

Outline

- 1 Ordinary Differential Equations
- 2 Numerical Solution of ODEs
- 3 Additional Numerical Methods



Differential Equations

- Differential equations involve derivatives of unknown solution function
- *Ordinary differential equation* (ODE): all derivatives are with respect to single independent variable, often representing time
- Solution of differential equation is *function* in infinite-dimensional space of functions
- Numerical solution of differential equations is based on finite-dimensional approximation
- Differential equation is replaced by algebraic equation whose solution approximates that of given differential equation



Single ODE (IVP) Examples:

- First-order derivative in time - *initial value problem*.
- $y' = \frac{1}{2}y, \quad y(0) = 1 : \quad y(t) = e^{\frac{1}{2}t}.$
- $y' = -2 - y + y^2 + \text{I.C.s}$ (Ricatti Equation example)
- $y' = \sin(t) + \text{I.C.s}$
- $y' = \lambda y + \cos(t) + \text{I.C.s}$
- $y' = e^{-y} + \text{I.C.s}$
- etc.

System of ODEs

$$\begin{bmatrix} \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \end{bmatrix} = \begin{bmatrix} f_1(t, y_1, y_2) \\ f_2(t, y_1, y_2) \end{bmatrix} = \mathbf{f}(t, \mathbf{y})$$

$$\mathbf{y}' := \frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y})$$

$$\begin{bmatrix} \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \end{bmatrix}_{t_k} = \begin{bmatrix} f_1(t, y_1, y_2) \\ f_2(t, y_1, y_2) \end{bmatrix}_{t_k} = \mathbf{f}(t_k, \mathbf{y}(t_k))$$

System of ODEs: Solved Numerically

Given data at t_k , find solution at t_{k+1} .

One example,

$$\left. \frac{d\mathbf{y}}{dt} \right|_{t_k} \approx \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_{(k)}} \approx \mathbf{f}(t_k, \mathbf{y}_k)$$

Yields Euler's method (a.k.a. Euler forward, "EF"):

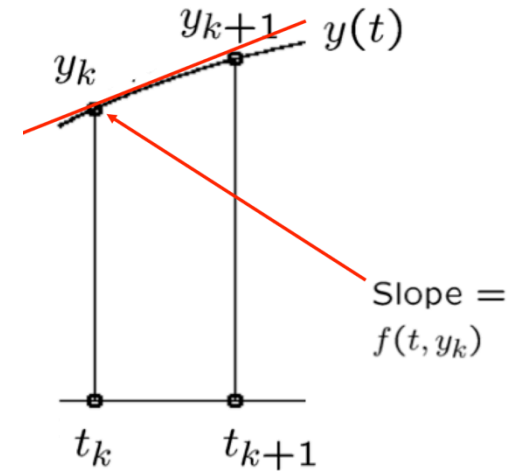
$$\begin{aligned} \mathbf{y}_{k+1} &= \mathbf{y}_k + h_{(k)} \mathbf{f}_k \\ &= \mathbf{y}_k + h \mathbf{f}_k \quad (\text{constant step-size case}) \end{aligned}$$

System of ODEs: Solved Numerically

Given data at t_k , find solution at t_{k+1} .

One example,

$$\left. \frac{d\mathbf{y}}{dt} \right|_{t_k} \approx \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h_{(k)}} \approx \mathbf{f}(t_k, \mathbf{y}_k)$$



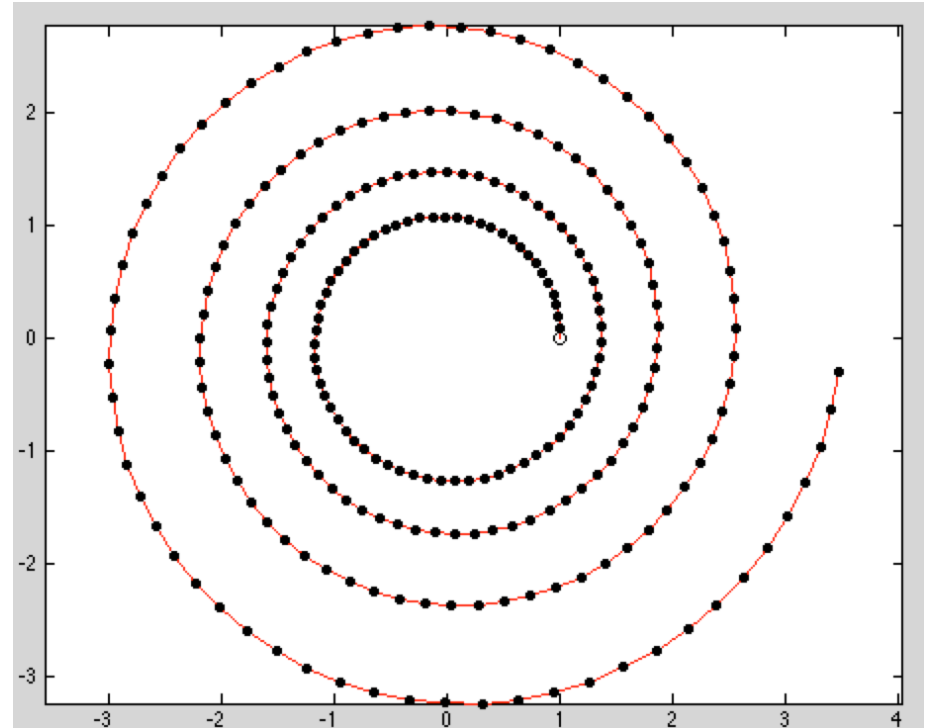
Yields Euler's method (a.k.a. Euler forward, "EF"):

$$\begin{aligned} \mathbf{y}_{k+1} &= \mathbf{y}_k + h_{(k)} \mathbf{f}_k \\ &= \mathbf{y}_k + h \mathbf{f}_k \quad (\text{constant step-size case}) \end{aligned}$$

Matlab Example: orbit.m, orbit1.m

```
T = 2*pi; dt = T/n; % Tfinal and dt
A = [ 0 -1 ; 1 0 ]; I=eye(2);
y0 = [ 1 ; 0 ]; % INITIAL CONDITION
y=y0;

io=floor(1+n/100);
for k=1:n;
    y=y + dt*(A*y);
    plot(y(1),y(2),'r. '); axis equal;hold on
end;
plot(y(1),y(2),'kx '); axis equal; hold on
```



Order of ODE

- *Order* of ODE is determined by highest-order derivative of solution function appearing in ODE
- ODE with higher-order derivatives can be transformed into equivalent first-order system *(For Initial Value Problems...)*
- We will discuss numerical solution methods only for first-order ODEs
- Most ODE software is designed to solve only first-order equations



Higher-Order ODEs, continued

- For k -th order ODE

$$y^{(k)}(t) = f(t, y, y', \dots, y^{(k-1)})$$

define k new unknown functions

$$u_1(t) = y(t), u_2(t) = y'(t), \dots, u_k(t) = y^{(k-1)}(t)$$

- Then original ODE is equivalent to first-order system

$$\begin{bmatrix} u_1'(t) \\ u_2'(t) \\ \vdots \\ u_{k-1}'(t) \\ u_k'(t) \end{bmatrix} = \begin{bmatrix} u_2(t) \\ u_3(t) \\ \vdots \\ u_k(t) \\ f(t, u_1, u_2, \dots, u_k) \end{bmatrix}$$



Converting Higher Order ODEs to First Order

Consider example:

$$y^{iv} = f(t, y, y', y'', y''')$$

Let $y_1 = y$, $y_2 = y'$, $y_3 = y''$, and $y_4 = y'''$.

$$\frac{d}{dt} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & & \\ & 0 & 1 & \\ & & 0 & 1 \\ & & & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ f \end{bmatrix}$$

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y} + \mathbf{f}$$

Example: Newton's Second Law

- Newton's Second Law of Motion, $F = ma$, is second-order ODE, since acceleration a is second derivative of position coordinate, which we denote by y
- Thus, ODE has form

$$y'' = F/m$$

where F and m are force and mass, respectively

- Defining $u_1 = y$ and $u_2 = y'$ yields equivalent system of two first-order ODEs

$$\begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \begin{bmatrix} u_2 \\ F/m \end{bmatrix}$$



Example, continued

- We can now use methods for first-order equations to solve this system
- First component of solution u_1 is solution y of original second-order equation
- Second component of solution u_2 is velocity y'



Ordinary Differential Equations

- General first-order system of ODEs has form

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

where $\mathbf{y}: \mathbb{R} \rightarrow \mathbb{R}^n$, $\mathbf{f}: \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n$, and $\mathbf{y}' = d\mathbf{y}/dt$ denotes derivative with respect to t ,

$$\begin{bmatrix} y_1'(t) \\ y_2'(t) \\ \vdots \\ y_n'(t) \end{bmatrix} = \begin{bmatrix} dy_1(t)/dt \\ dy_2(t)/dt \\ \vdots \\ dy_n(t)/dt \end{bmatrix}$$

- Function \mathbf{f} is given and we wish to determine unknown function \mathbf{y} satisfying ODE
- For simplicity, we will often consider special case of single scalar ODE, $n = 1$



Initial Value Problems

- By itself, ODE $y' = f(t, y)$ does not determine unique solution function
- This is because ODE merely specifies *slope* $y'(t)$ of solution function at each point, but not actual value $y(t)$ at any point
- Infinite family of functions satisfies ODE, in general, provided f is sufficiently smooth
- To single out particular solution, value y_0 of solution function must be specified at some point t_0

y_0 is the initial condition.



Initial Value Problems, continued

- Thus, part of given problem data is requirement that $\mathbf{y}(t_0) = \mathbf{y}_0$, which determines unique solution to ODE
- Because of interpretation of independent variable t as time, think of t_0 as initial time and \mathbf{y}_0 as initial value
- Hence, this is termed *initial value problem*, or *IVP*
- ODE governs evolution of system in time from its initial state \mathbf{y}_0 at time t_0 onward, and we seek function $\mathbf{y}(t)$ that describes state of system as function of time



Example: Initial Value Problem

- Consider scalar ODE

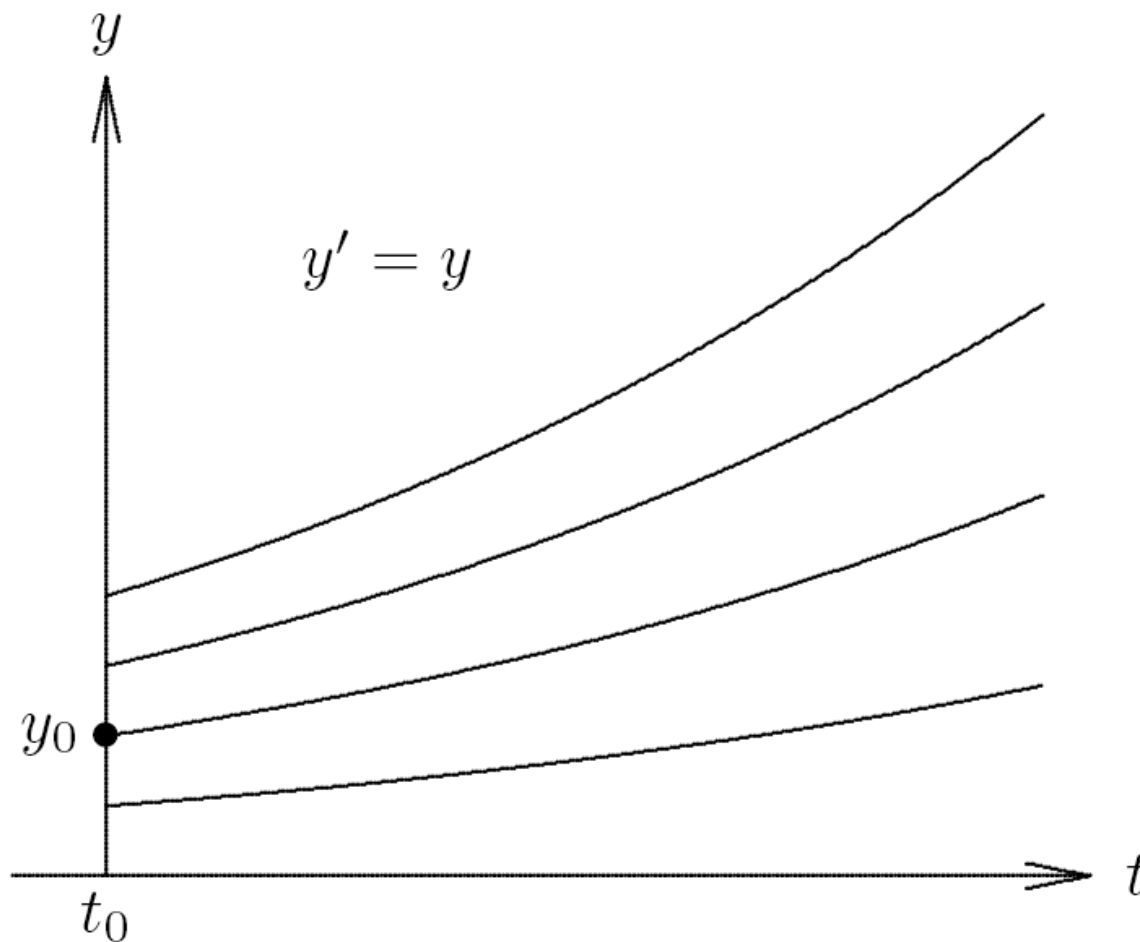
$$y' = y$$

- Family of solutions is given by $y(t) = ce^t$, where c is any real constant
- Imposing initial condition $y(t_0) = y_0$ singles out unique particular solution
- For this example, if $t_0 = 0$, then $c = y_0$, which means that solution is $y(t) = y_0e^t$



Example: Initial Value Problem

Family of solutions for ODE $y' = y$



Stability of Solutions

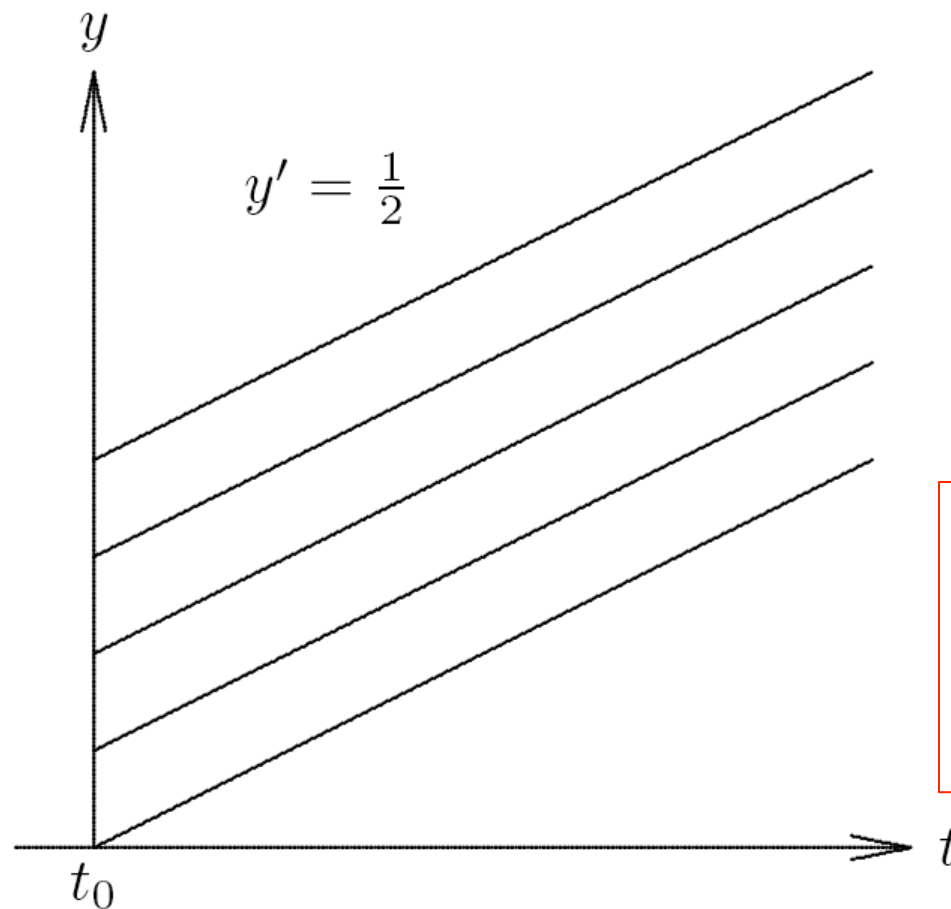
Solution of ODE is

- *Stable* if solutions resulting from perturbations of initial value remain close to original solution
- *Asymptotically stable* if solutions resulting from perturbations converge back to original solution
- *Unstable* if solutions resulting from perturbations diverge away from original solution without bound



Example: Stable Solutions

Family of solutions for ODE $y' = \frac{1}{2}$

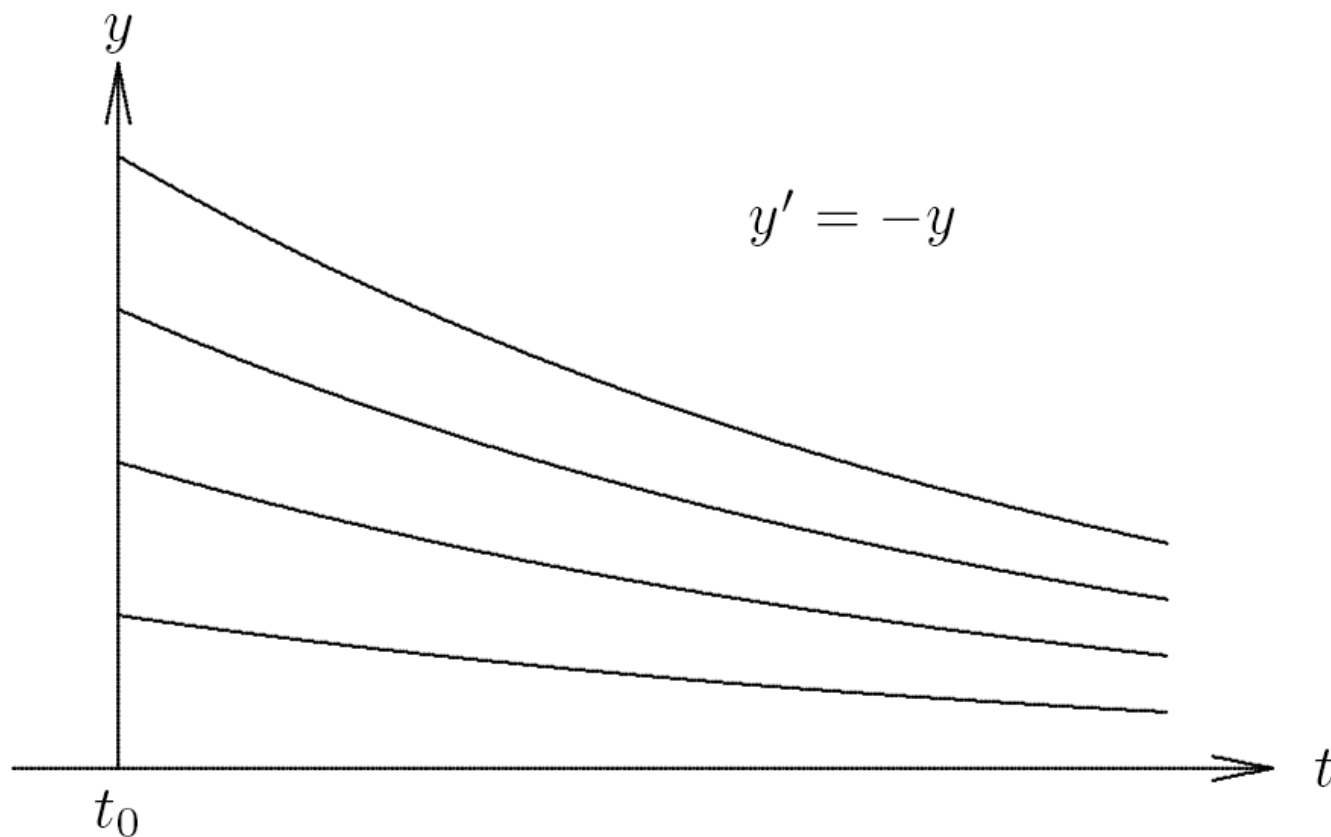


Closed Orbits are another example of stable, but not asymptotically stable ODEs.

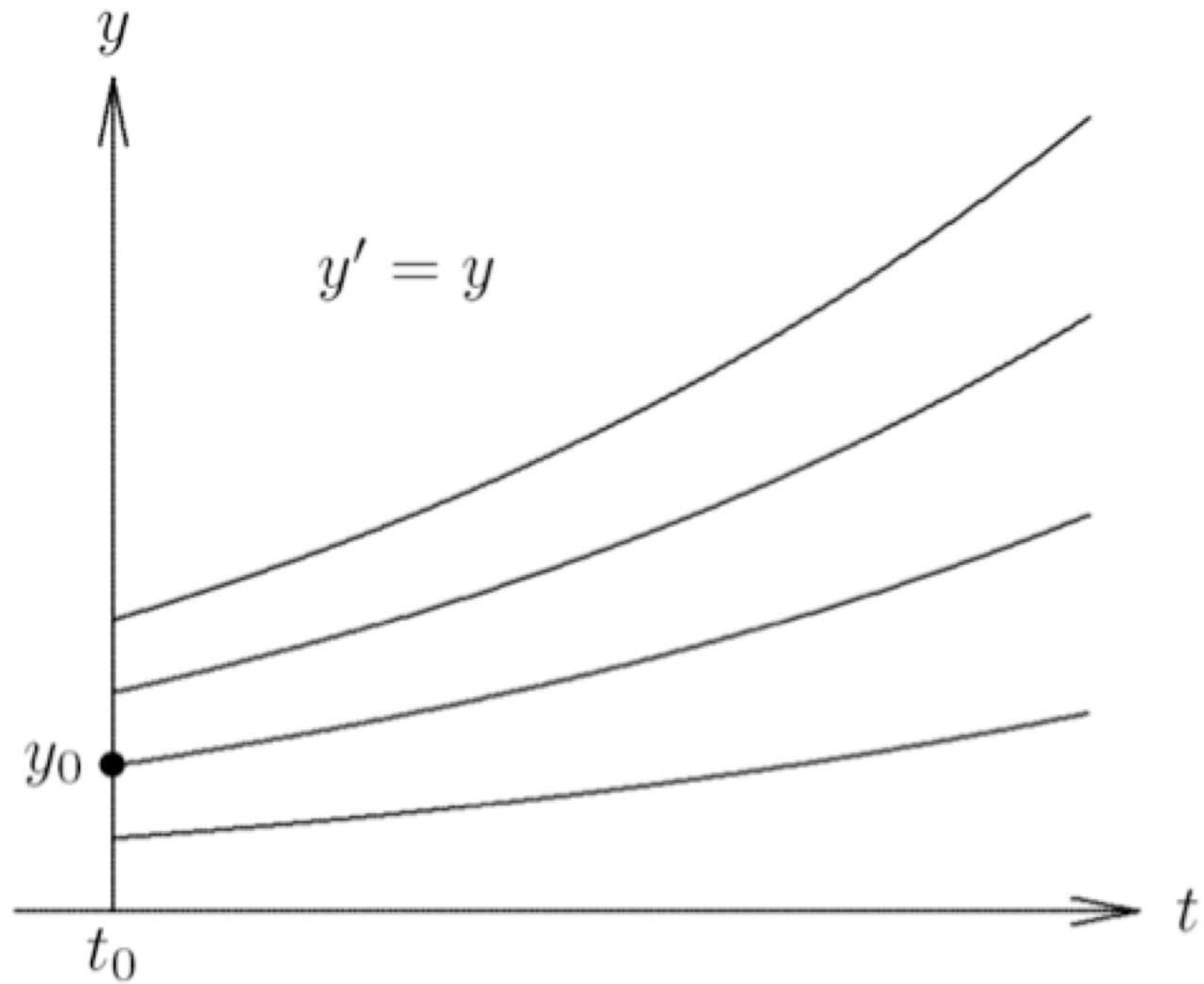


Example: Asymptotically Stable Solutions

Family of solutions for ODE $y' = -y$



□ Unstable ODE Example



Example: Stability of Solutions

- Consider scalar ODE $y' = \lambda y$, where λ is constant.
- Solution is given by $y(t) = y_0 e^{\lambda t}$, where $t_0 = 0$ is initial time and $y(0) = y_0$ is initial value
- For real λ
 - $\lambda > 0$: all nonzero solutions grow exponentially, so every solution is unstable
 - $\lambda < 0$: all nonzero solutions decay exponentially, so every solution is not only stable, but asymptotically stable
- For complex λ
 - $\operatorname{Re}(\lambda) > 0$: unstable
 - $\operatorname{Re}(\lambda) < 0$: asymptotically stable
 - $\operatorname{Re}(\lambda) = 0$: stable but not asymptotically stable



Most Important Model Problem

- $\frac{dy}{dt} = \lambda y, \quad y(t = 0) = y_0.$

- Exact solution: $y(t) = y_0 e^{\lambda t}.$

- Note that, $y_n = y_0 e^{\lambda n \Delta t}$, so

$$\begin{aligned} y_n &= y_0 e^{\lambda(t_{n-1} + \Delta t)} = y_0 e^{\lambda t_{n-1}} e^{\lambda \Delta t} \\ &= e^{\lambda \Delta t} y_{n-1}. \\ &= \tilde{G} y_{n-1}, \end{aligned}$$

where \tilde{G} is a constant, referred to as the (analytical) *growth factor*.

Example: Linear System of ODEs

- Linear, homogeneous system of ODEs with constant coefficients has form

$$\mathbf{y}' = \mathbf{A}\mathbf{y}$$

where \mathbf{A} is $n \times n$ matrix, and initial condition is $\mathbf{y}(0) = \mathbf{y}_0$

- Suppose \mathbf{A} is diagonalizable, with eigenvalues λ_i and corresponding eigenvectors \mathbf{v}_i , $i = 1, \dots, n$
- Express \mathbf{y}_0 as linear combination $\mathbf{y}_0 = \sum_{i=1}^n \alpha_i \mathbf{v}_i$
- Then

$$\mathbf{y}(t) = \sum_{i=1}^n \alpha_i \mathbf{v}_i e^{\lambda_i t}$$

Here, linearity implies \mathbf{A} is a constant matrix $\mathbf{A} \neq \mathbf{A}(\mathbf{y})$. (Actually, that is also implied by the “constant coefficients” qualifier.)

is solution to ODE satisfying initial condition $\mathbf{y}(0) = \mathbf{y}_0$



Eigenvalues and ODEs

$$\frac{d\mathbf{y}}{dt} = J\mathbf{y} + \mathbf{f}(t)$$

Assume $J = \text{constant}$ and there exist n eigenvectors \mathbf{v}_j such that

$$J\mathbf{v}_j = \lambda_j\mathbf{v}_j$$

$$JV = V\Lambda = (\mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_n) \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

Then, there exists a set of coefficients $\hat{y}_j(t)$ such that

$$\mathbf{y} = \sum_{j=1}^n \mathbf{v}_j \hat{y}_j \iff \mathbf{y} = V\hat{\mathbf{y}} \iff \hat{\mathbf{y}} = V^{-1}\mathbf{y}$$

$$J\mathbf{y} = \sum_{j=1}^n J\mathbf{v}_j \hat{y}_j = \sum_{j=1}^n \lambda_j \mathbf{v}_j \hat{y}_j = \sum_{j=1}^n \mathbf{v}_j \lambda_j \hat{y}_j = V\Lambda\hat{\mathbf{y}}.$$

Eigenvalues and ODEs

Inserting the expansion $\mathbf{y} = V\hat{\mathbf{y}}$ into our ODE...

$$\frac{d\mathbf{y}}{dt} = J\mathbf{y} + \mathbf{f}$$

$$\begin{aligned}\frac{d}{dt}V\hat{\mathbf{y}} &= JV\hat{\mathbf{y}} + V\hat{\mathbf{f}} \\ &= V\Lambda\hat{\mathbf{y}} + V\hat{\mathbf{f}}\end{aligned}$$

Multiply through by V^{-1} :

$$\frac{d\hat{\mathbf{y}}}{dt} = \Lambda\hat{\mathbf{y}} + \hat{\mathbf{f}}$$

Eigenvalues and ODEs

$$\frac{d}{dt} \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix} = \begin{bmatrix} \lambda_1 & & \\ & \cdots & \\ & & \lambda_n \end{bmatrix} \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix} + \begin{pmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{pmatrix}$$

$$= \begin{pmatrix} \lambda_1 \hat{y}_1 \\ \vdots \\ \lambda_n \hat{y}_n \end{pmatrix} + \begin{pmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{pmatrix}$$

$$\frac{d\hat{y}_i}{dt} = \lambda_i \hat{y}_i + \hat{f}_i, \quad i = 1, \dots, n$$

- We now have n *decoupled* systems.
- Numerically, we solve these as the coupled system $\mathbf{y} = J\mathbf{y} + \mathbf{f}$.
- The behavior, however, is the same as this decoupled system, which is easier to understand.
- In particular, stability is governed by the maximum real part of the λ_i s.

Example, continued

- Eigenvalues of A with positive real parts yield exponentially growing solution components
- Eigenvalues with negative real parts yield exponentially decaying solution components
- Eigenvalues with zero real parts (i.e., pure imaginary) yield oscillatory solution components
- Solutions stable if $\operatorname{Re}(\lambda_i) \leq 0$ for every eigenvalue, and asymptotically stable if $\operatorname{Re}(\lambda_i) < 0$ for every eigenvalue, but unstable if $\operatorname{Re}(\lambda_i) > 0$ for any eigenvalue



Stability of Solutions, continued

- For general nonlinear system of ODEs $y' = f(t, y)$, determining stability of solutions is more complicated
- ODE can be linearized locally about solution $y(t)$ by truncated Taylor series, yielding linear ODE

$$z' = J_f(t, y(t)) z$$

where J_f is Jacobian matrix of f with respect to y

- Eigenvalues of J_f determine stability locally, but conclusions drawn may not be valid globally



Numerical Solution of ODEs

- Analytical solution of ODE is closed-form formula that can be evaluated at any point t
- Numerical solution of ODE is table of approximate values of solution function at discrete set of points
- Numerical solution is generated by simulating behavior of system governed by ODE
- Starting at t_0 with given initial value y_0 , we track trajectory dictated by ODE
- Evaluating $f(t_0, y_0)$ tells us slope of trajectory at that point
- We use this information to predict value y_1 of solution at future time $t_1 = t_0 + h$ for some suitably chosen time increment h



Numerical Solution of ODEs, continued

- Approximate solution values are generated step by step in increments moving across interval in which solution is sought
- In stepping from one discrete point to next, we incur some error, which means that next approximate solution value lies on *different* solution from one we started on
- Stability or instability of solutions determines, in part, whether such errors are magnified or diminished with time
- Two types of stability to consider:
 - Stability of ODE
 - Stability of numerical method



Developing Numerical Solution Approaches for ODEs (IVPs)

- ❑ Principal considerations:

- ❑ Deriving a **computable formula** for \mathbf{y}_{k+1} , given \mathbf{y}_k , $\mathbf{f}(\mathbf{y}_k, t_k)$ (and perhaps, $\mathbf{f}(\mathbf{y}_{k-1}, t_{k-1}), \dots$).
- ❑ Understanding **accuracy** as a function of stepsize, h , and function f .
- ❑ Understanding **stability** as a function of stepsize, h , and function f .

Developing Numerical Solution Approaches for ODEs (IVPs)

- ❑ Some methods we'll encounter:
 - ❑ Euler's method (aka Forward Euler)
 - ❑ Backward Euler
 - ❑ Trapezoid Method
 - ❑ Backward difference formulae of order k (BDF k)
 - ❑ k th-order Runge-Kutta methods
 - ❑ ...

- ❑ These methods are characterized by
 - ❑ Implicit, explicit, semi-implicit
 - ❑ Stability properties
 - ❑ Accuracy
 - ❑ **Cost**

Developing Numerical Solution Approaches for ODEs (IVPs)

❑ Some methods we'll encounter:

- ❑ Euler's method (aka Forward Euler) $O(h)$
- ❑ Backward Euler $O(h)$
- ❑ Trapezoid Method $O(h^2)$ – **not L-stable** 😞
- ❑ Backward difference formulae of order k (BDF k) $O(h^k)$
- ❑ k th-order Runge-Kutta methods $O(h^k)$
- ❑ ...

❑ These methods are characterized by

- ❑ Implicit, explicit, semi-implicit
- ❑ Stability properties
- ❑ Accuracy
- ❑ **Cost**

Euler's Method

- For general system of ODEs $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, consider Taylor series

$$\begin{aligned}\mathbf{y}(t+h) &= \mathbf{y}(t) + h\mathbf{y}'(t) + \frac{h^2}{2}\mathbf{y}''(t) + \dots \\ &= \mathbf{y}(t) + h\mathbf{f}(t, \mathbf{y}(t)) + \frac{h^2}{2}\mathbf{y}''(t) + \dots\end{aligned}$$

- *Euler's method* results from dropping terms of second and higher order to obtain approximate solution value

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

- Euler's method advances solution by extrapolating along straight line whose slope is given by $\mathbf{f}(t_k, \mathbf{y}_k)$
- Euler's method is *single-step* method because it depends on information at only one point in time to advance to next point



Example: Euler's Method

- Applying Euler's method to ODE $y' = y$ with step size h , we advance solution from time $t_0 = 0$ to time $t_1 = t_0 + h$

$$y_1 = y_0 + hy'_0 = y_0 + hy_0 = (1 + h)y_0$$

- Value for solution we obtain at t_1 is not exact, $y_1 \neq y(t_1)$
- For example, if $t_0 = 0$, $y_0 = 1$, and $h = 0.5$, then $y_1 = 1.5$, whereas exact solution for this initial value is $y(0.5) = \exp(0.5) \approx 1.649$
- Thus, y_1 lies on different solution from one we started on

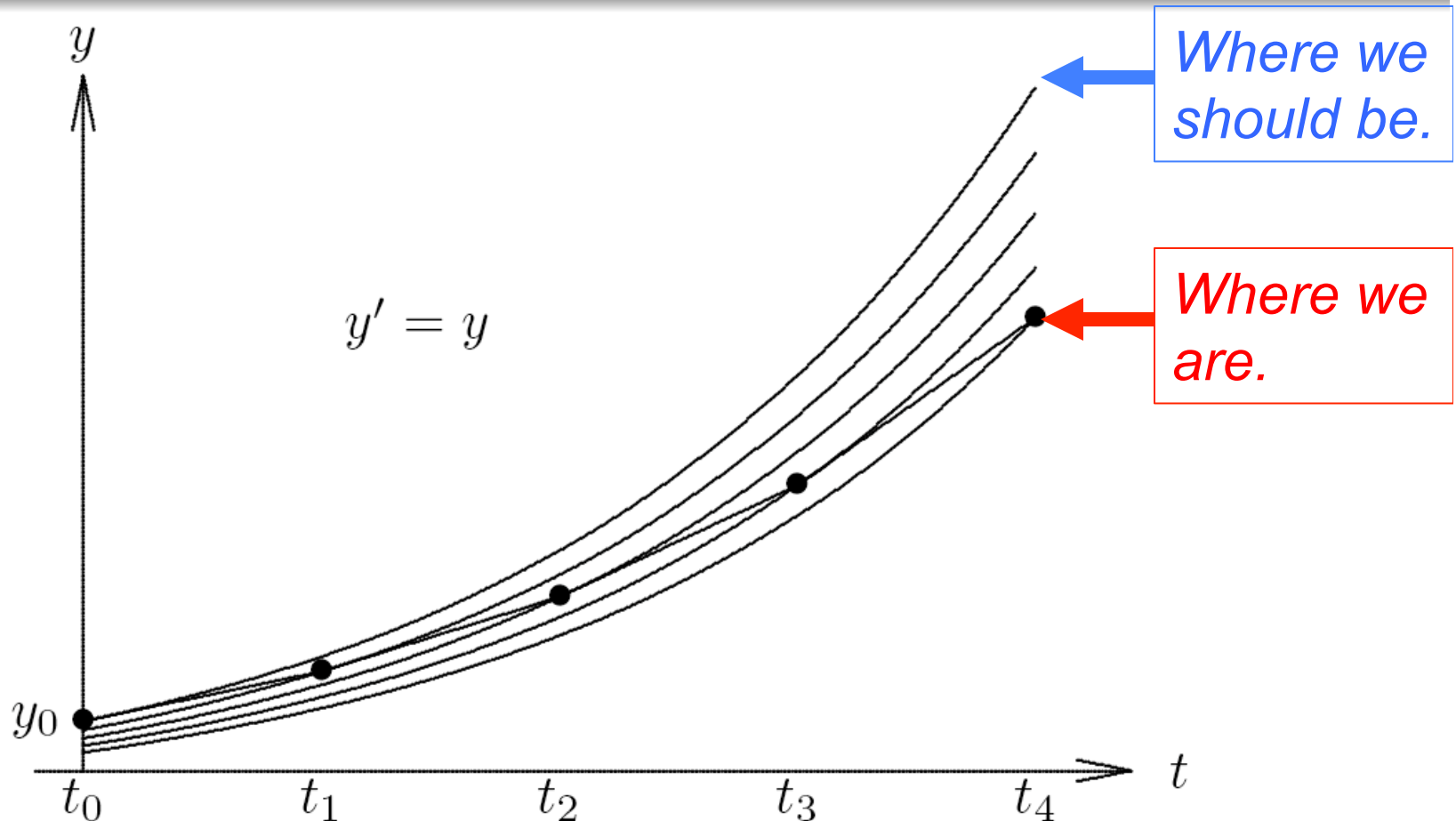


Example, continued

- To continue numerical solution process, we take another step from t_1 to $t_2 = t_1 + h = 1.0$, obtaining
$$y_2 = y_1 + hy_1 = 1.5 + (0.5)(1.5) = 2.25$$
- Now y_2 differs not only from true solution of original problem at $t = 1$, $y(1) = \exp(1) \approx 2.718$, but it also differs from solution through previous point (t_1, y_1) , which has approximate value 2.473 at $t = 1$
- Thus, we have moved to still another solution for this ODE



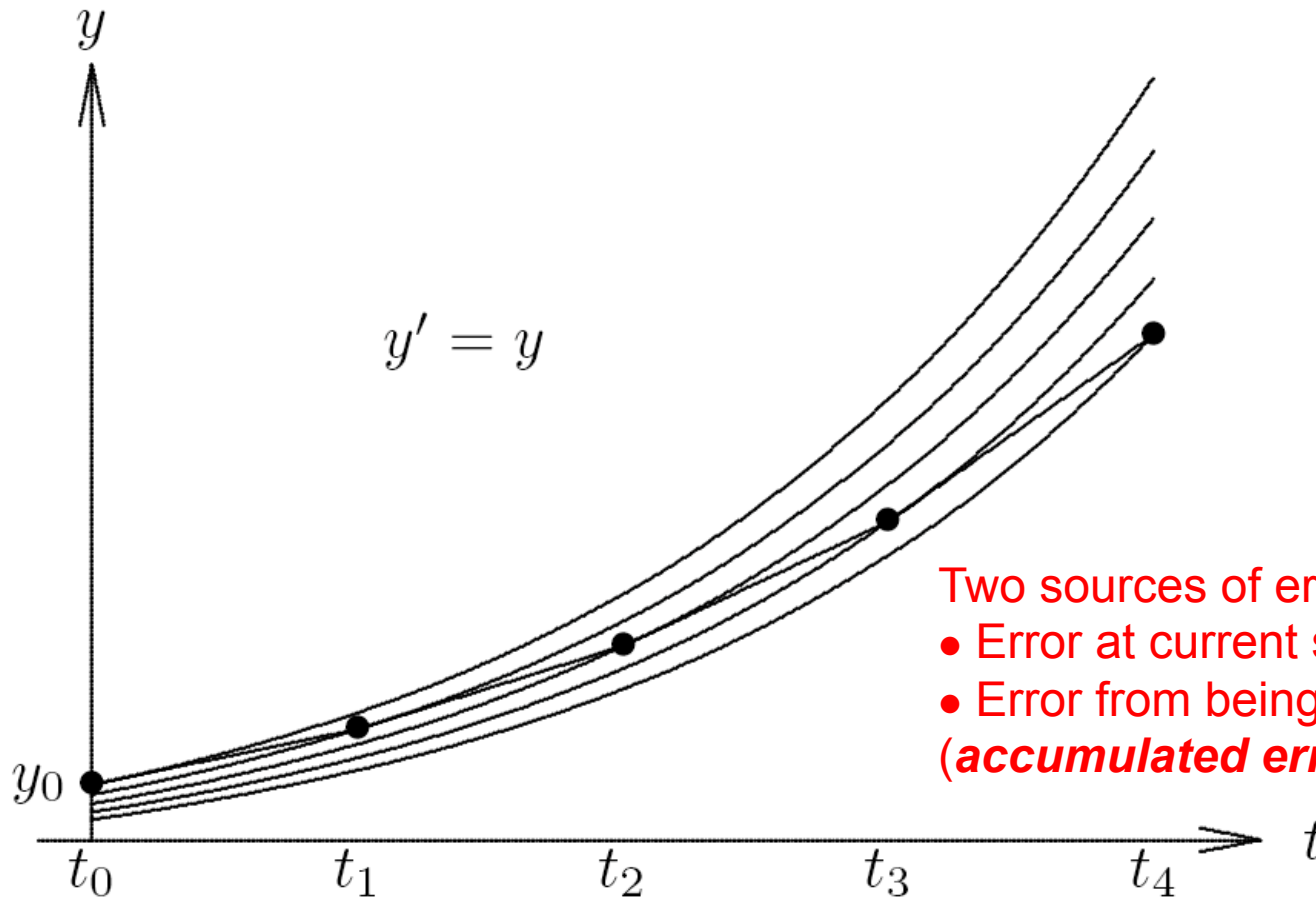
Example, continued



For unstable solutions, errors in numerical solution grow with time



Example, continued

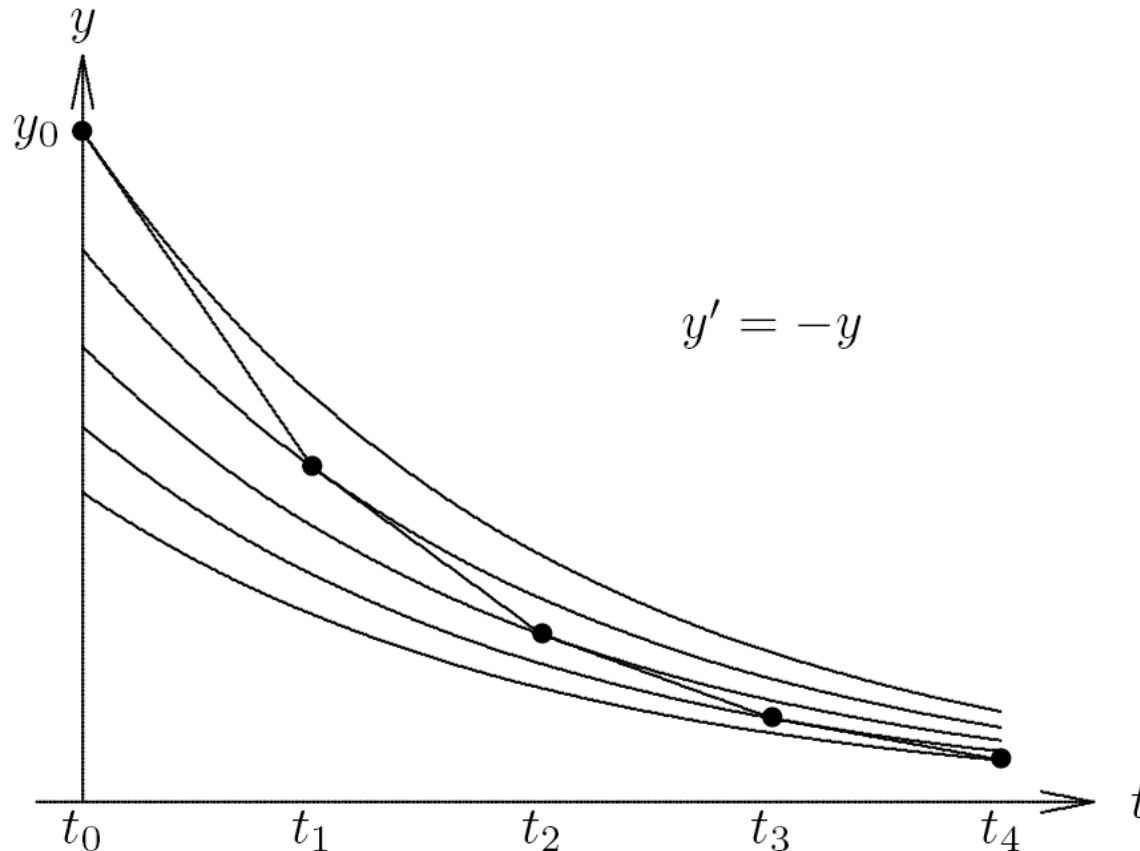


Two sources of error:

- Error at current step (**local error**)
- Error from being on the wrong trajectory (**accumulated error**)



Example, continued



For stable solutions, errors in numerical solution may diminish with time
(Local errors are suppressed.)



Errors in Numerical Solution of ODEs

- Numerical methods for solving ODEs incur two distinct types of error
 - *Rounding error*, which is due to finite precision of floating-point arithmetic
 - *Truncation error* (*discretization error*), which is due to approximation method used and would remain even if all arithmetic were exact
- In practice, truncation error is dominant factor determining accuracy of numerical solutions of ODEs, so we will henceforth ignore rounding error

(Keep in mind, however, that even with a perfect algorithm, round-off is an ever-present source of noise in the system – e.g., we must assume that all eigenmodes are present, so any unstable mode will grow.)



Global Error and Local Error

Truncation error at any point t_k can be broken down into

- **Global error**: difference between computed solution and true solution $\mathbf{y}(t)$ passing through initial point (t_0, \mathbf{y}_0)

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

- **Local error**: error made in one step of numerical method

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

where $\mathbf{u}_{k-1}(t)$ is true solution passing through previous point $(t_{k-1}, \mathbf{y}_{k-1})$



Global Error and Local Error

Truncation error at any point t_k can be broken down into

- **Global error**: difference between computed solution and true solution $\mathbf{y}(t)$ passing through initial point (t_0, \mathbf{y}_0)

$$\mathbf{e}_k = \mathbf{y}_k - \mathbf{y}(t_k) \quad \text{GTE - Global Truncation Error}$$

- **Local error**: error made in one step of numerical method

$$\mathbf{l}_k = \mathbf{y}_k - \mathbf{u}_{k-1}(t_k) \quad \text{LTE - Local Truncation Error}$$

where $\mathbf{u}_{k-1}(t)$ is true solution passing through previous point $(t_{k-1}, \mathbf{y}_{k-1})$

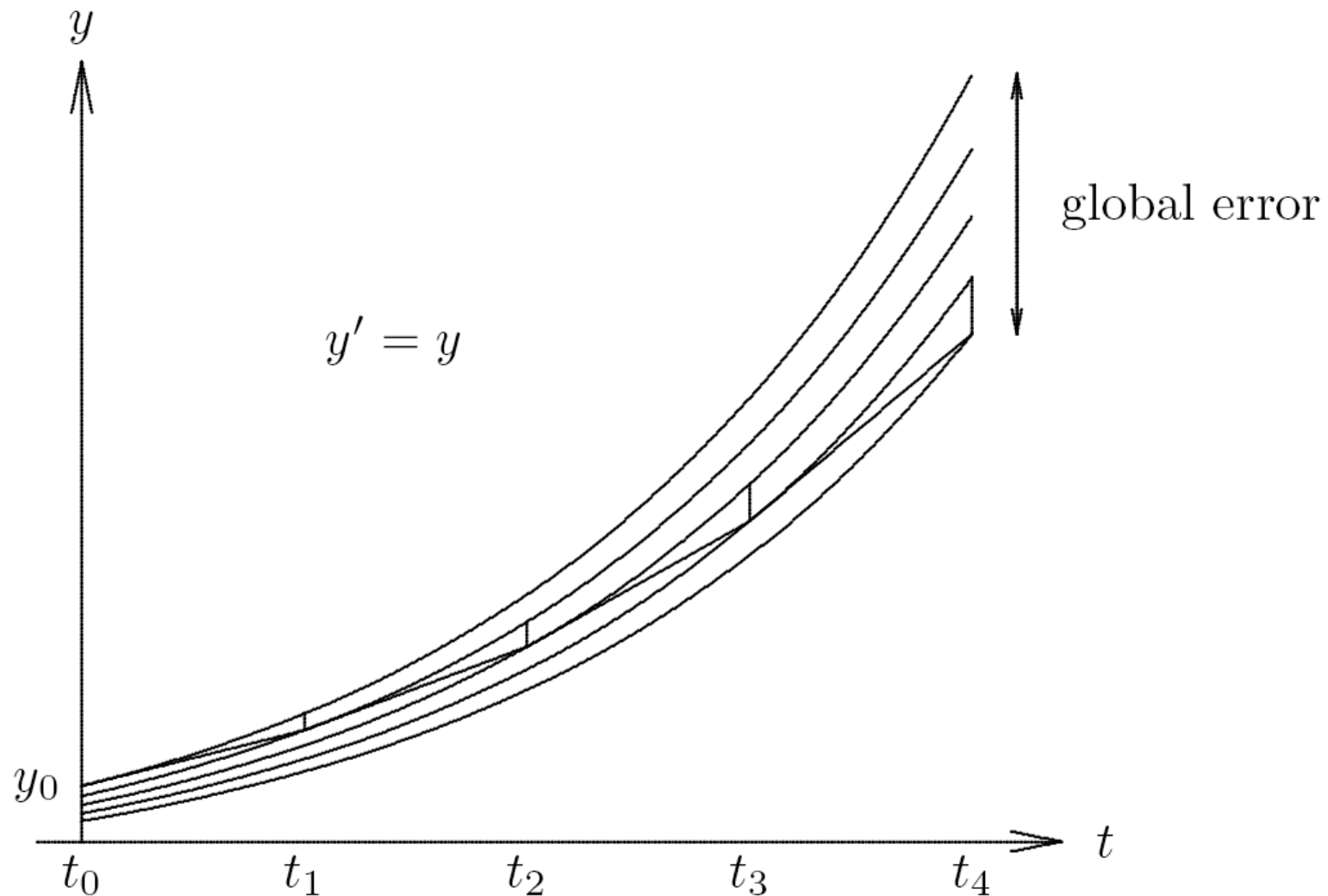


Global Error and Local Error, continued

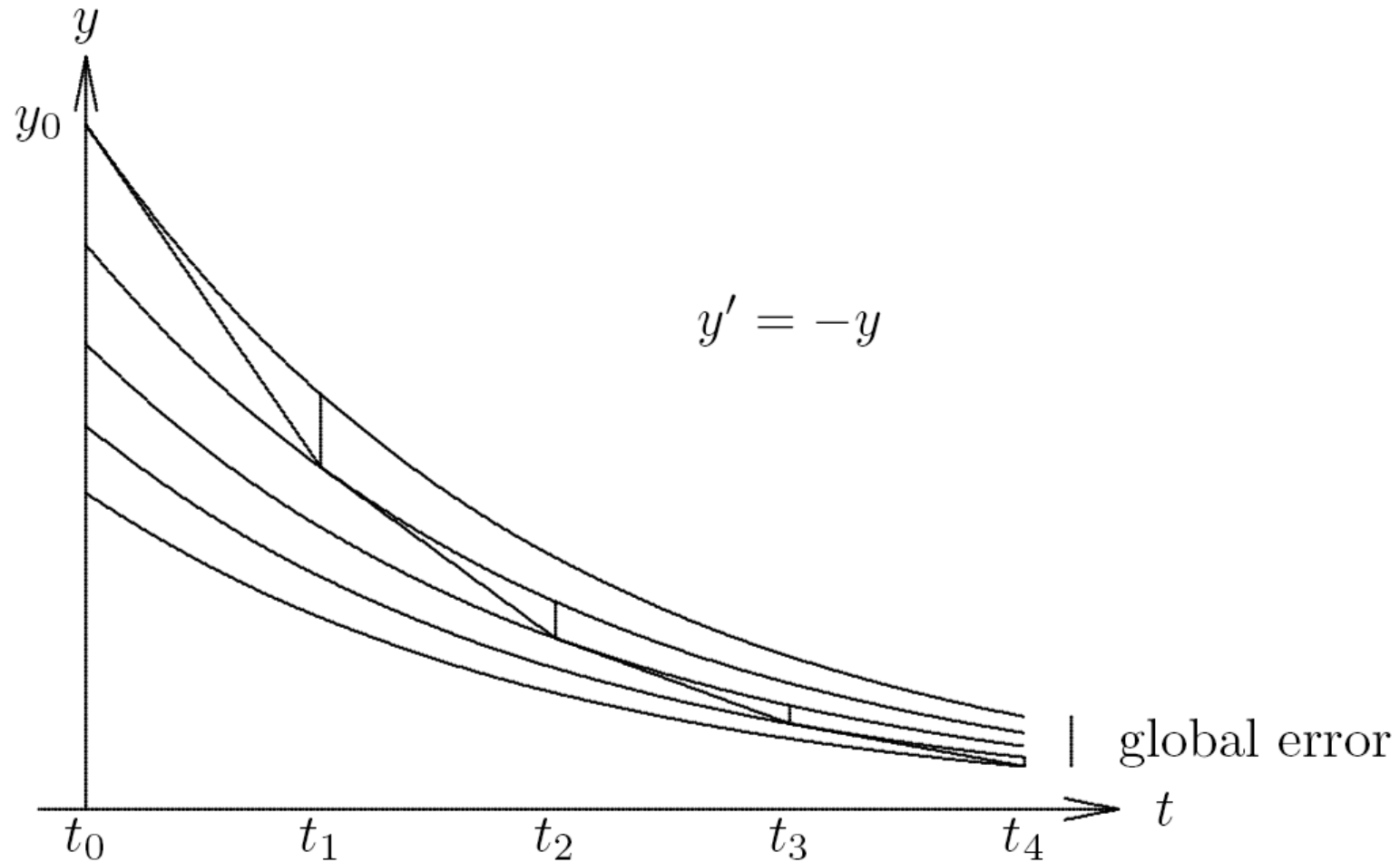
- Global error is not necessarily sum of local errors
- Global error is generally greater than sum of local errors if solutions are unstable, but may be less than sum if solutions are stable
- Having small global error is what we want, but we can control only local error directly



Global Error and Local Error, continued



Global and Local Error, continued



Order of Accuracy

- *Order of accuracy* of numerical method is p if

$$\ell_k = \mathcal{O}(h_k^{p+1})$$

- Then local error per unit step, $\ell_k/h_k = \mathcal{O}(h_k^p)$
- Under reasonable conditions, $e_k = \mathcal{O}(h^p)$, where h is average step size

GTE – Global Truncation Error $\sim \mathcal{O}(h^p)$

LTE – Local Truncation Error $\sim \mathcal{O}(h^{p+1})$

Global Truncation Error is what you expect after final time $T = n \cdot h$, where n =number of steps.

Making n local errors of size $\mathcal{O}(h^{p+1})$, expect $e(T) = \mathcal{O}(nh^{p+1}) = T \mathcal{O}(h^p)$.



Stability

- Numerical method is *stable* if small perturbations do not cause resulting numerical solutions to diverge from each other without bound
- Such divergence of numerical solutions could be caused by instability of solution to ODE, but can also be due to numerical method itself, even when solutions to ODE are stable

Stability, Accuracy and Cost of the numerical scheme are the primary considerations in the development of ODE solvers.



Determining Stability and Accuracy

- Simple approach to determining stability and accuracy of numerical method is to apply it to scalar ODE $y' = \lambda y$, where λ is (possibly complex) constant
- Exact solution is given by $y(t) = y_0 e^{\lambda t}$, where $y(0) = y_0$ is initial condition
- Determine stability of numerical method by characterizing growth of numerical solution
- Determine accuracy of numerical method by comparing exact and numerical solutions



Example: Euler's Method

- Applying Euler's method to $y' = \lambda y$ using fixed step size h ,

$$y_{k+1} = y_k + h\lambda y_k = (1 + h\lambda)y_k$$

which means that

$$y_k = (1 + h\lambda)^k y_0$$

- If $\text{Re}(\lambda) < 0$, exact solution decays to zero as t increases, as does computed solution if

$$|1 + h\lambda| < 1$$

which holds if $h\lambda$ lies inside circle in complex plane of radius 1 centered at -1

Here, $(1 + \lambda h)$ is the *growth factor*.



Recall Eigenvalues and ODEs, Analytical Case

$$\frac{d\mathbf{y}}{dt} = J\mathbf{y} + \mathbf{f}(t)$$

Assume $J = \text{constant}$ and there exist n eigenvectors \mathbf{v}_j such that

$$J\mathbf{v}_j = \lambda_j\mathbf{v}_j$$

$$JV = V\Lambda = (\mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_n) \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

Then, there exists a set of coefficients $\hat{y}_j(t)$ such that

$$\mathbf{y} = \sum_{j=1}^n \mathbf{v}_j \hat{y}_j \iff \mathbf{y} = V\hat{\mathbf{y}} \iff \hat{\mathbf{y}} = V^{-1}\mathbf{y}$$

$$J\mathbf{y} = \sum_{j=1}^n J\mathbf{v}_j \hat{y}_j = \sum_{j=1}^n \lambda_j \mathbf{v}_j \hat{y}_j$$

Recall, Eigenvalues and ODEs, Analytical Case

Inserting this expansion into our ODE...

$$\frac{d\mathbf{y}}{dt} = J\mathbf{y} + \mathbf{f}$$

$$\begin{aligned}\frac{d}{dt}V\hat{\mathbf{y}} &= JV\hat{\mathbf{y}} + V\hat{\mathbf{f}} \\ &= V\Lambda\hat{\mathbf{y}} + V\hat{\mathbf{f}}\end{aligned}$$

Multiply through by V^{-1} :

$$\frac{d\hat{\mathbf{y}}}{dt} = \Lambda\hat{\mathbf{y}} + \hat{\mathbf{f}}$$

Eigenvalues and ODEs, Analytical Case

$$\frac{d}{dt} \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix} = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix} + \begin{pmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{pmatrix}$$

$$= \begin{pmatrix} \lambda_1 \hat{y}_1 \\ \vdots \\ \lambda_n \hat{y}_n \end{pmatrix} + \begin{pmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{pmatrix}$$

$$\frac{d\hat{y}_i}{dt} = \lambda_i \hat{y}_i + \hat{f}_i, \quad i = 1, \dots, n$$

- We now have n *decoupled* systems.
- We solve these (numerically) in a coupled fashion.
- The behavior, however, is the same as this decoupled system, which is easier to understand.

Discrete Case: Euler Forward Example

$$\frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{\Delta t} = J\mathbf{y}_k + \mathbf{f}_k \quad (h := \Delta t)$$

$$V^{-1} \times \left[\frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{\Delta t} = J\mathbf{y}_k + \mathbf{f}_k \right]$$

$$\implies \frac{\hat{\mathbf{y}}_{k+1} - \hat{\mathbf{y}}_k}{\Delta t} = \Lambda \hat{\mathbf{y}}_k + \hat{\mathbf{f}}_k$$

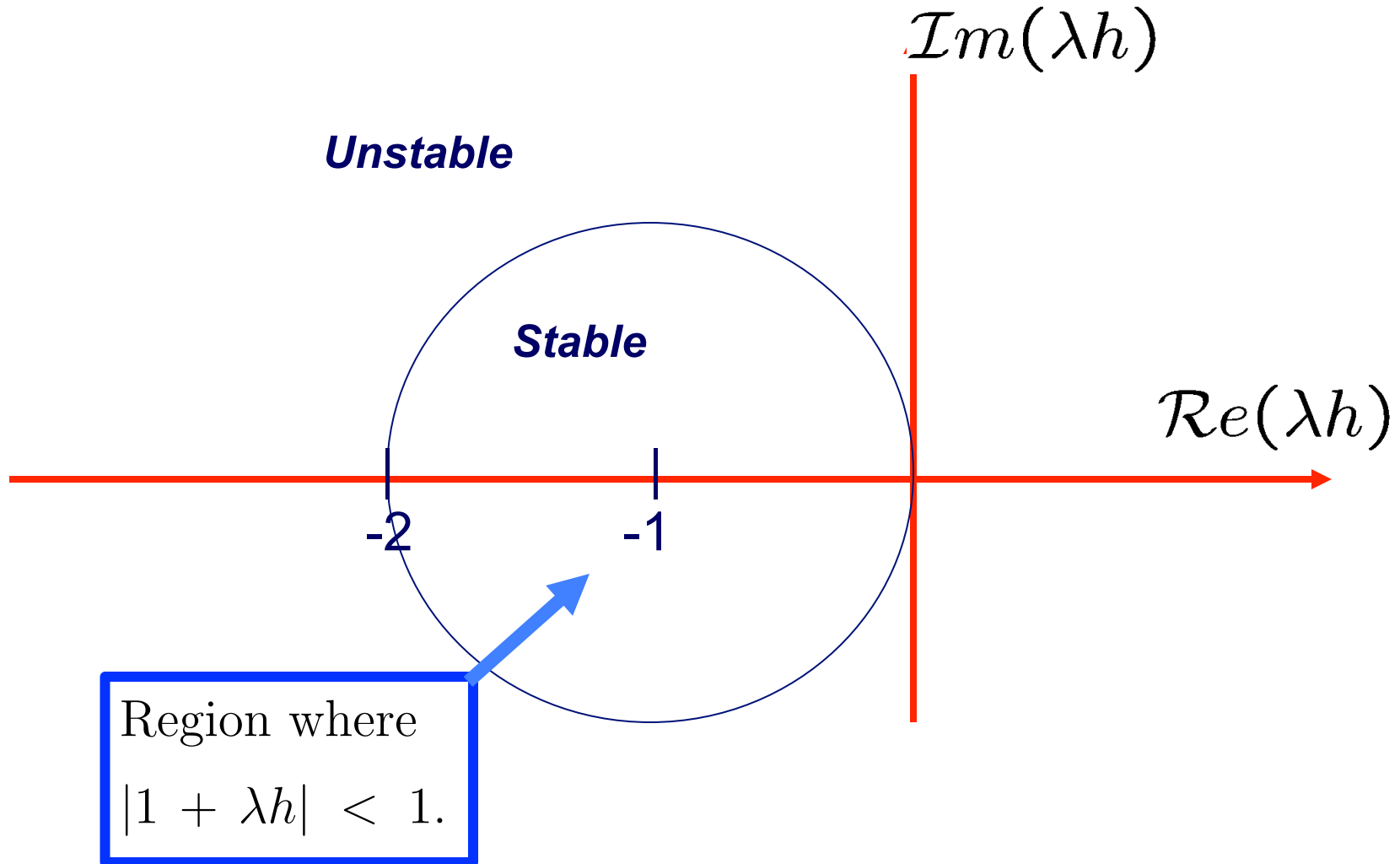
$$\hat{\mathbf{y}}_{k+1} = \hat{\mathbf{y}}_k + \Delta t \Lambda \hat{\mathbf{y}}_k + \Delta t \hat{\mathbf{f}}_k$$

$$\hat{y}_{j,k+1} = (1 + \Delta t \lambda_j) \hat{y}_{j,k} + \Delta t \hat{f}_{j,k}$$

Therefore, forward Euler stability requires

$$|1 + \Delta t \lambda_j| < 1, \quad j = 1, \dots, n$$

Stability Region for Euler's Method



MATLAB EXAMPLE: Euler for $y' = \lambda y$ (ef1.m)

```
%% A simple Euler forward integrator
%
% Typical Usage: h=.01; lambda=3; ef1
%

tfinal = 4; nsteps=ceil(tfinal/h); h=tfinal/nsteps;

x=zeros(nsteps+1,1);t=x;

t=h*(0:nsteps);

hold off; x(1)=1; plot(t(1),x(1),'ko'); hold on;

xe=x(1)*exp(lambda*t); plot(t,xe,'r-')

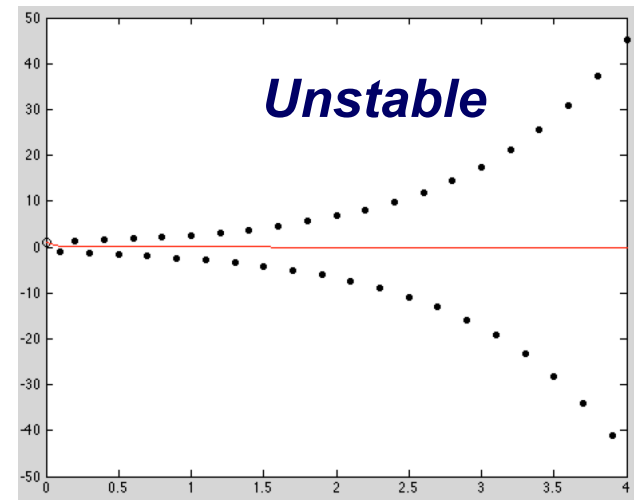
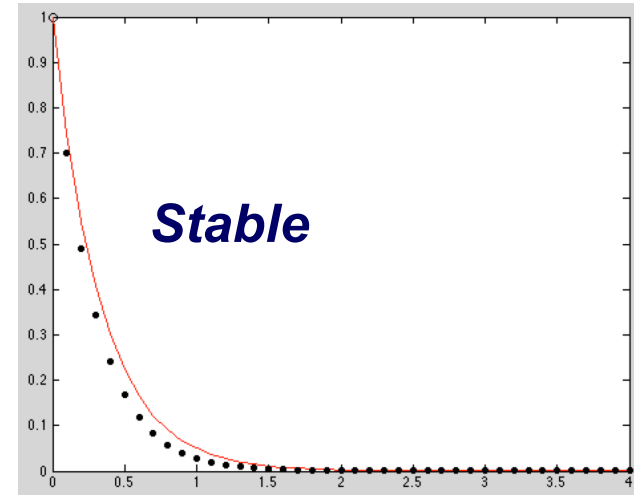
for k=1:nsteps;

    fx = lambda*x(k);

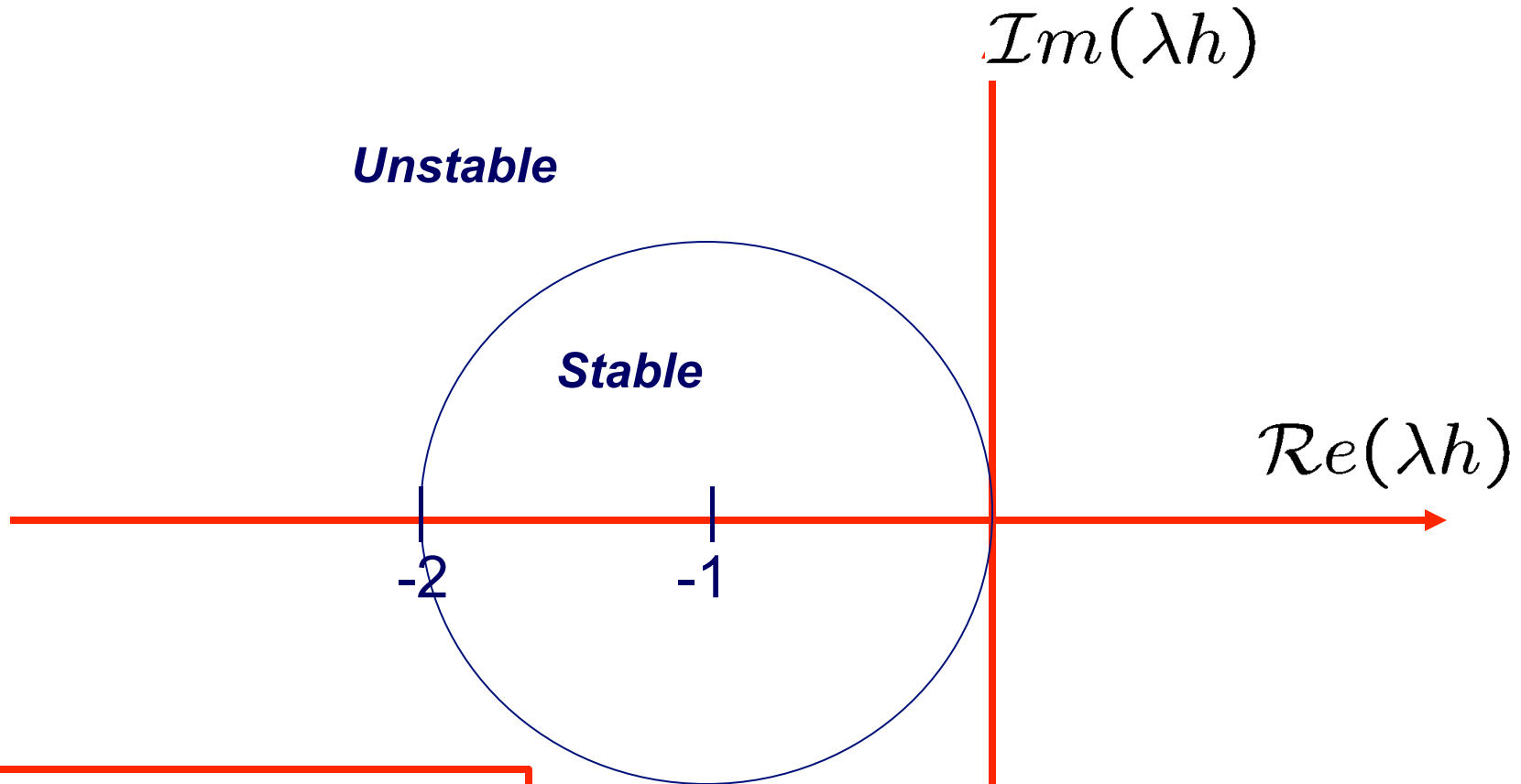
    x(k+1)=x(k) + h*fx;
    t(k+1)=k*h;

    plot(t(k+1),x(k+1),'k. '); drawnow;

end;
```



Stability Region for Euler's Method



Why complex plane?

Recall: Orbit Example

$$\frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = A \mathbf{y}.$$

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y}$$

$$\begin{aligned} |A - \lambda I| &= \begin{vmatrix} -\lambda & -1 \\ 1 & -\lambda \end{vmatrix} \\ &= \lambda^2 + 1 = 0 \end{aligned}$$

$$\lambda = \pm i$$

- ***Even though ODE involves only reals, the behavior can be governed by complex eigenvalues.***

Euler's Method, continued

- If λ is real, then $h\lambda$ must lie in interval $(-2, 0)$, so for $\lambda < 0$, we must have

$$h \leq -\frac{2}{\lambda}$$

for Euler's method to be stable

- *Growth factor* $1 + h\lambda$ agrees with series expansion

$$e^{h\lambda} = 1 + h\lambda + \frac{(h\lambda)^2}{2} + \frac{(h\lambda)^3}{6} + \dots$$

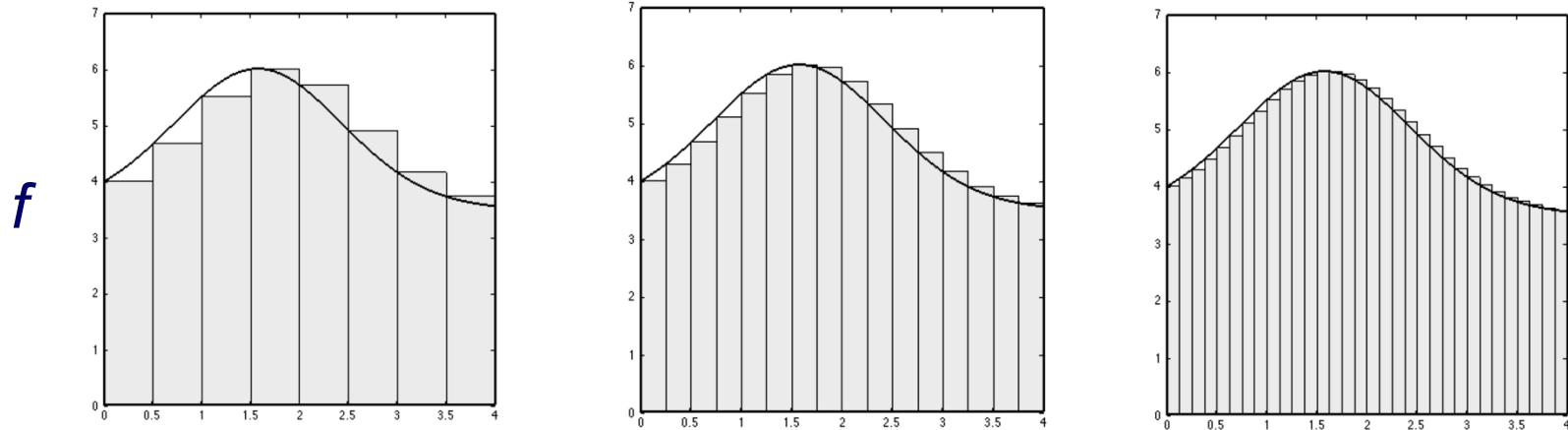
through terms of first order in h , so Euler's method is first-order accurate

LTE is $O(h^2)$!

GTE is $O(h)$!



Relationship between LTE and GTE



$$y_n = y_0 + \int_0^T f(t, y) dt$$

- If $\text{LTE} = O(\Delta t^2)$, then commit $O(\Delta t^2)$ error on each step.
- Interested in final error at time $t = T = n\Delta t$.
- Interested in the final error $e_n := y(t_n) - y_n$ in the limit $n \rightarrow \infty$, $n\Delta t = T$ fixed.
- Nominally, the final error will be proportional to the sum of the local errors,

$$e_n \sim C n \cdot \text{LTE} \sim C n \Delta t^2 \sim C (n\Delta t) \Delta t \sim C T \Delta t$$

- $\text{GTE} \sim \text{LTE} / \Delta t$

Stability of Numerical Methods for ODEs

In general, growth factor depends on

- Numerical method, which determines form of growth factor
- Step size h
- Jacobian J_f , which is determined by particular ODE



Implicit Methods

- Euler's method is *explicit* in that it uses only information at time t_k to advance solution to time t_{k+1}
- This may seem desirable, but Euler's method has rather limited stability region
- Larger stability region can be obtained by using information at time t_{k+1} , which makes method *implicit*
- Simplest example is *backward Euler method*

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

- Method is implicit because we must evaluate \mathbf{f} with argument \mathbf{y}_{k+1} before we know its value



Implicit Methods, continued

- This means that we must solve algebraic equation to determine \mathbf{y}_{k+1}
- Typically, we use iterative method such as Newton's method or fixed-point iteration to solve for \mathbf{y}_{k+1}
- Good starting guess for iteration can be obtained from explicit method, such as Euler's method, or from solution at previous time step

< interactive example >



Implicit Methods, continued

- Given extra trouble and computation in using implicit method, one might wonder why we bother
- Answer is that implicit methods generally have significantly larger stability region than comparable explicit methods
- Increased stability implies we can take (many) fewer steps, assuming that accuracy is not compromised by the larger stepsize, h .



Backward Euler Method

- To determine stability of backward Euler, we apply it to scalar ODE $y' = \lambda y$, obtaining

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

$$(1 - h\lambda)y_{k+1} = y_k$$

$$y_k = \left(\frac{1}{1 - h\lambda} \right)^k y_0$$

- Thus, for backward Euler to be stable we must have

$$\left| \frac{1}{1 - h\lambda} \right| \leq 1$$

which holds for *any* $h > 0$ when $\operatorname{Re}(\lambda) < 0$

- So stability region for backward Euler method includes entire left half of complex plane, or interval $(-\infty, 0)$ if λ is real



Backward Euler Method

- To determine stability of backward Euler, we apply it to scalar ODE $y' = \lambda y$, obtaining

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

$$(1 - h\lambda)y_{k+1} = y_k$$

$$y_k = \left(\frac{1}{1 - h\lambda} \right)^k y_0$$

Growth Factor, G



- Thus, for backward Euler to be stable we must have

$$\left| \frac{1}{1 - h\lambda} \right| \leq 1$$

which holds for *any* $h > 0$ when $\text{Re}(\lambda) < 0$

- So stability region for backward Euler method includes entire left half of complex plane, or interval $(-\infty, 0)$ if λ is real



Backward Euler Method, continued

- Growth factor

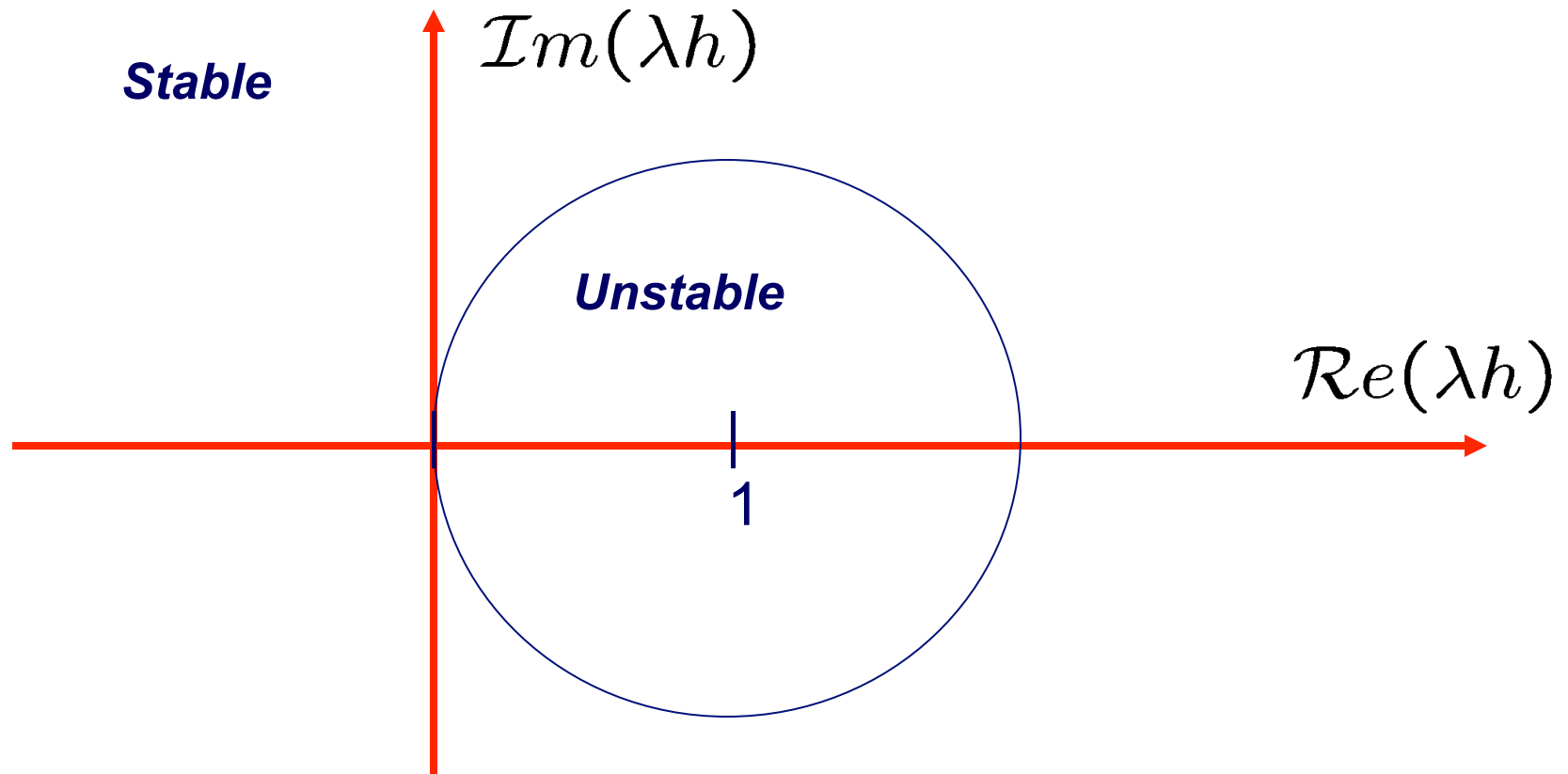
$$\frac{1}{1 - h\lambda} = 1 + h\lambda + (h\lambda)^2 + \dots$$

agrees with expansion for $e^{\lambda h}$ through terms of order h , so backward Euler method is first-order accurate

- Growth factor of backward Euler method for general system of ODEs $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$ is $(\mathbf{I} - h\mathbf{J}_f)^{-1}$, whose spectral radius is less than 1 provided all eigenvalues of $h\mathbf{J}_f$ lie outside circle in complex plane of radius 1 centered at 1
- Thus, stability region of backward Euler for general system of ODEs is entire left half of complex plane



Stability Region for Backward Euler Method



Unconditionally Stable Methods

- Thus, for computing stable solution backward Euler is stable for any positive step size, which means that it is *unconditionally* stable
- Great virtue of unconditionally stable method is that desired accuracy is only constraint on choice of step size
- Thus, we may be able to take much larger steps than for explicit method of comparable order and attain much higher overall efficiency despite requiring more computation per step
- Although backward Euler method is unconditionally stable, its accuracy is only of first order, which severely limits its usefulness



Trapezoid Method

- Higher-order accuracy can be achieved by averaging Euler and backward Euler methods to obtain implicit *trapezoid method*

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

- To determine its stability and accuracy, we apply it to scalar ODE $y' = \lambda y$, obtaining

$$y_{k+1} = y_k + h (\lambda y_k + \lambda y_{k+1}) / 2$$

$$y_k = \left(\frac{1 + h\lambda/2}{1 - h\lambda/2} \right)^k y_0$$

- Method is stable if

$$\left| \frac{1 + h\lambda/2}{1 - h\lambda/2} \right| < 1$$

which holds for any $h > 0$ when $\text{Re}(\lambda) < 0$



Trapezoid Method, continued

- Thus, trapezoid method is unconditionally stable
- Its growth factor

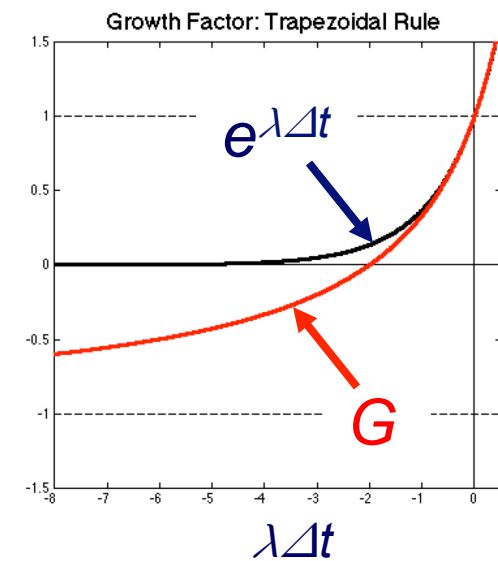
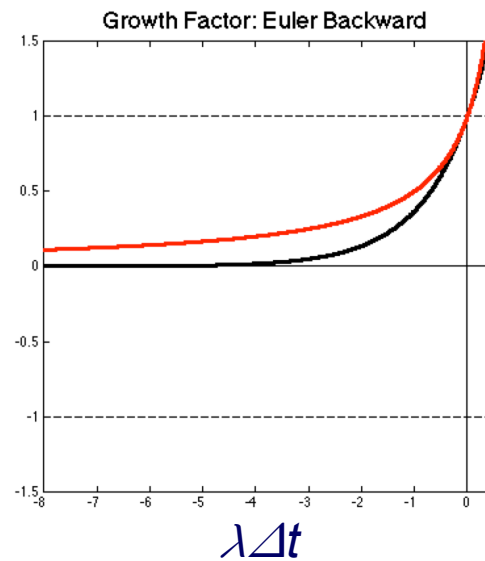
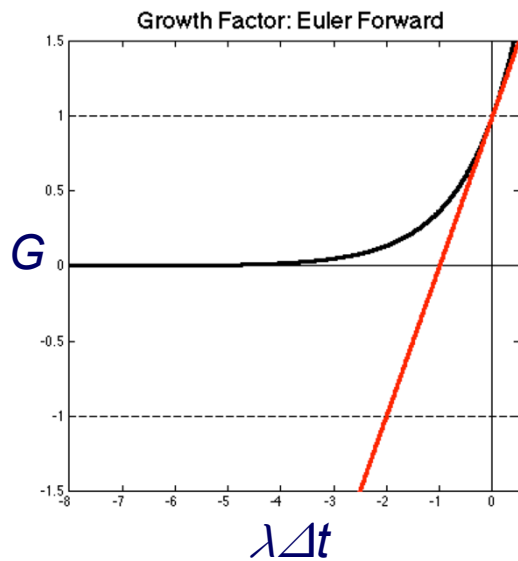
$$\begin{aligned} \frac{1 + h\lambda/2}{1 - h\lambda/2} &= \left(1 + \frac{h\lambda}{2}\right) \left(1 + \frac{h\lambda}{2} + \left(\frac{h\lambda}{2}\right)^2 + \left(\frac{h\lambda}{2}\right)^3 + \dots\right) \\ &= 1 + h\lambda + \frac{(h\lambda)^2}{2} + \frac{(h\lambda)^3}{4} + \dots \end{aligned}$$

agrees with expansion of $e^{h\lambda}$ through terms of order h^2 , so trapezoid method is second-order accurate

- For general system of ODEs $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, trapezoid method has growth factor $(\mathbf{I} + \frac{1}{2}h\mathbf{J}_f)(\mathbf{I} - \frac{1}{2}h\mathbf{J}_f)^{-1}$, whose spectral radius is less than 1 provided eigenvalues of $h\mathbf{J}_f$ lie in left half of complex plane



Growth Factors for Real λ



- ❑ Each growth factor approximates $e^{\lambda\Delta t}$ for $\lambda\Delta t \rightarrow 0$
- ❑ For EF, $|G|$ is not bounded by 1
- ❑ For Trapezoid method, local (small $\lambda\Delta t$) approximation is $O(\lambda\Delta t^2)$, but $|G| \rightarrow -1$ as $\lambda\Delta t \rightarrow -\infty$. [Trapezoid method is not **L-stable**.]
- ❑ BDF2 will give 2nd-order accuracy, stability, and $|G| \rightarrow 0$ as $\lambda\Delta t \rightarrow -\infty$.

Quiz

- For which of the following methods is the growth factor more accurate when $\lambda\Delta t = -0.1$?
 - Euler forward
 - Euler backward
 - Trapezoidal (Crank Nicholson) method

Quiz

- For which of the following methods is the growth factor more accurate when $\lambda\Delta t = -10$?
 - Euler forward
 - Euler backward
 - Trapezoidal (Crank Nicholson) method

Implicit Methods, continued

- We have now seen two examples of implicit methods that are unconditionally stable, but not all implicit methods have this property
- Implicit methods generally have larger stability regions than explicit methods, but allowable step size is not always unlimited
- Implicitness alone is not sufficient to guarantee stability

*Example: backward difference formula
of order 3 or higher*



BDFk Formulas: GTE = $O(h^k)$

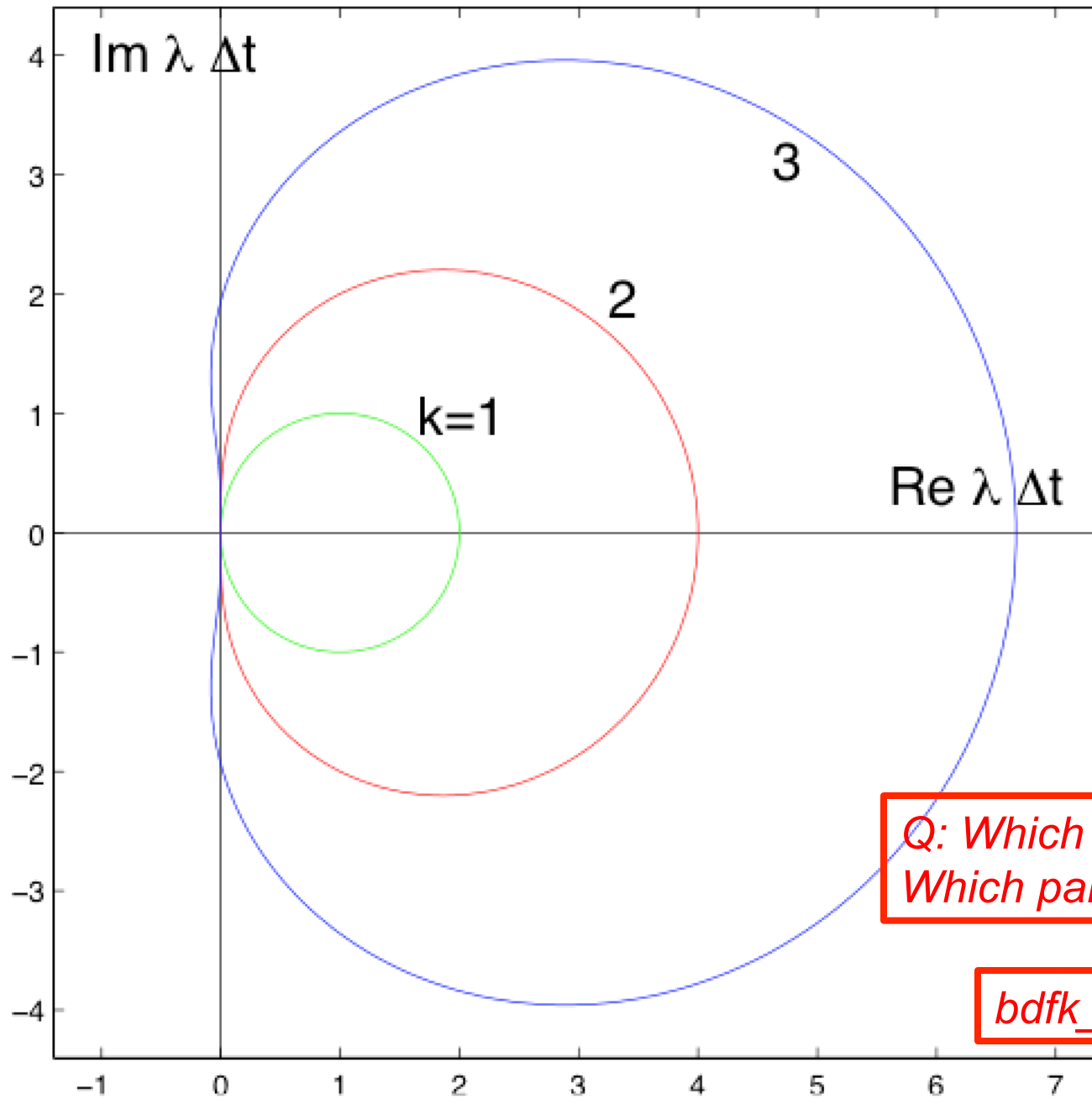
$$\text{BDF1: } \left. \frac{\partial u}{\partial t} \right|_{t^n} = \frac{u^n - u^{n-1}}{\Delta t} + O(\Delta t)$$

$$\text{BDF2: } \left. \frac{\partial u}{\partial t} \right|_{t^n} = \frac{3u^n - 4u^{n-1} + u^{n-2}}{2\Delta t} + O(\Delta t^2)$$

$$\text{BDF3: } \left. \frac{\partial u}{\partial t} \right|_{t^n} = \frac{11u^n - 18u^{n-1} + 9u^{n-2} - 2u^{n-3}}{6\Delta t} + O(\Delta t^3).$$

- ❑ Unlike the trapezoidal rule, these methods are L-stable:
 - ❑ $|G| \rightarrow 0$ as $\lambda \Delta t \rightarrow -\infty$
- ❑ k-th order accurate
- ❑ Implicit
- ❑ Unconditionally stable only for $k \leq 2$ (here, $k :=$ order of method)
- ❑ Multi-step: require data from previous timesteps

BDFk Neutral Stability Curve



Implicit Orbit Example

```
% BDFk-Orbit: Typical Usage: dt=.1; bdfk_orbit

T = 2*pi; n=ceil(T/dt); dt = T/n; n=10*n; % Tfinal and dt; 10 orbits

A = [ 0 -1 ; 1 0 ]; I=eye(2);

y0 = [ 1 ; 0 ]; % INITIAL CONDITION

y=y0; y2=y; y1=y; y3=y;

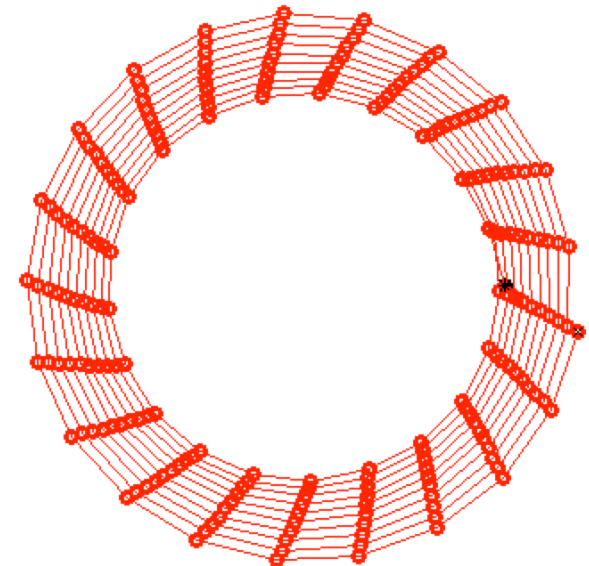
xk=zeros(n+1,1); yk=xk;
for k=1:n;

    xk(k)=y(1); yk(k)=y(2);
    y3=y2; y2=y1;y1=y;

    if k==1; % BDF 1 for 1st step
        E = I - dt*A;
        y = E\y1;
    elseif k==2; % BDF 2 for 2nd step
        E = 1.5*I - dt*A;
        y = E\((4*y1-y2)/2);
    else % BDF 3 for step > 2
        E = (11/6)*I - dt*A;
        y = E\((18*y1-9*y2+2*y3)/6);
    end

end;

xk(end)=y(1); yk(end)=y(2);
plot(xk,yk,'r-'); axis equal;hold on
plot(xk(1),yk(1),'k*'); axis equal; hold on
plot(y(1),y(2),'kx'); axis equal; hold on
```



Explicit High-Order Methods

- ❑ High-order explicit methods are of interest for several reasons:
 - ❑ Lower cost per step than implicit (but possibly many steps if system has disparate timescales, i.e., is stiff --- spring-mass example).
 - ❑ More accuracy
 - ❑ For $k > 2$, encompass part of the imaginary axis near zero, so stable for systems having purely imaginary eigenvalues, ***provided h is sufficiently small.***
 - ❑ We'll look at three classes of high-order explicit methods:
 - ❑ BDFk / Ext k
 - ❑ kth-order Adams Bashforth
 - ❑ Runge-Kutta methods
 - ❑ Each has pros and cons...

Higher-Order Explicit Timesteppers: BDFk/EXTk

- Idea: evaluate left-hand and right-hand sides at t_{k+1} to accuracy $O(\Delta t^k)$.

$$\left. \frac{dy}{dt} \right|_{t_{k+1}} = f(t, y)|_{t_{k+1}}$$

- Can treat term on the right via k th-order extrapolation.
- For example, for $k = 2$,

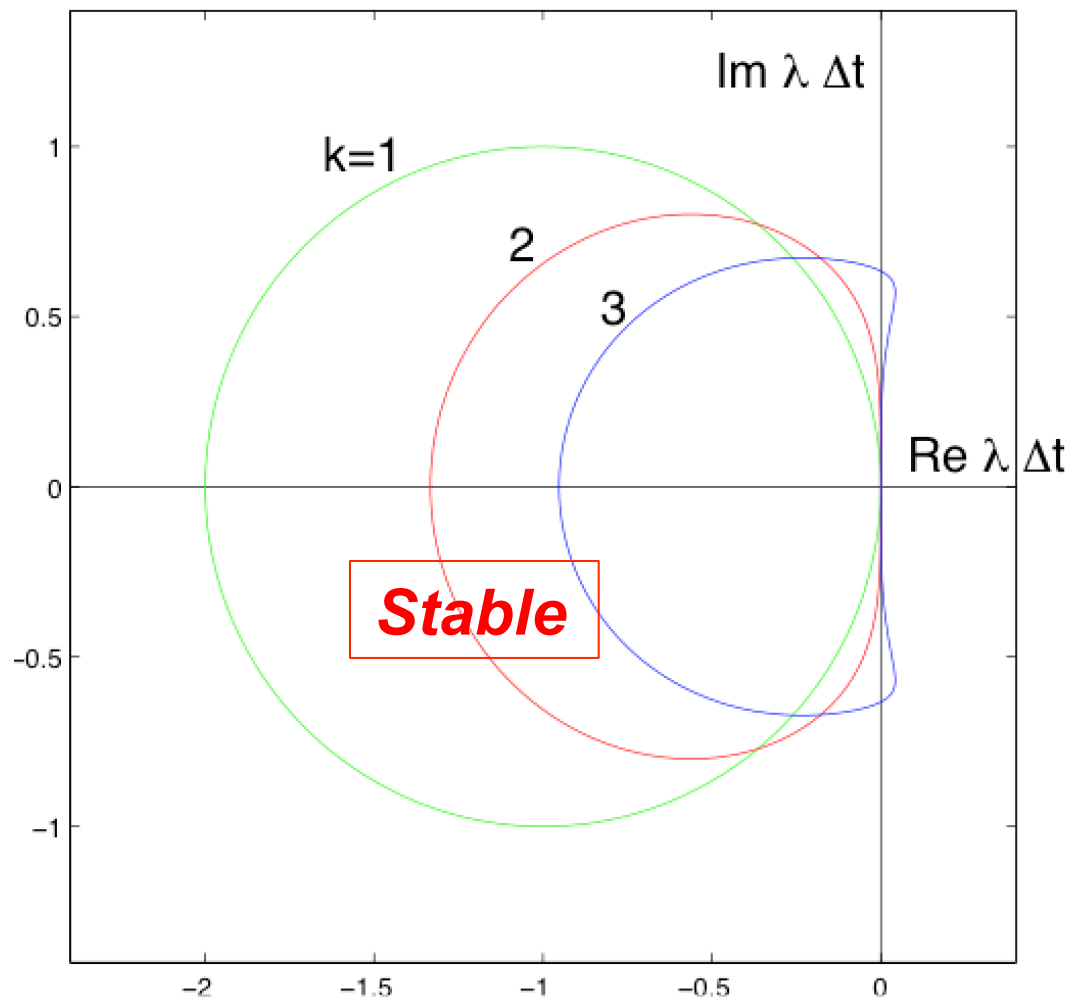
$$\frac{3y_{k+1} - 4y_k + y_{k-1}}{2\Delta t} + O(\Delta t^2) = 2f_k - f_{k-1} + O(\Delta t^2)$$

- Solve for y_{k+1} in terms of known quantities on the right:

$$y_{k+1} = \frac{2}{3} \left[\frac{4y_k - y_{k-1}}{2} + \Delta t(2f_k - f_{k-1}) \right] + O(\Delta t^3)$$

- Note that LTE is $O(\Delta t^3)$, GTE= $O(\Delta t^2)$.

BDF/EXtk Neutral Stability Curve



- Here we see that the $k=3$ curve encompasses part of the imaginary axis near the origin of the $\lambda\Delta t$ plane, which is important for stability of non-dissipative systems.

Higher-Order Explicit Timesteppers: kth-order Adams-Bashforth

- Adams-Bashforth methods are a somewhat simpler alternative to BDFk/EXTk.
- Time advancement via integration:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \int_{t_k}^{t_{k+1}} \mathbf{f}(t, \mathbf{y}) dt$$

- AB1:

$$\int_{t_k}^{t_{k+1}} f(t, \mathbf{y}) dt = h_k f_k + O(h^2)$$

- AB2:

$$\begin{aligned} \int_{t_k}^{t_{k+1}} \mathbf{f}(t, \mathbf{y}) dt &= h_k \mathbf{f}_k + \frac{h_k^2}{2} \left[\frac{\mathbf{f}_k - \mathbf{f}_{k-1}}{h_{k-1}} \right] + O(h^3) \\ &= h \left(\frac{3}{2} \mathbf{f}_k - \frac{1}{2} \mathbf{f}_{k-1} \right) + O(h^3) \text{ (if } h \text{ is constant)} \end{aligned}$$

- AB3:

$$\int_{t_k}^{t_{k+1}} \mathbf{f}(t, \mathbf{y}) dt = h \left(\frac{23}{12} \mathbf{f}_k - \frac{16}{12} \mathbf{f}_{k-1} + \frac{5}{12} \mathbf{f}_{k-2} \right) + O(h^4) \text{ (if } h \text{ is constant)}$$

- LTE for AB m is $O(h^{m+1})$. GTE for AB m is $O(h^m)$.

Stability of Various Timesteppers

- Derived from model problem $\frac{du}{dt} = \lambda u$
- Stability regions shown in the $\lambda\Delta t$ plane (stable *inside* the curves)

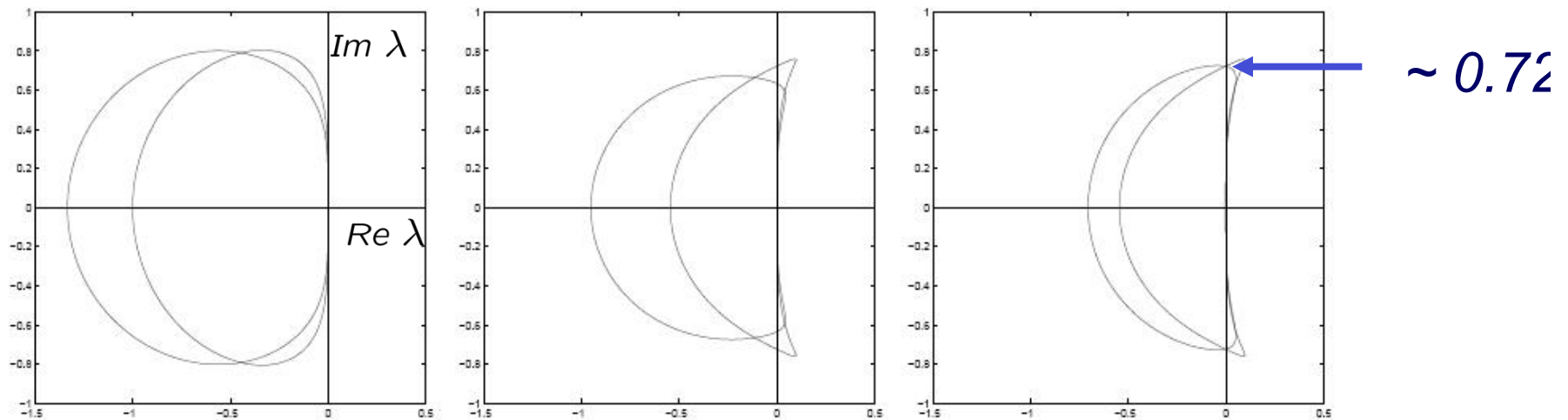


Figure 1: Stability regions for (left) AB2 and BDF2/EXT2, (center) AB3 and BDF3/EXT3, and (right) AB3 and BDF2/EXT2a.

- To make effective use of this plot, we need to know something about the eigenvalues λ of the Jacobian.
- But first, How are these plots generated?

Determining the Neutral-Stability Curve

Consider BDF2/EXT2, and apply it to $\frac{du}{dt} = \lambda u$:

$$3u^m - 4u^{m-1} + u^{m-2} = 2\lambda\Delta t \left(2u^{m-1} - u^{m-2}\right).$$

Seek solutions of the form $u^m = (z)^m$, $z \in C$:

$$3z^m - 4z^{m-1} + z^{m-2} = 2\lambda\Delta t \left(2z^{m-1} - z^{m-2}\right).$$

$$3z^2 - 4z + 1 = 2\lambda\Delta t (2z - 1).$$

Set $z = e^{i\theta}$, $\theta \in [0, 2\pi]$, and solve for $\lambda\Delta t$:

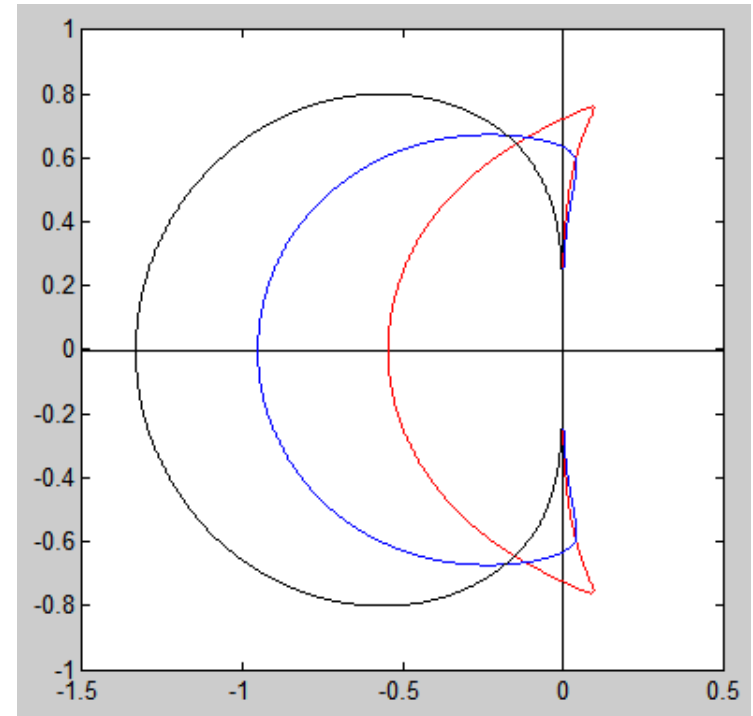
$$\lambda\Delta t = \frac{3e^{i2\theta} - 4e^{i\theta} + 1}{2(2e^{i\theta} - 1)}.$$

Matlab Code: stab.m

```
yymax=1; ep=1.e-13; yaxis=[-yymax*ii yymax*ii]'; % Plot axes
xaxis=[-2.0+ep*ii 2.0+ep*ii]';
hold off; plot (yaxis,'k-'); hold on; plot (xaxis,'k-');
axis square; axis([-yymax-.5 yymax-.5 -yymax yymax]);

ii=sqrt(-1); th=0:.001:2*pi; th=th'; ith=ii*th; ei=exp(ith);
E = [ ei 1+0*ei 1./ei 1./(ei.*ei) 1./(ei.*ei.*ei)];
```

```
ab0 = [1 0.0 0.0 0. 0.]';
ab1 = [0 1.0 0.0 0. 0.]';
ab2 = [0 1.5 -.5 0. 0.]';
ab3 = [0 23./12. -16./12. 5./12. 0.]';
bdf1 = (([ 1. -1. 0. 0. 0.])/1.)';
bdf2 = (([ 3. -4. 1. 0. 0.])/2.)';
bdf3 = (([11. -18. 9. -2. 0.])/6.)';
exm = [1 0 0 0 0]';
ex1 = [0 1 0 0 0]';
ex2 = [0 2 -1 0 0]';
ex3 = [0 3 -3 1 0]';
du = [1. -1. 0. 0. 0.]';
```

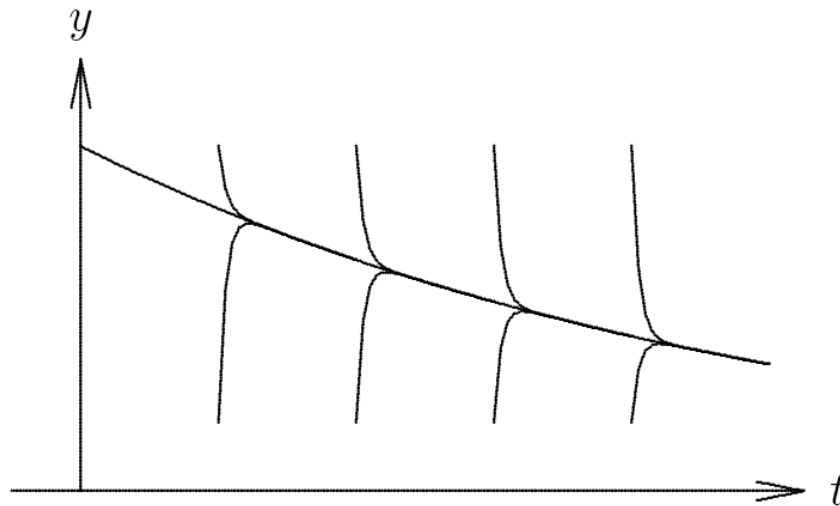


```
ldtab3 =(E*du)./(E*ab3); plot (ldtab3 ,'r-'); % AB3
bdf3ex3=(E*bdf3)./(E*ex3); plot (bdf3ex3,'b-'); % BDF3/EXT3
bdf2ex2=(E*bdf2)./(E*ex2); plot (bdf2ex2,'k-'); % BDF2/EXT2
```



Stiff Differential Equations

- Asymptotically stable solutions converge with time, and this has favorable property of damping errors in numerical solution
- But if convergence of solutions is too rapid, then difficulties of different type may arise
- Such ODE is said to be *stiff*



Stiff System Examples

- Spring-mass system

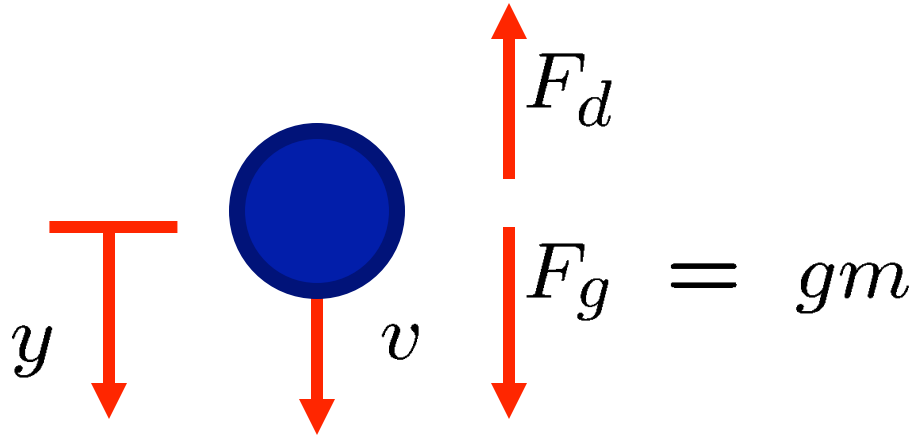
$$M\mathbf{y}'' = -K\mathbf{y}$$

- Nonlinear ODE

$$y' = -1000y + \cos(y^3)$$

- Falling Particle in a viscous fluid

Stiffness Example: Drag on a Falling Particle



$$\begin{pmatrix} \frac{dy}{dt} \\ \frac{dv}{dt} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -s \end{bmatrix} \begin{pmatrix} y \\ v \end{pmatrix} + \begin{pmatrix} 0 \\ g \end{pmatrix}$$

- Stokes drag for small particles:

$$F_d = -6\pi\mu r v$$
$$\frac{1}{m}F_d = -\frac{6\pi\mu r v}{\frac{4}{3}\pi r^3 \rho_p} = -\frac{18\mu}{\rho_p d^2} v =: -s v,$$

where $d = 2r =$ particle diameter.

Stiffness Example: Drag on a Falling Particle

- Consider acceleration of falling particle,

$$\begin{aligned} y &= \text{position} \\ v &= \frac{dy}{dt} = \text{velocity} \\ \frac{1}{m} F_{\text{net}} &= \frac{dv}{dt} = \text{acceleration} \end{aligned}$$

- Equation of motion:

$$\begin{pmatrix} \frac{dy}{dt} \\ \frac{dv}{dt} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -s \end{bmatrix} \begin{pmatrix} y \\ v \end{pmatrix} + \begin{pmatrix} 0 \\ g \end{pmatrix}$$

Stiffness Example: Drag on a Falling Particle

- Equation of motion:

$$\begin{pmatrix} \frac{dy}{dt} \\ \frac{dv}{dt} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -s \end{bmatrix} \begin{pmatrix} y \\ v \end{pmatrix} + \begin{pmatrix} 0 \\ g \end{pmatrix}$$

- Stokes drag for small particles:

$$F_d = -6\pi\mu r v$$
$$\frac{1}{m}F_d = -\frac{6\pi\mu r v}{\frac{4}{3}\pi r^3 \rho_p} = -\frac{18\mu}{\rho_p d^2} v =: -s v,$$

where $d = 2r =$ particle diameter.

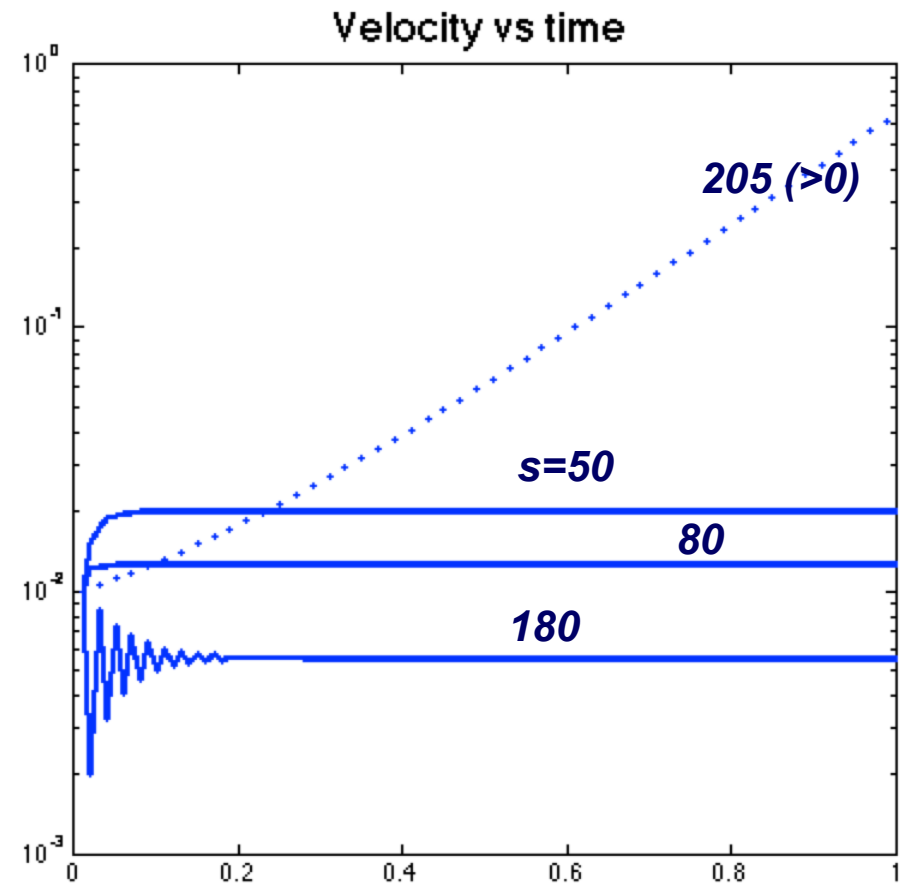
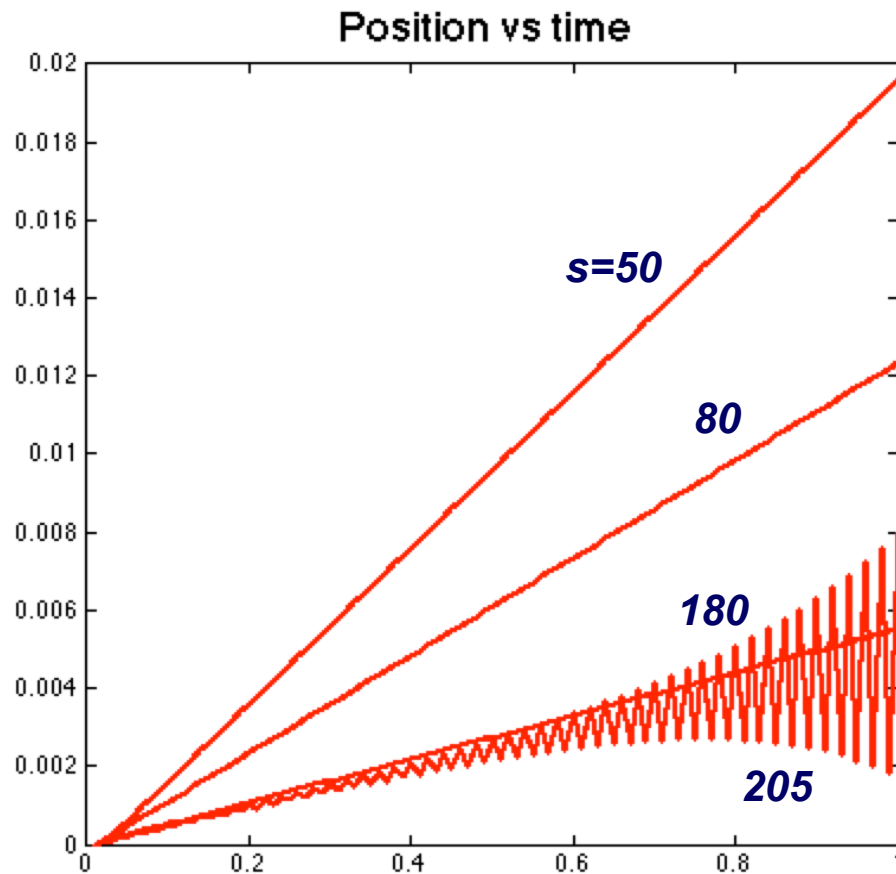
- Eigenvalues of the system:

$$0 = |J - I\lambda| = \begin{vmatrix} -\lambda & 1 \\ 0 & -s - \lambda \end{vmatrix} = \lambda(\lambda + s)$$

$$\lambda = 0, \quad \boxed{\lambda = -s.}$$

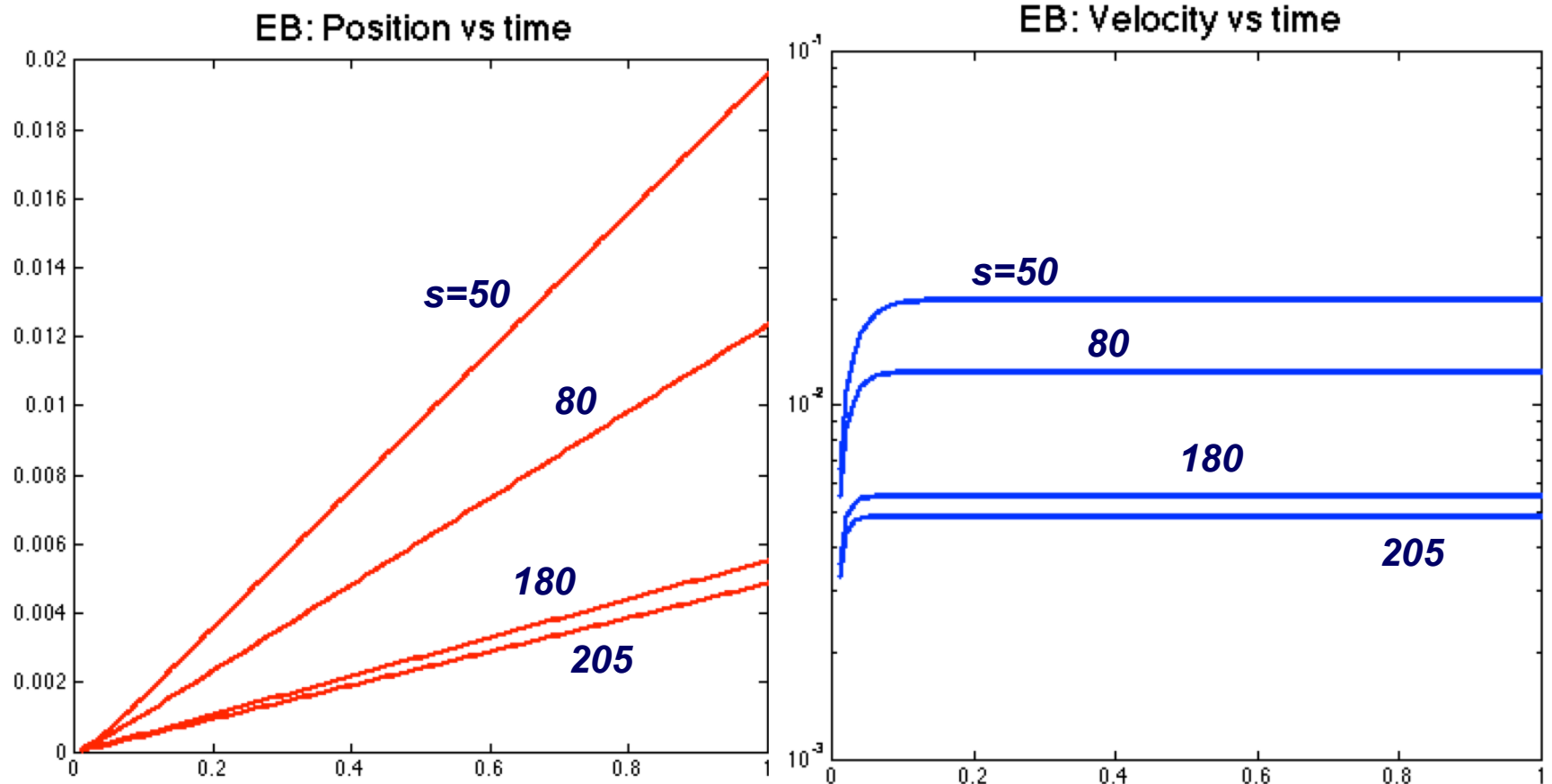
Falling Particle Using Euler Forward

`stokeseb.m`



Q: What must the h value be, in this case?

Falling Particle Using Euler Backward



Note: Still only $O(h)$ accurate. Trapezoid or BDF2 would be $O(h^2)$

Stiffness Example: Drag on a Particle in Orbit

- Equation of motion:

`stokes_orbit.m`

$$\begin{pmatrix} \frac{dy}{dt} \\ \frac{dv}{dt} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -s \end{bmatrix} \begin{pmatrix} y \\ v \end{pmatrix} + \begin{pmatrix} 0 \\ g \end{pmatrix}$$

- Stokes drag for small particles:

$$F_d = -6\pi\mu r v$$
$$\frac{1}{m}F_d = -\frac{6\pi\mu r v}{\frac{4}{3}\pi r^3 \rho_p} = -\frac{18\mu}{\rho_p d^2} v =: -s v,$$

where $d = 2r =$ particle diameter.

- Eigenvalues of the system:

$$0 = |J - I\lambda| = \begin{vmatrix} -\lambda & 1 \\ 0 & -s - \lambda \end{vmatrix} = \lambda(\lambda + s)$$

$$\lambda = 0, \quad \lambda = -s.$$

Stiffness Example: Drag on a Particle in Orbit

`stokes_orbit.m`

```
% EF,EB,CN integrator for particle subject to viscous plane rotation

% Usage: h=.01;s=500; int=1; stokes_orbit
%         h=.01;s=195; int=0; stokes_orbit; % Pathological EF case
%         h=.01;s=1e4; int=2; stokes_orbit; % Pathological CN case
%         h=.01;s=1e4; int=1; stokes_orbit; % Pathological EB case (spiral in)

% y1 = x position
% y2 = y position
% y3 = x velocity
% y4 = y velocity

%      y1 = /          1      \ y1
%      d y2 = |          1      | y2
%      --      |          |      + | 0  \
%      dt y3 = |      -s      | y3   | u  |
%      y4 = \          -s / y4   \ v /

% so

%      y1 = /          1      \ y1
%      d y2 = |          1      | y2
%      --      |          |      |
%      dt y3 = |      -s      -s | y3
%      y4 = \s          -s / y4

close all; format compact; format longe
tfinal = 22.75*2*pi; nsteps=ceil(tfinal/h); h=tfinal/nsteps;
tfinal = 2.75*2*pi; nsteps=ceil(tfinal/h); h=tfinal/nsteps;

J = zeros(4,4);
J = [ 0 0 1 0 ;
      0 0 0 1 ;
      0 -s -s 0 ;
      s 0 0 -s];
y0=[1 ; 0 ; 0; 0]; y=y0;

[V,d]=eig(J); d=diag(d)

if int==0; Hi= [ eye(4) + h*J ]; end; %% EF
if int==1; H = [ eye(4) - h*J ]; Hi=inv(H); end; %% EB
if int==2; H1= [ eye(4) - h*J/2]; Hr=[eye(4) + h*J/2]; Hi=H1\Hr; end; %% CN

yy=zeros(4,nsteps); tt=zeros(nsteps,1);

hold off
for k=1:nsteps;
    y = Hi*y;
    yy(:,k) = y; tt(k) = k*h;
% xp=yy(1,k); yp=yy(2,k); u=yy(3,k); v=yy(4,k);
end;

xp=yy(1,:); yp=yy(2,:); u=yy(3,:); v=yy(4,:);
plot(xp,yp,'r-',u+3.0,v,'b-', 'linewidth',0.3); axis equal;
```

Semi-Implicit Methods for Stiff ODEs

- Recall, for general system of ODEs,

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

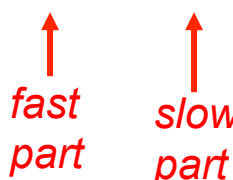
growth factor for EB is spectral radius ($\max |\lambda_j|$) of $(I - hJ)^{-1}$

- Recall our basic timesteppers for model problem $\mathbf{y}' = J\mathbf{y}$:
 - EF: $\mathbf{y}_{k+1} = (I + hJ)\mathbf{y}_k$
 - EB: $(I - hJ)\mathbf{y}_{k+1} = \mathbf{y}_k$
 - Trap: $(I - \frac{h}{2}J)\mathbf{y}_{k+1} = (1 + \frac{h}{2}J)\mathbf{y}_k$
- We see that the trapezoidal rule is just a splitting of the Jacobian J .
- We can effect other splittings to get stable schemes that are easier to solve than the fully-implicit systems (where $J = J(\mathbf{y}_{k+1})$, in general).

Semi-Implicit Methods for Stiff ODEs

- For stiff problems can split Jacobian into fast and slow parts

$$J = J_f + J_s$$


fast part *slow part*

$$(I - hJ_f)\mathbf{y}_{k+1} = (I + hJ_s)\mathbf{y}_k$$

- Growth factor is spectral radius

$$\rho [(I - hJ_f)^{-1}(I + hJ_s)]$$

- Often, J_f can be linear and diagonal or a linearization of the nonlinear system operator.
- J_s , treated explicitly, can be nonlinear, nonlocal, nonsymmetric, etc.

Semi-Implicit Methods for Stiff ODEs

- Method can be extended to high-order using BDF $_q$ /EXT $_q$.
- For example, a 2nd-order BDF/EXT scheme would be:

$$\begin{aligned}\frac{d\mathbf{y}}{dt} &= \frac{3\mathbf{y}_{k+1} - 4\mathbf{y}_k + \mathbf{y}_{k-1}}{2h} + O(h^2) \\ &= J_f \mathbf{y}_{k+1} + (2q_k - q_{k-1}) + O(h^2)\end{aligned}$$

with $q_{k-j} := J_s(t_{k-j}, \mathbf{y}_{k-j})\mathbf{y}_{k-j}$.

- One can rearrange to solve for \mathbf{y}_{k+1} . The system is of the form

$$\left(3I + \frac{2h}{3}J_f\right)\mathbf{y}_{k+1} = \mathbf{g}$$

- This scheme can be relatively stable and 2nd-order accurate.

Example

- Consider $\mathbf{F} = m\mathbf{a} = m\ddot{\mathbf{x}}$, with

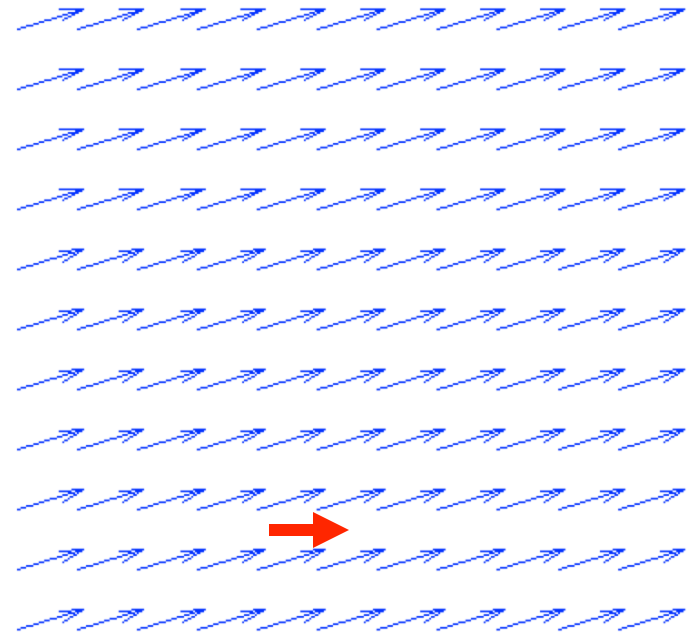
$$m\ddot{\mathbf{x}} = c(\mathbf{u} - \mathbf{v})$$

$\mathbf{u} :=$ background flow field

$\mathbf{v} := \dot{\mathbf{x}} =$ particle velocity

$m :=$ mass of particle

$c :=$ a constant proportional to the fluid viscosity.



- First-order ODE for velocity:

$$\begin{aligned} m\dot{\mathbf{v}} &= -c\mathbf{v} + c\mathbf{u} \\ \dot{\mathbf{v}} &= -\frac{c}{m}\mathbf{v} + \frac{c}{m}\mathbf{u}. \end{aligned}$$

- If $\mathbf{u} = \text{constant}$, then $\dot{\mathbf{u}} = 0$, so

$$\dot{\mathbf{v}} - \dot{\mathbf{u}} = \frac{d}{dt}(\mathbf{v} - \mathbf{u}) = -\frac{c}{m}(\mathbf{v} - \mathbf{u}),$$

which has the solution

$$\begin{aligned} (\mathbf{v} - \mathbf{u}) &= (\mathbf{v} - \mathbf{u})_0 e^{-st} \\ \implies \mathbf{v} &= \mathbf{u} + (\mathbf{v} - \mathbf{u})_0 e^{-st}, \end{aligned}$$

where

$$s := \frac{c}{m}.$$

- Notice that for large c or relatively small m the particle velocity rapidly converges to the background velocity.
- (Think of a dust particle carried by the wind.)

- The rate of convergence is governed by the eigenvalue for this problem

$$\lambda = -s = -\frac{c}{m},$$

which can be large.

- If we also want to integrate the velocity to get position, we have the system of equations

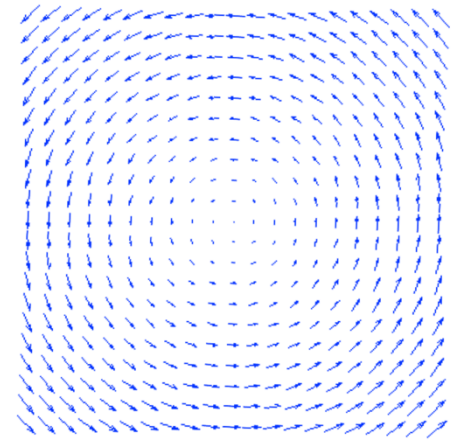
$$\frac{d}{dt} \begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -s \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ s\mathbf{u} \end{pmatrix},$$

which is potentially *stiff* if $s \gg 1$.

- It's a bit odd that large values of s make this problem difficult.
- A large value of s means that, after time $t \approx 1/s \ll 1$, we have $\mathbf{v} \approx \mathbf{u}$.

- Let's extend this example in one more way.
- Consider a rotating background flow:

$$\mathbf{u} = \begin{pmatrix} -y \\ x \end{pmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \mathbf{x}.$$



- The particle motion is governed by

$$\frac{m}{c} \ddot{\mathbf{x}} = \mathbf{u} - \dot{\mathbf{x}},$$

$$\implies \frac{m}{c} \ddot{\mathbf{x}} + \dot{\mathbf{x}} - A\mathbf{x} = \mathbf{0}.$$

- If $\frac{m}{c} = 0$ then we have a simple orbit problem governed by the imaginary eigenvalues ($\lambda = \pm i$) of A .
- Moreover, we know that the tendency of this equation when $\frac{m}{c} \ll 1$ is for the velocity $\dot{\mathbf{x}} \rightarrow \mathbf{u}$, very rapidly, and that this process is associated with large negative eigenvalues of size $O(\frac{c}{m})$.

Returning to our orbit problem

- What happens to the particle in the rotating field?
- What do we expect to happen physically?
- What does this say about the eigenvalues?

Runge-Kutta Methods

- *Runge-Kutta methods* are single-step methods similar in motivation to Taylor series methods, but they do not require computation of higher derivatives
- Instead, Runge-Kutta methods simulate effect of higher derivatives by evaluating f several times between t_k and t_{k+1}
- Simplest example is second-order *Heun's method*

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h_k}{2} (\mathbf{k}_1 + \mathbf{k}_2)$$

where

Essentially, "explicit" trapezoidal rule.

$$\mathbf{k}_1 = \mathbf{f}(t_k, \mathbf{y}_k)$$

$$\mathbf{k}_2 = \mathbf{f}(t_k + h_k, \mathbf{y}_k + h_k \mathbf{k}_1)$$



Runge-Kutta Methods, continued

- Heun's method is analogous to implicit trapezoid method, but remains explicit by using Euler prediction $\mathbf{y}_k + h_k \mathbf{k}_1$ instead of $\mathbf{y}(t_{k+1})$ in evaluating \mathbf{f} at t_{k+1}
- Best known Runge-Kutta method is classical fourth-order scheme

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h_k}{6} (\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

where

$$\mathbf{k}_1 = \mathbf{f}(t_k, \mathbf{y}_k)$$

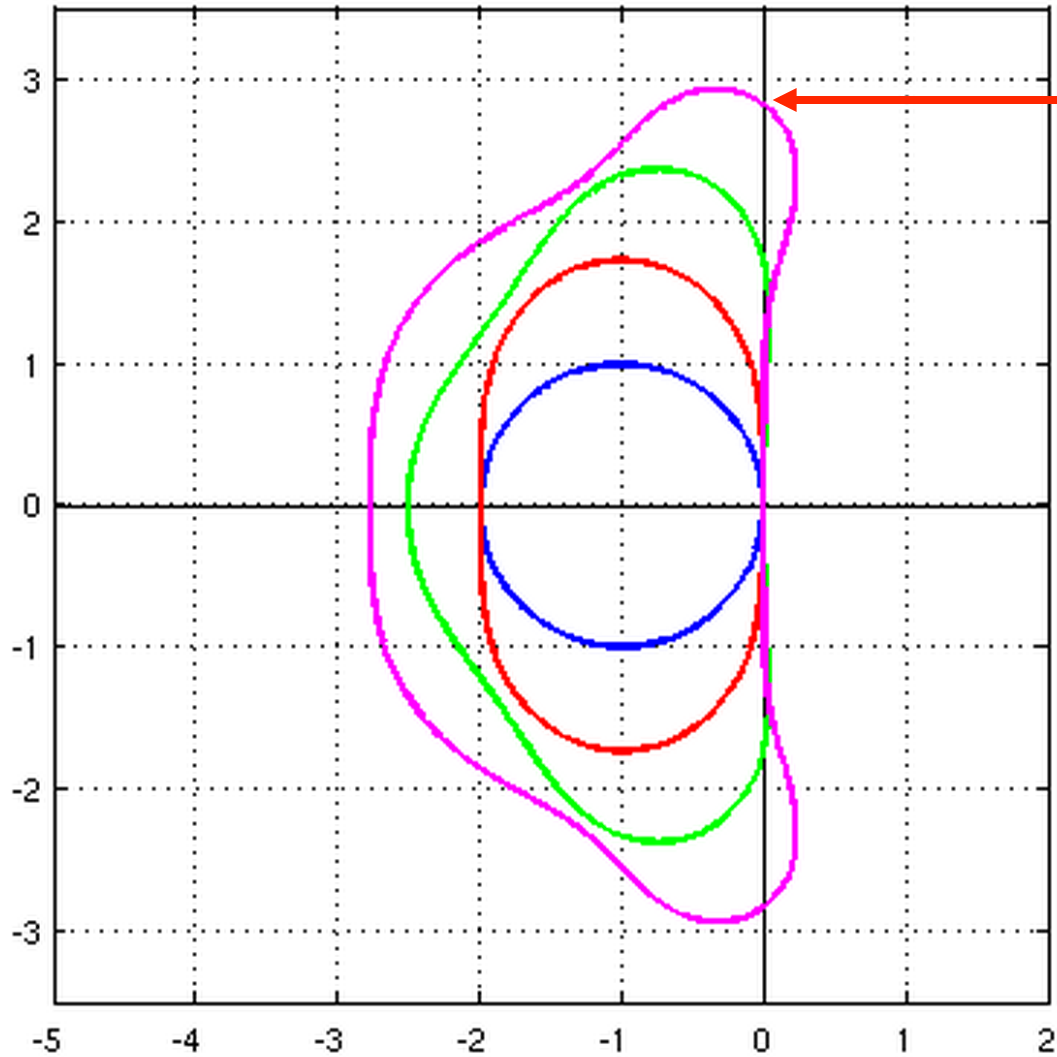
$$\mathbf{k}_2 = \mathbf{f}(t_k + h_k/2, \mathbf{y}_k + (h_k/2)\mathbf{k}_1)$$

$$\mathbf{k}_3 = \mathbf{f}(t_k + h_k/2, \mathbf{y}_k + (h_k/2)\mathbf{k}_2)$$

$$\mathbf{k}_4 = \mathbf{f}(t_k + h_k, \mathbf{y}_k + h_k\mathbf{k}_3)$$



Stability Regions for RK1—4



- ~ 2.828
- RK4 stability region on imaginary axis extends about 4x higher than for AB3 or BDF3/EXT3
- Cost is 4 function evaluations per step instead of 1
- Method is 4th order
- Method is self-starting (good for variable timestep)

Derivation of Stability Region

- Begin with $f(y) = \lambda y$.
- Expand all terms in the time advancement scheme.
- Gives the growth factor G .
- Set $G = e^{i\theta}$ and solve for λh .

Runge-Kutta Methods, continued

- To proceed to time t_{k+1} , Runge-Kutta methods require no history of solution prior to time t_k , which makes them *self-starting* at beginning of integration, and also makes it easy to change step size during integration
- These facts also make Runge-Kutta methods relatively easy to program, which accounts in part for their popularity
- Unfortunately, classical Runge-Kutta methods provide no error estimate on which to base choice of step size



Runge-Kutta Methods, continued

- Fehlberg devised *embedded Runge-Kutta* method that uses six function evaluations per step to produce both fifth-order and fourth-order estimates of solution, whose difference provides estimate for local error
- Another embedded Runge-Kutta method is due to Dormand and Prince
- This approach has led to automatic Runge-Kutta solvers that are effective for many problems, but which are relatively inefficient for stiff problems or when very high accuracy is required
- It is possible, however, to define *implicit* Runge-Kutta methods with superior stability properties that are suitable for solving stiff ODEs



Time Step Selection

- ❑ Assuming Δt satisfies the stability criteria, can also choose Δt based on accuracy by estimating the LTE at each step.
 - ❑ Common way to estimate with, say, RK4 scheme, is to take a step with size Δt and another pair of steps with size $\Delta t/2$.
 - ❑ The difference gives an estimate of LTE (for step size $\Delta t/2$).
 - ❑ If $GTE \sim LTE * T / \Delta t$, and $LTE \sim C \Delta t^5$, solve for Δt such that you will realize the desired final error.
- ❑ Self-starting (i.e., multistage) methods such as RK are well-suited to this strategy.

Summary of Methods / Properties

- Multistep methods of order > 2 require special starting procedures
- Multistage (e.g., RK q) methods are self-starting and easy to change stepsize h .
- Multistage methods are attractive for automated stepsize selection.
- Be sure to understand the stability diagrams of these methods.
 - Left, or right side of complex λh plane?
 - Does it include Im. axis? If so, where does it cut?
 - Where does it cut the real axis?

Method	Implementation	LTE	GTE	Comments
EF	$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}_k$	$O(h^2)$	$O(h)$	explicit
EB	$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}_{k+1}$	$O(h^2)$	$O(h)$	implicit/stable
Trap	$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{2}(\mathbf{f}_k + \mathbf{f}_{k+1})$	$O(h^3)$	$O(h^2)$	implicit/stable (but <i>not</i> L-stable)
BDF q	interpolation of \mathbf{y}_{k-j}	$O(h^{q+1})$	$O(h^q)$	multistep, implicit, stable for $q < 3$
BDF q /EXT q	interpolation of $\mathbf{y}_{k-j}, \mathbf{f}_{k-j}$	$O(h^{q+1})$	$O(h^q)$	multistep, extends to semi-implicit
AB q	integration over $[t_k, t_{k+1}]$	$O(h^{q+1})$	$O(h^q)$	multistep, explicit, captures Im. axis for $q=3$
RK q	integration over $[t_k, t_{k+1}]$	$O(h^{q+1})$	$O(h^q)$	multistage explicit, easy to start
Extrapolation	extends methods above			implicit or explicit

Example: Stiff ODE

- Consider scalar ODE

$$y' = -100y + 100t + 101$$

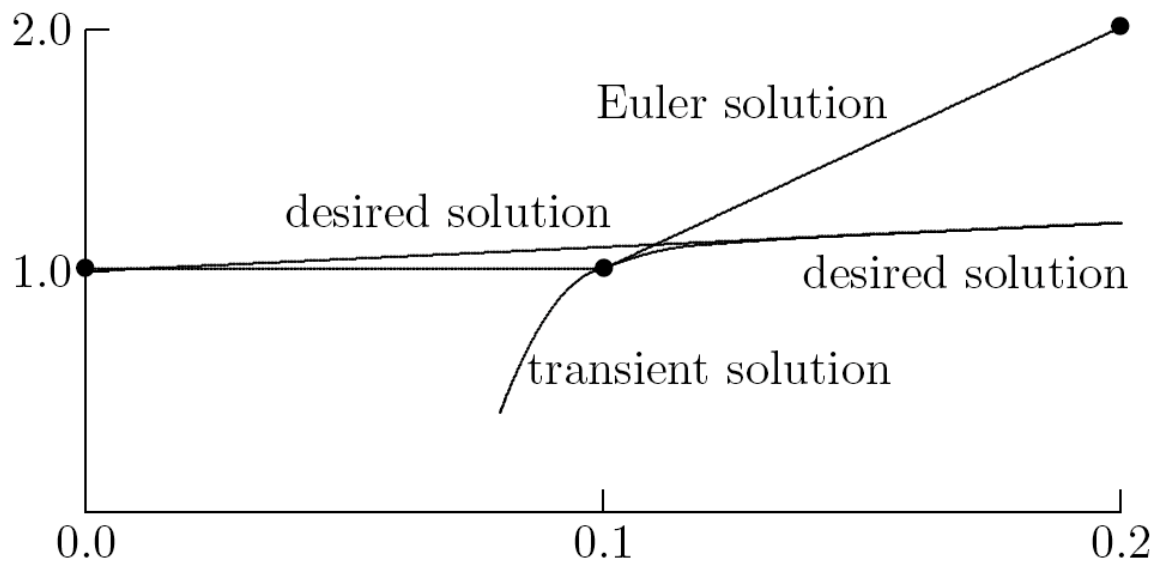
with initial condition $y(0) = 1$

- General solution is $y(t) = 1 + t + ce^{-100t}$, and particular solution satisfying initial condition is $y(t) = 1 + t$ (i.e., $c = 0$)
- Since solution is linear, Euler's method is theoretically exact for this problem
- However, to illustrate effect of using finite precision arithmetic, let us perturb initial value slightly



Example, continued

- Euler's method bases its projection on derivative at current point, and resulting large value causes numerical solution to diverge radically from desired solution
- Jacobian for this ODE is $J_f = -100$, so stability condition for Euler's method requires step size $h < 0.02$,



Multistep Methods

- *Multistep methods* use information at more than one previous point to estimate solution at next point
- *Linear multistep* methods have form

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

- Parameters α_i and β_i are determined by polynomial interpolation
- If $\beta_0 = 0$, method is explicit, but if $\beta_0 \neq 0$, method is implicit
- Implicit methods are usually more accurate and stable than explicit methods, but require starting guess for \mathbf{y}_{k+1}



Examples: Multistep Methods

- Simplest second-order accurate explicit two-step method is

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{2}(3\mathbf{y}'_k - \mathbf{y}'_{k-1})$$

- Simplest second-order accurate implicit method is trapezoid method

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{2}(\mathbf{y}'_{k+1} + \mathbf{y}'_k)$$

- One of most popular pairs of multistep methods is explicit fourth-order Adams-Bashforth predictor

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{24}(55\mathbf{y}'_k - 59\mathbf{y}'_{k-1} + 37\mathbf{y}'_{k-2} - 9\mathbf{y}'_{k-3})$$

and implicit fourth-order Adams-Moulton corrector

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{24}(9\mathbf{y}'_{k+1} + 19\mathbf{y}'_k - 5\mathbf{y}'_{k-1} + \mathbf{y}'_{k-2})$$



Examples: Multistep Methods

- *Backward differentiation formulas* form another important family of implicit multistep methods
- BDF methods, typified by popular formula

$$\mathbf{y}_{k+1} = \frac{1}{11}(18\mathbf{y}_k - 9\mathbf{y}_{k-1} + 2\mathbf{y}_{k-2}) + \frac{6h}{11}\mathbf{y}'_{k+1}$$

are effective for solving stiff ODEs

< interactive example >



Multistep Adams Methods

- Stability and accuracy of some Adams methods are summarized below
 - *Stability threshold* indicates left endpoint of stability interval for scalar ODE
 - *Error constant* indicates coefficient of h^{p+1} term in local truncation error, where p is order of method

Explicit Methods			Implicit Methods		
Order	Stability threshold	Error constant	Order	Stability threshold	Error constant
1	-2	1/2	1	$-\infty$	-1/2
2	-1	5/12	2	$-\infty$	-1/12
3	-6/11	3/8	3	-6	-1/24
4	-3/10	251/720	4	-3	-19/720

- Implicit methods are both more stable and more accurate than corresponding explicit methods of same order



Properties of Multistep Methods

- They are not self-starting, since several previous values of y_k are needed initially
- Changing step size is complicated, since interpolation formulas are most conveniently based on equally spaced intervals for several consecutive points
- Good local error estimate can be determined from difference between predictor and corrector
- They are relatively complicated to program
- Being based on interpolation, they can efficiently provide solution values at output points other than integration points



Properties of Multistep, continued

- Implicit methods have much greater region of stability than explicit methods, but must be iterated to convergence to enjoy this benefit fully
 - PECE scheme is actually explicit, though in a somewhat complicated way
- Although implicit methods are more stable than explicit methods, they are still not necessarily unconditionally stable
 - No multistep method of greater than second order is unconditionally stable, even if it is implicit
- Properly designed implicit multistep method can be very effective for solving stiff ODEs



Summary of Methods / Properties

Method	Implementation	LTE	GTE	Comments
EF	$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}_k$	$O(h^2)$	$O(h)$	explicit
EB	$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}_{k+1}$	$O(h^2)$	$O(h)$	implicit/stable
Trap	$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{2}(\mathbf{f}_k + \mathbf{f}_{k+1})$	$O(h^3)$	$O(h^2)$	implicit/stable
BDF q	interpolation of \mathbf{y}_{k-j}	$O(h^{q+1})$	$O(h^q)$	multistep, implicit, stable for $q < 3$
BDF q /EXT q	interpolation of $\mathbf{y}_{k-j}, \mathbf{f}_{k-j}$	$O(h^{q+1})$	$O(h^q)$	multistep, extends to semi-implicit
AB q	integration over $[t_k, t_{k+1}]$	$O(h^{q+1})$	$O(h^q)$	multistep, explicit, captures Im. axis for $q=3$
RK q	integration over $[t_k, t_{k+1}]$	$O(h^{q+1})$	$O(h^q)$	multistage explicit, easy to start
Extrapolation	extends methods above			implicit or explicit

- Multistep methods of order > 2 require special starting procedures
- Multistage (e.g., RK q) methods are self-starting and easy to change stepsize h .
- Multistage methods are attractive for automated stepsize selection.
- Be sure to understand the stability diagrams of these methods.
 - Left, or right side of complex λh plane?
 - Does it include Im. axis? If so, where does it cut?
 - Where does it cut the real axis?