$ *are quadrature (node, weight) pairs.*

- A general family of Runge Kutta methods can be defined by

$$\hat{\boldsymbol{y}}_{ki} = \boldsymbol{y}_k + h \sum_j a_{ij} \boldsymbol{f}(t_k + c_i h, \hat{\boldsymbol{y}}_{kj}).$$

Runge Kutta methods can then be represented by a Butcher tableau,

$\boldsymbol{c}$	$\boldsymbol{A}$
	$\boldsymbol{w}^T$

e.g. for RK4 $\boldsymbol{A}$ *has a single subdiagonal,*

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

If $\boldsymbol{A}$ *is strictly lower triangular* ($a_{ij} = 0$ *for* $j \geq i$*), the scheme is explicit, if* $\boldsymbol{A}$ *is lower-triangular then it is diagonally implicit, and otherwise implicit.*

Properties of Runge-Kutta and Extrapolation Methods

- Runge-Kutta methods are *self-starting*, but are harder to use to obtain error estimates.
  - *Self-starting means that we only need* $\boldsymbol{y}_k$ *to form* $\boldsymbol{y}_{k+1}$.
  - *Embedded Runge-Kutta schemes provides 4th + 5th order results, yielding an error estimate.*
- *Extrapolation methods* achieve high accuracy by successively reducing step-size.

  Use single-step method with step sizes $h, h/2, h/4, \ldots$ *to approximate solution at* $t_k + h$.

Multistep Methods

- *Multistep methods* employ $\{\boldsymbol{y}_k\}_{i=0}^{k}$ to compute $\boldsymbol{y}_{k+1}$:

  Linear multistep methods have the form,

$$\boldsymbol{y}_{k+1} = \sum_{i=1}^{m} \alpha_i \boldsymbol{y}_{k+1-i} + h \sum_{i=0}^{m} \beta_i \boldsymbol{f}(t_{k+1-i}, \boldsymbol{y}_{k+1-i}).$$

  Interpolation is used to determine each α_i and β_i, method is explicit if $\beta_0 = 0$.

- Multistep methods are not self-starting, but have practical advantages:
  - *Can be initiated by Runge-Kutta methods.*
  - *They require few function evaluations.*
  - *Generalize to non-uniformly-spaced points (multivalue methods).*