

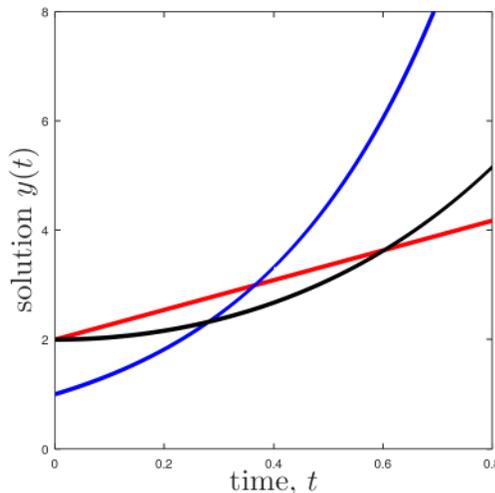
Chapter 9: Ordinary Differential Equations

- Numerical Solution of ODEs: Initial Value Problems

Differential Equations

- Differential equations involve derivatives of unknown solution function
- *Ordinary differential equation* (ODE): all derivatives are with respect to a *single* independent variable
- If it is an *initial value problem* (IVP), then the independent variable is typically thought of as time, t
- If it is an *boundary value problem* (BVP), the independent variable is typically thought of as space (e.g., x)

- IVP Examples:



ODE	I.C.	Solution
$\dot{y} = 3$	$y(0) = 2$	$y(t) = 2 + 3t$
$\dot{y} = 3y$	$y(0) = 1$	$y(t) = e^{3t}$
$\ddot{y} = 4y$	$y(0) = 2$ $\dot{y}(0) = 0$	$y(t) = e^{2t} + e^{-2t}$

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.

Scalar ODE Examples

- All of the examples of the preceding slide can be solved *numerically*
- Meaning that, starting with an initial condition (say, $y(t) = y_0$ at $t = 0$), we can evaluate $y(t)$ for $t > 0$ to almost arbitrary accuracy
- Common questions are,
 - *Will the numerical solution blow up (i.e., be unstable)?*
 - Will the *true solution* blow up?
 - *What is the accuracy of the numerical solution?*
 - *How can one assess the accuracy?*
 - *What is the cost?*
 - *How can we mitigate the cost?*

ODE Solvers as Integrators

- Two common approaches to developing numerical methods for IVPs are based on evaluating the following at specific time points, t_k ,

i. Approximating the *derivative* at t_k

$$\begin{aligned} f(t_k, y_k) &= \left. \frac{dy}{dt} \right|_{t_k} \approx \text{finite-difference approximation} \\ &= \frac{y_{k+1} - y_k}{\Delta t} + O(\Delta t) \\ &\approx \frac{y_{k+1} - y_k}{\Delta t} \text{ (for example)} \end{aligned}$$

where $h = \Delta t = t_{k+1} - t_k$ is the step size, which may or may not be uniform for all t_k

ODE Solvers as Integrators

ii. Approximating the *integral*

$$\begin{aligned}y_{k+1} &= y_k + \int_{t_k}^{t_{k+1}} f(t, y) dt \\ &= y_k + \Delta t f(t_k, y_k) + O(\Delta t^2) \\ &\approx y_k + \Delta t f(t_k, y_k)\end{aligned}$$

where we have used the (locally) $O(h^2)$ -accurate *rectangle rule* to approximate the integral on $[t_k, t_{k+1}]$

- Both approaches are equally viable and equally used
- Because of the second approach, however, timesteppers for solving IVPs are frequently referred to as ODE *integrators*

Example

- In the preceding example, each approach provided a *formula* for updating y_{k+1} , an approximation to $y(t_{k+1})$, based on y_k and $f_k = f(t_k, y_k)$
- Because they do not involve solving for the *argument* of $f(t, y)$, these methods are termed *explicit* in time, as opposed to *implicit*.
- Also, because they involve only information at t_k and (nominally) t_{k+1} , these are *single-step* methods
- The *particular* approximations made on the preceding slides lead to the *same* update formula:

$$y_{k+1} = y_k + \Delta t f(t_k, y_k)$$

which is known as *Euler Forward* timestepping

- It is *first-order accurate* in time, meaning that for a *fixed* final time T , the accuracy is $O(\Delta t)$ when taking n steps of size Δt such that $n\Delta t = T$.

Assessing Accuracy

- The most common approach to assessing accuracy, in practice, is to run numerical experiments
- We will also of course analyze the methods to understand how to control the accuracy but, because code is involved, testing is mandatory to assess whether the software matches the expected theoretical behavior
- Consider the nonlinear *Ricatti equation* example,

$$\frac{dy}{dt} = y^2 - y - 2, \quad y(0) = 0$$

- Solutions for this problem are well known
- According to chatgpt, the analytical solution for this particular case is

$$y(t) = 2 \frac{1 - e^{3t}}{1 + 2e^{3t}}$$

which (as is obligatory) we can check numerically

Ricatti Example, continued

- In the following matlab code, we run a sequence of tests out to time $t = t_{\text{final}}$ using n steps of size $\Delta t = t_{\text{final}}/n$
- Using Euler forward, we would expect that the final error is $O(\Delta t)$

```
%% RICATTI EQN Example:  $y' = y^2 - y - 2$ ,  $y(0)=0$ 
```

```
n = 1000;          # Number of timesteps
tfinal = 4.0;
dt = tfinal / n;

t = 0;
y = 0; %% Initial condition

for k=1:n;

    tk(k) = t;
    yk(k) = y;
    uk(k) = 2*(1-exp(3*t)) / (1 + 2*exp(3*t) );

    t = k*dt;

    f = y*y - y - 2;
    y = y + dt*f;          %% Euler Forward

end;
```

```
% Evaluate at final time:
```

```
k=n+1;
tk(k) = t;
yk(k) = y;
uk(k) = 2*(1-exp(3*t)) / (1 + 2*exp(3*t) );
ek = uk-yk;

plot(tk,uk,'k-',lw,2,tk,yk,'r--',lw,2,tk,ek,'b-',lw,2);
title(['Ricatti Equation Test, $n = '$ int2str(n)],intp,ltx,fs,24);
xlabel('Time, $t$',intp,ltx,fs,24);
ylabel('Solution and Error',intp,ltx,fs,24);
text(0.1,0.1,'$f=y^2-y-2$, $y(0)=0$',intp,ltx,fs,24);

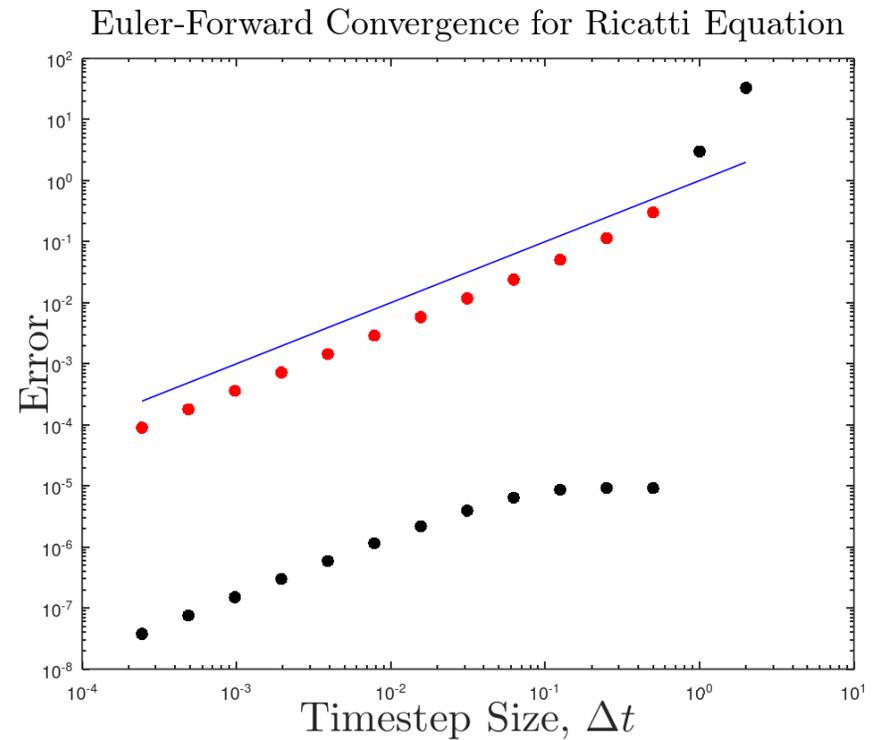
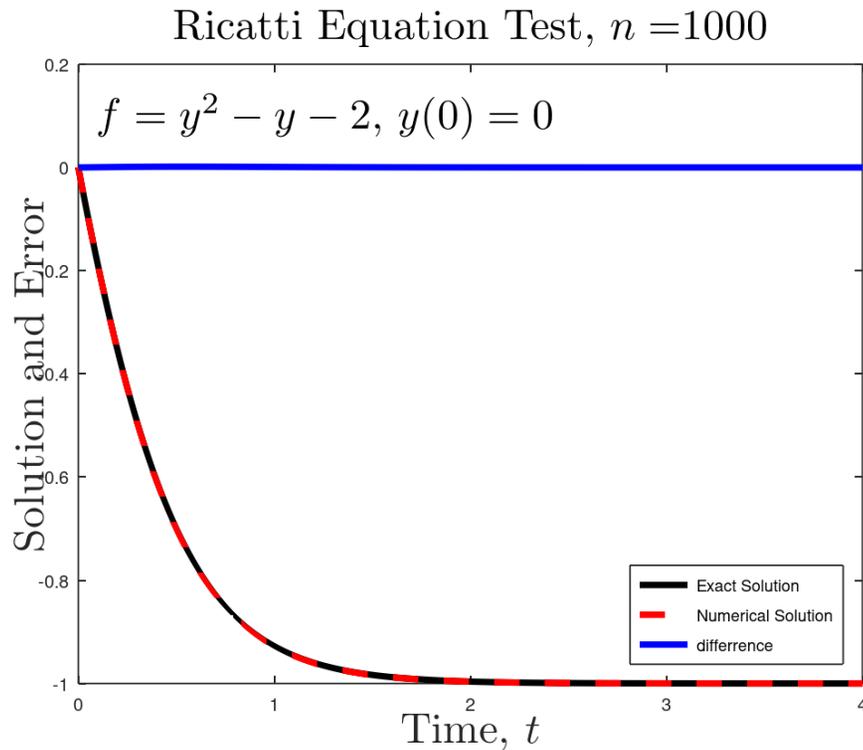
legend('Exact Solution','Numerical Solution','difference',...
       'location','southwest')
```

ricatti.m

ricatti_conv.m

Ricatti Example, continued

- The first plot shows that the exact solution and EF agree
- The second shows that the error is indeed $O(\Delta t)$



ricatti.m

ricatti_conv.m

Local Truncation Error

- Whether derived from the integral or differential approach, the *update step* for Euler forward has the following form

$$y_{k+1} = y_k + \Delta t f(t_k, y_k) + \underbrace{O(\Delta t^2)}_{LTE},$$

where *LTE* stands for *local truncation error*

- The LTE is the error incurred in moving from one step to the next if y_k had been the *exact* solution at t_k , $y(t_k)$
- In the case of Euler forward, the LTE is $O(\Delta t^2)$

Global Truncation Error

- We also have the *global truncation error*, or GTE, which is the difference $y_k - y(t_{k+1})$ or, more specifically, $y_n - y(T)$, where $T = t_{\text{final}}$ is the target integration time.
- Note that T is *fixed*, independent of n .
- As we refine the stepsize, $h \rightarrow 0$, we must *increase* n , such that $T = n\Delta t$ is *fixed*
- It is in this context that we talk about the order of accuracy of the method (i.e., the GTE)
- In the case of Euler forward, the GTE is $O(h)$ (i.e., $O(\Delta t)$)

Order of Accuracy

- In general, if the LTE is $O(h_k^{p+1})$, the GTE will be of order $O(h^p)$, and the order of accuracy will be p

- We can understand this result as follows:

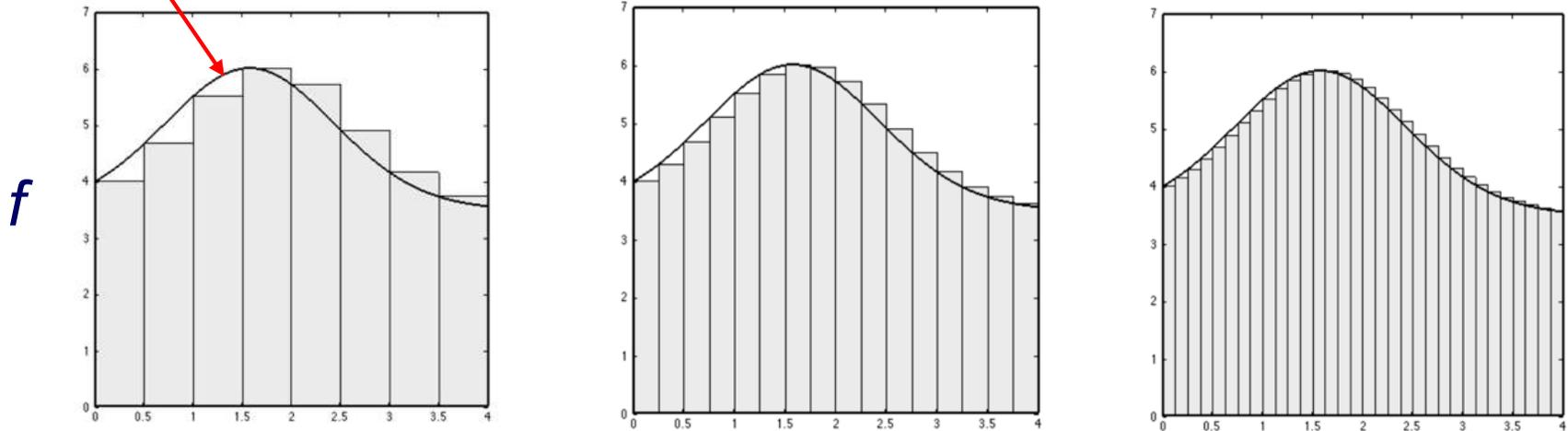
– If we make an LTE of $O(h^{p+1})$ on each step, we can expect, on average that the incurred error per step is $O(h^{p+1})$, which is committed n times, for a total error of

$$n \cdot O(h^{p+1}) = O(nh \cdot h^p) = O(T \cdot h^p) = T \cdot O(h^p)$$

- Assuming y is sufficiently smooth, then if the order of accuracy is $p = 2$, then the error (at time $t = T$!) will drop by a factor of 4 when we take twice as many steps with stepsize $h = T/n$
- If $p = 1$, we would expect the error to reduce by a factor of 2 under the same circumstances
- Notice that whether the error scales as $O(T)$ or not very much depends on the ODE itself

Relationship between LTE and GTE

$O(\Delta t^2)$



$$\frac{dy}{dt} = f \iff y_n = y_0 + \int_0^T f(t, y) dt$$

- 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,

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

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

Differential Equations

- Solution of differential equation is a *function* in an infinite-dimensional space of functions
- Numerical solution of differential equations is based on *finite-dimensional* approximation
- Differential equation is replaced by an algebraic equation whose solution approximates that of the original equation
- The process of mapping from the infinite-dimensional solution space to the finite-dimensional one is termed **discretization**

System of ODEs

- We can also have more than one unknown and, thus, more than one differential equation, with a corresponding number of initial conditions
- For two equations, we have

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

- We refer to this as a *system of ODEs* and express it in vector form as

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

- It of course holds true at a finite number of time points, which is what we typically try to approximate
- That is, given initial value $\mathbf{y}(t_0)$, we seek $\mathbf{y}(t_k)$, $k = 1, 2, \dots$, satisfying

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

Orbiting Particle Example

- Consider a particle with position $\mathbf{x}(t)$,

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

- Particle motion prescribed as

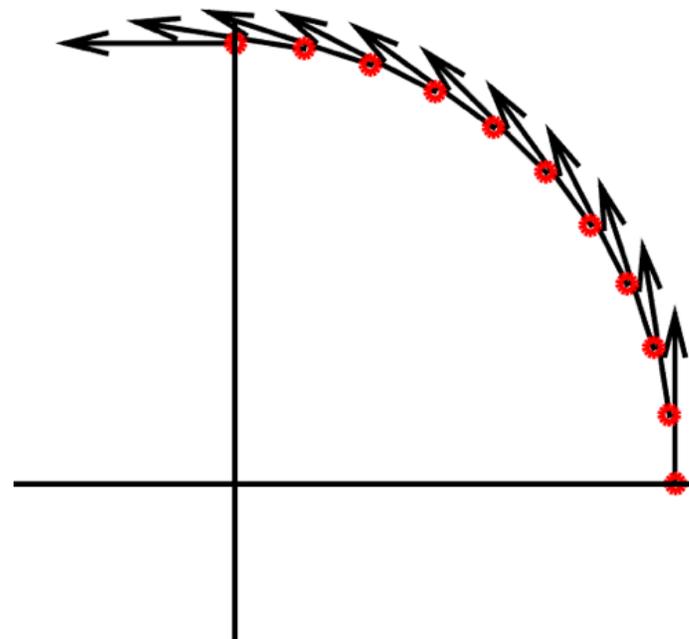
$$\dot{x} = -y$$

$$\dot{y} = x$$

- Vector form,

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

- We'll see shortly that the eigenvalues of this linear ODE system, $\dot{\mathbf{x}} = A\mathbf{x}$, govern the overall behavior of the solution.
- In this example, the eigenvalues are imaginary, which means that the norm of the solution neither decays nor grows.



Order of ODE

- **Order** of ODE is determined by highest order derivative of solution function appearing in ODE

- For example,

$$y'' = 3y' + y^2 + \cos t,$$

would be a *second-order* ODE

- ODE with higher-order derivatives can be transformed into equivalent first-order system (for IVPs)
- We will discuss numerical methods only for first-order ODE systems
- Most ODE software is designed to solve only first-order equations

Example: Second-Order ODE

- Consider the preceding example,

$$y'' = 3y' + y^2 + \cos t,$$

- Define $u_1(t) = y(t)$, $u_2(t) = y'(t)$

- By construction, $y'' = u_2'$

- Can thus write equivalent system of first-order ODEs,

$$\begin{bmatrix} \frac{du_1}{dt} \\ \frac{du_2}{dt} \end{bmatrix} = \begin{bmatrix} u_2(t) \\ 3u_2(t) + u_1(t)^2 + \cos t \end{bmatrix}$$

Higher-Order ODEs, continued

- For k th-order ODE,

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

define k new unknown functions,

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

- Then original ODE is equivalent to first-order system

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

4th-Order Example, Matrix Form

- Consider example:

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

Let $u_1 = y$, $u_2 = y'$, $u_3 = y''$, and $u_4 = y'''$.

$$\frac{d}{dt} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & & \\ & 0 & 1 & \\ & & 0 & 1 \\ & & & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ f \end{bmatrix}$$

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

Example: Newton's Second Law

- Newton's Second Law of Motion, $F = ma$, is a 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 is force and m is mass

- Defining $u_1 = y$ and $u_2 = y'$ yields the 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 position y of original second-order equation
- Second component u_2 is the velocity, y'

Ordinary Differential Equations

- General first-order system of ODEs has form

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

where $\mathbf{y} : \mathbb{R} \longrightarrow \mathbb{R}^n$, $\mathbf{f} : \mathbb{R}^{n+1} \longrightarrow \mathbb{R}^n$, 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, $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$ does not determine unique solution function
- This is because ODE merely specifies *slope* $\mathbf{y}'(t)$ of solution function at each point, but not actual value of $\mathbf{y}(t)$ at any point
- Infinite family of functions satisfies ODE, in general, provided \mathbf{f} is smooth
- To single out particular solution, \mathbf{y}_0 of solution function must be specified at some point t_0
- \mathbf{y}_0 is the **initial condition**

IVPs, 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 the term *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 a function of time

Example: IVP

- 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

$$y(t) = y_0 e^t$$

Example: IVP

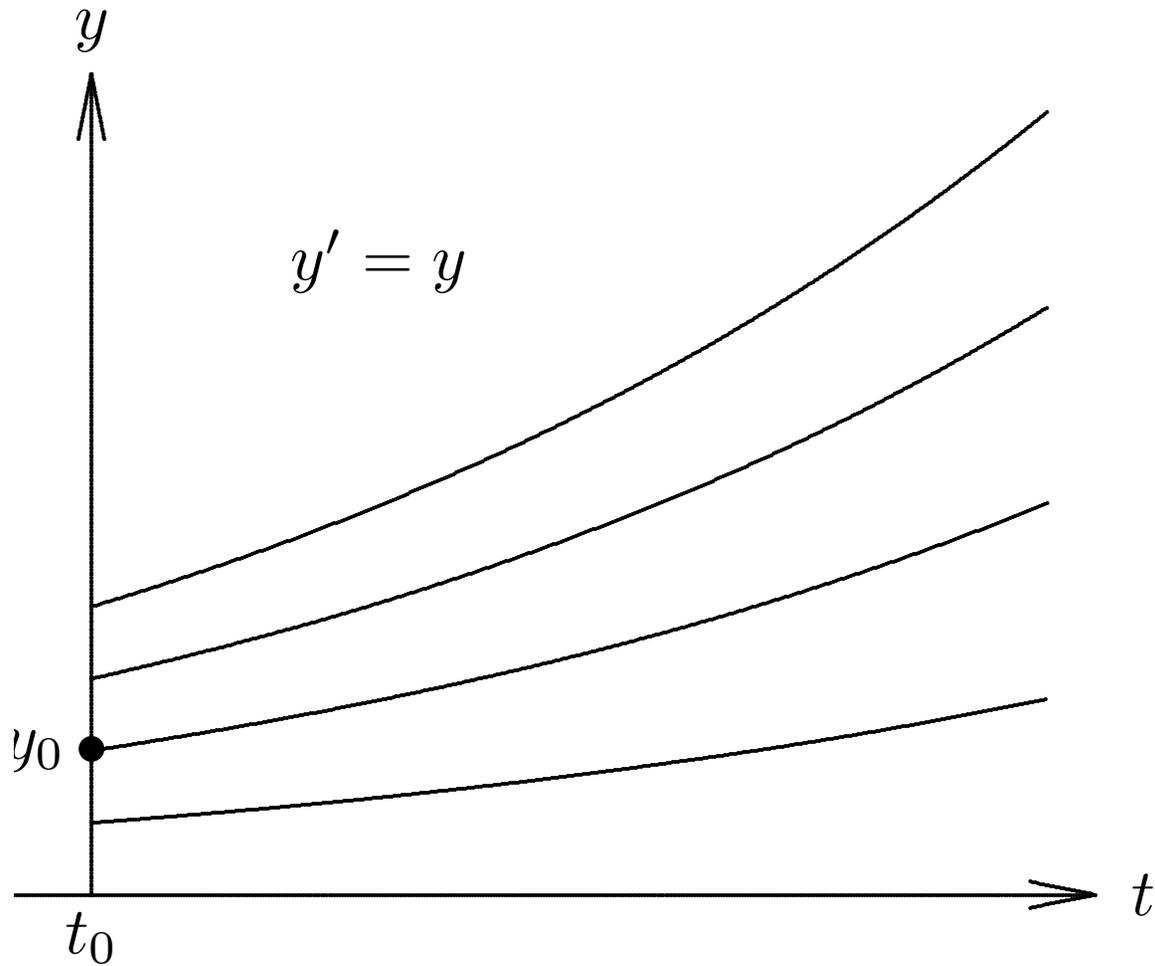


Figure 9.2: Some solutions for $y' = y$.

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 $y' = \frac{1}{2}$

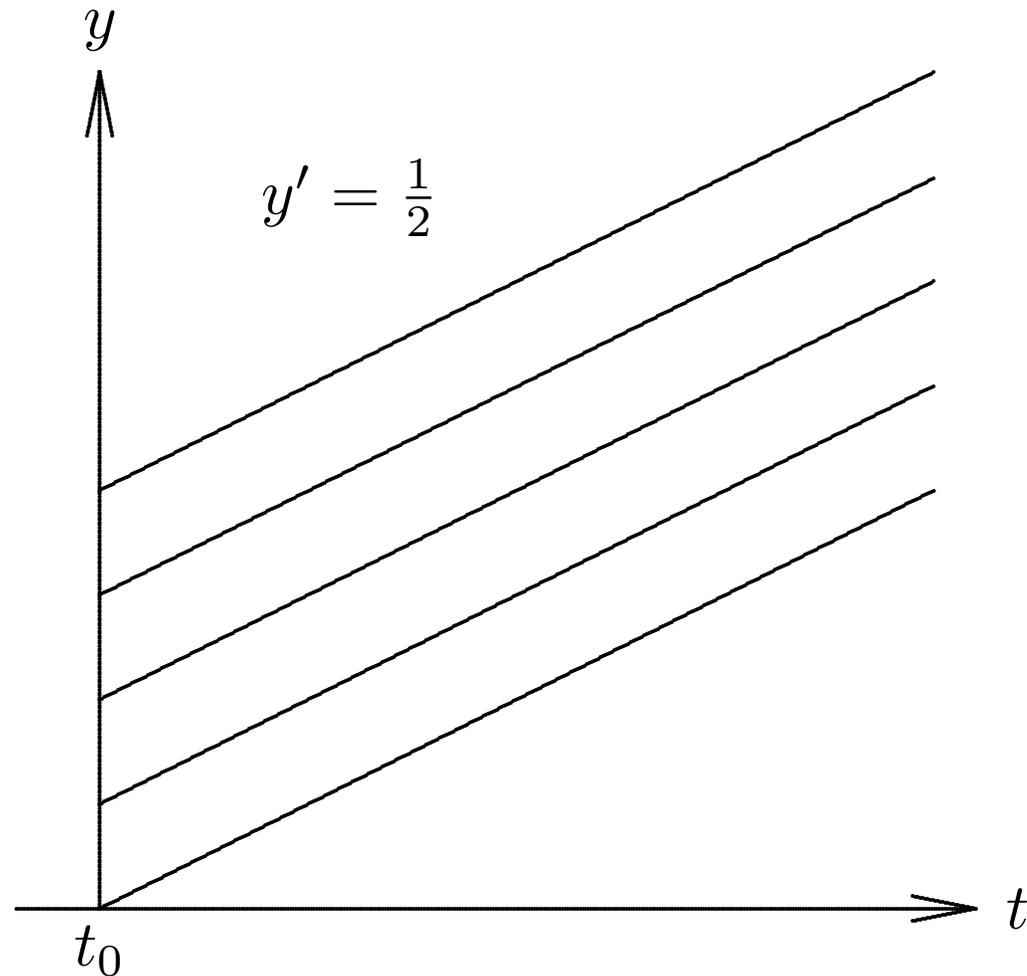
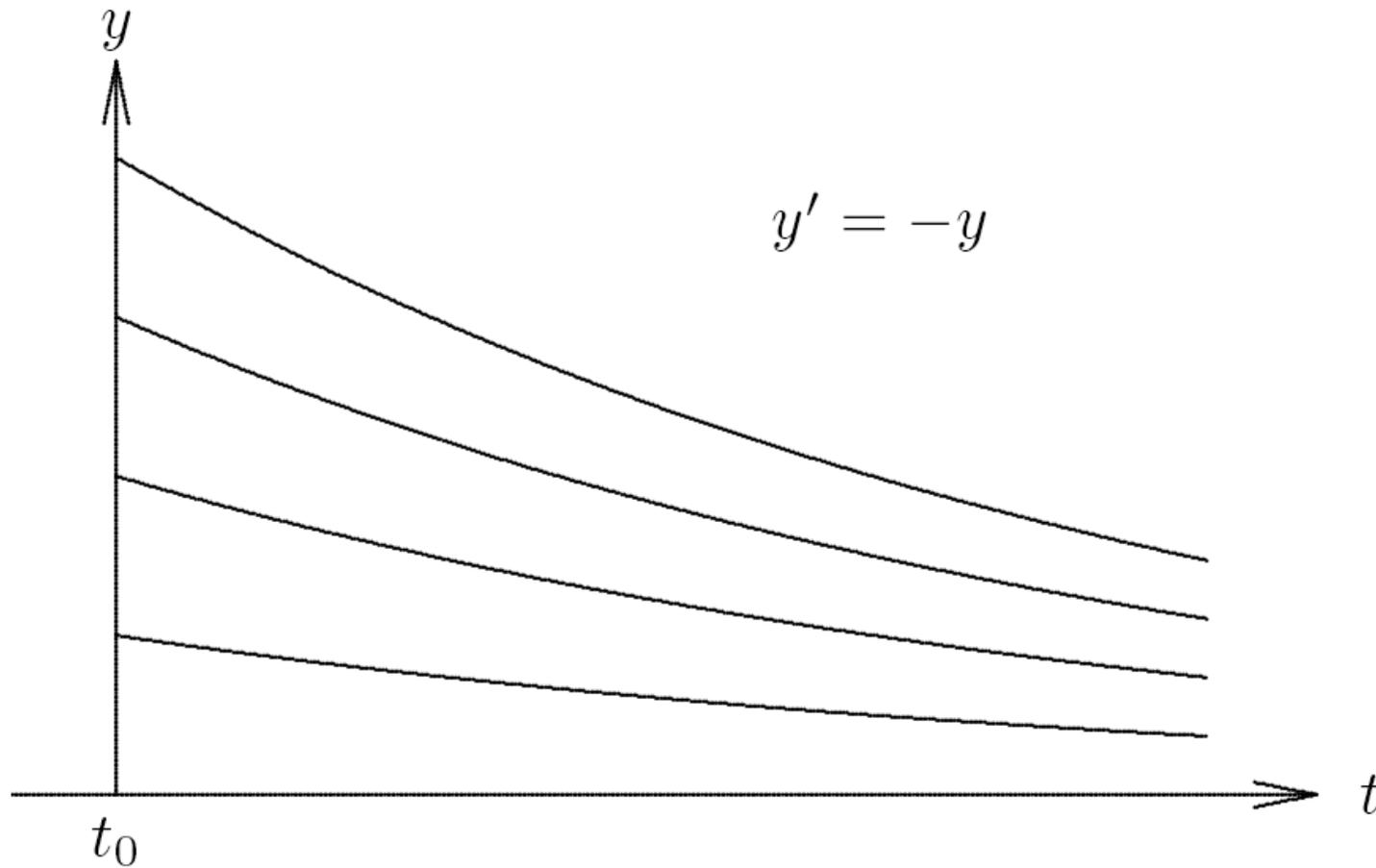


Figure 9.1: Some solutions for $y' = 1/2$.

Example: Asymptotically Stable Solutions

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



Example: Unstable Solutions

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

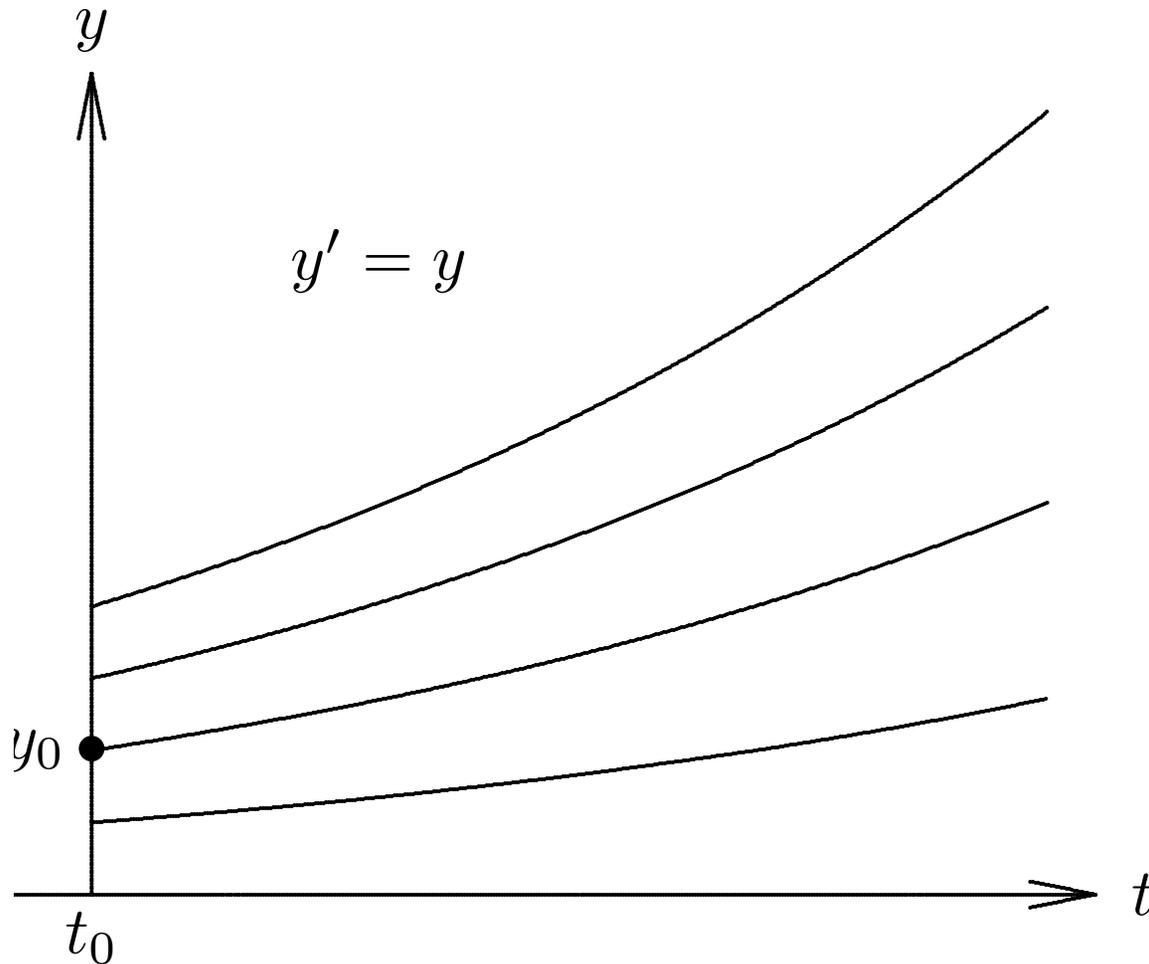


Figure 9.2: Some solutions for $y' = y$.

Example: Scalar Test ODE

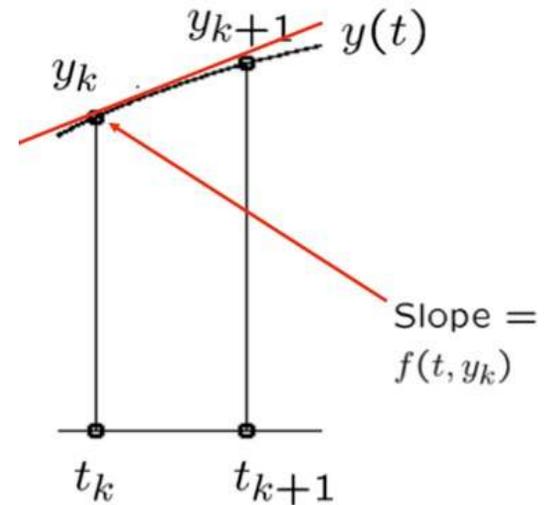
- Consider scalar ODE $y' = \lambda y$, where λ is constant
- Solution is given by $y(t) = y_0 e^{\lambda t}$, where t_0 is initial time and $y(0) = y_0$ is initial value
- For real λ ,
 - $\lambda > 0$: all nonzero solutions grow exponentially, so every solution is unstable
 - $\lambda < 0$: all nonzero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

System of ODEs: Solved Numerically

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

One example,

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



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

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

Orbiting Particle Example

- Consider a particle with position $\mathbf{x}(t)$,

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

- Particle motion prescribed as

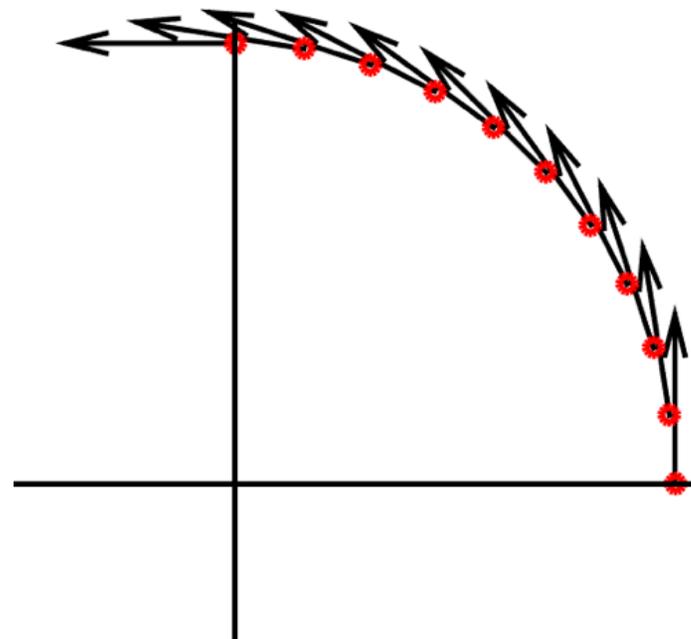
$$\dot{x} = -y$$

$$\dot{y} = x$$

- Vector form,

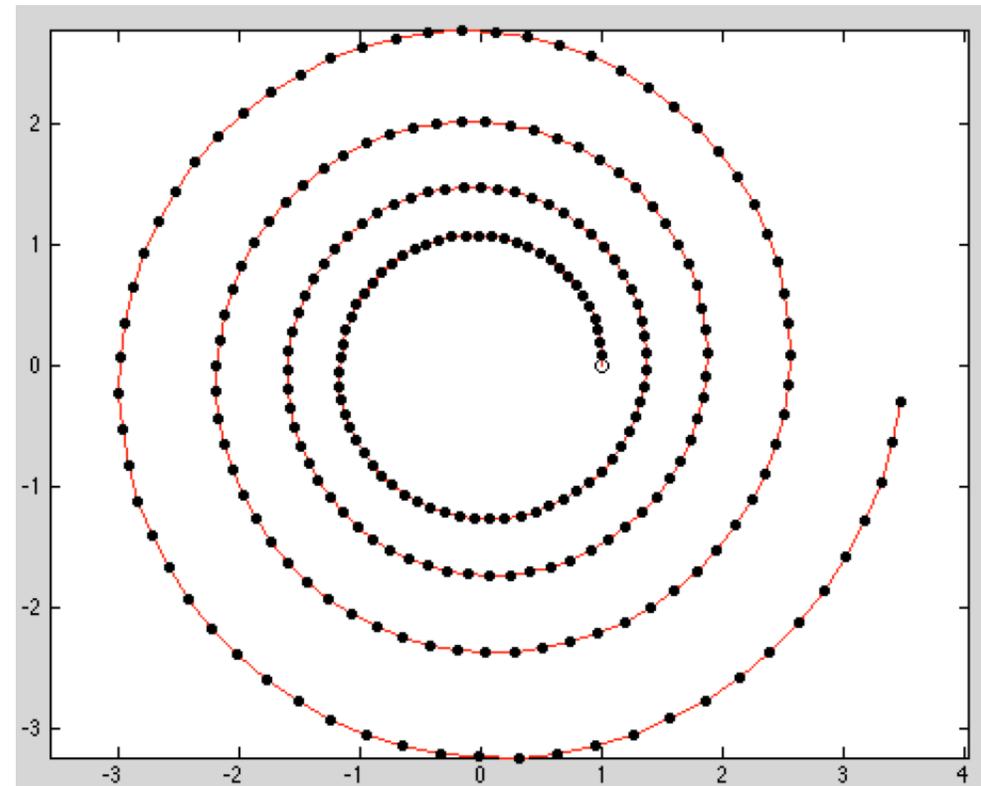
$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

- We'll see shortly that the eigenvalues of this linear ODE system, $\dot{\mathbf{x}} = A\mathbf{x}$, govern the overall behavior of the solution.
- In this example, the eigenvalues are imaginary, which means that the norm of the solution neither decays nor grows.



Euler Forward Example: orbit.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
```



Example: Linear System of ODEs

- Linear homogeneous system of ODEs with constant coefficients has the 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_j and corresponding eigenvectors \mathbf{v}_j , $j = 1, \dots, n$

- Then V^{-1} exists, so can find $\hat{\mathbf{y}} = V^{-1}\mathbf{y}_0$ such that $\mathbf{y}_0 = \sum_{j=1}^n \mathbf{v}_j \hat{y}_j$

- Solution is

$$\mathbf{y}(t) = \sum_{j=1}^n \hat{y}_j \mathbf{v}_j e^{\lambda_j t}$$

- If $Re(\lambda_j) > 0$ for *any* j then solution is *unstable*

Example, continued

- Eigenvalues of \mathbf{A} with *positive* real parts yield exponentially *growing* solution components
- Eigenvalues of \mathbf{A} with *negative* real parts yield exponentially *decaying* solution components
- Eigenvalues of \mathbf{A} with *zero* real parts (i.e., pure imaginary) yield *oscillatory* solution components
- Solutions are stable if $Re(\lambda_j) \leq 0$ for *all* j , and asymptotically stable if $Re(\lambda_j) < 0$ for *all* j
- Solutions are unstable if $Re(\lambda_j) > 0$ for *any* j

Stability in the General Case

- For nonlinear system of ODEs, $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, can generally only determine *local stability*, by linearizing the Jacobian at time t to yield new system

$$\mathbf{z}' = \mathbf{J}_f(t, \mathbf{y}(t))\mathbf{z}$$

where \mathbf{J}_f is Jacobian of \mathbf{f} with respect to \mathbf{y}

- Eigenvalues of \mathbf{J}_f determine local stability, which may not be valid globally

Eigenvalues and ODEs

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

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

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

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

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

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

Eigenvalues and ODEs

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

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

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

$$= V\Lambda\hat{\mathbf{y}} + V\hat{\mathbf{f}}$$

Multiply through by V^{-1} :

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

Eigenvalues and ODEs

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

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

Computable

- We now have n *decoupled* systems.
- Numerically, we solve these as the coupled system $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} + \mathbf{f}$.
- 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.

Stability of Numerical Method

- Numerical method is *stable* if small perturbations do not cause resulting numerical solutions to diverge from each other without bound
- Such divergence could be caused by numerical stability of the ODE itself or by the numerical method (even if the ODE is stable)

Determining Stability and Accuracy

- Simple approach to determining stability and accuracy is to consider the **model problem**

$$y' = \lambda y$$

where λ is a constant (possibly complex)

- **Exact solution** is

$$y(t) = y_0 e^{\lambda t}$$

with IC $y_0 = y(0)$

- Determine stability of numerical method by characterizing growth of numerical solution
- Determine accuracy of numerical method by comparing numerical solution to *exact* solution

Example: Euler Forward

- Applying Euler forward to $y' = \lambda y$, we have

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

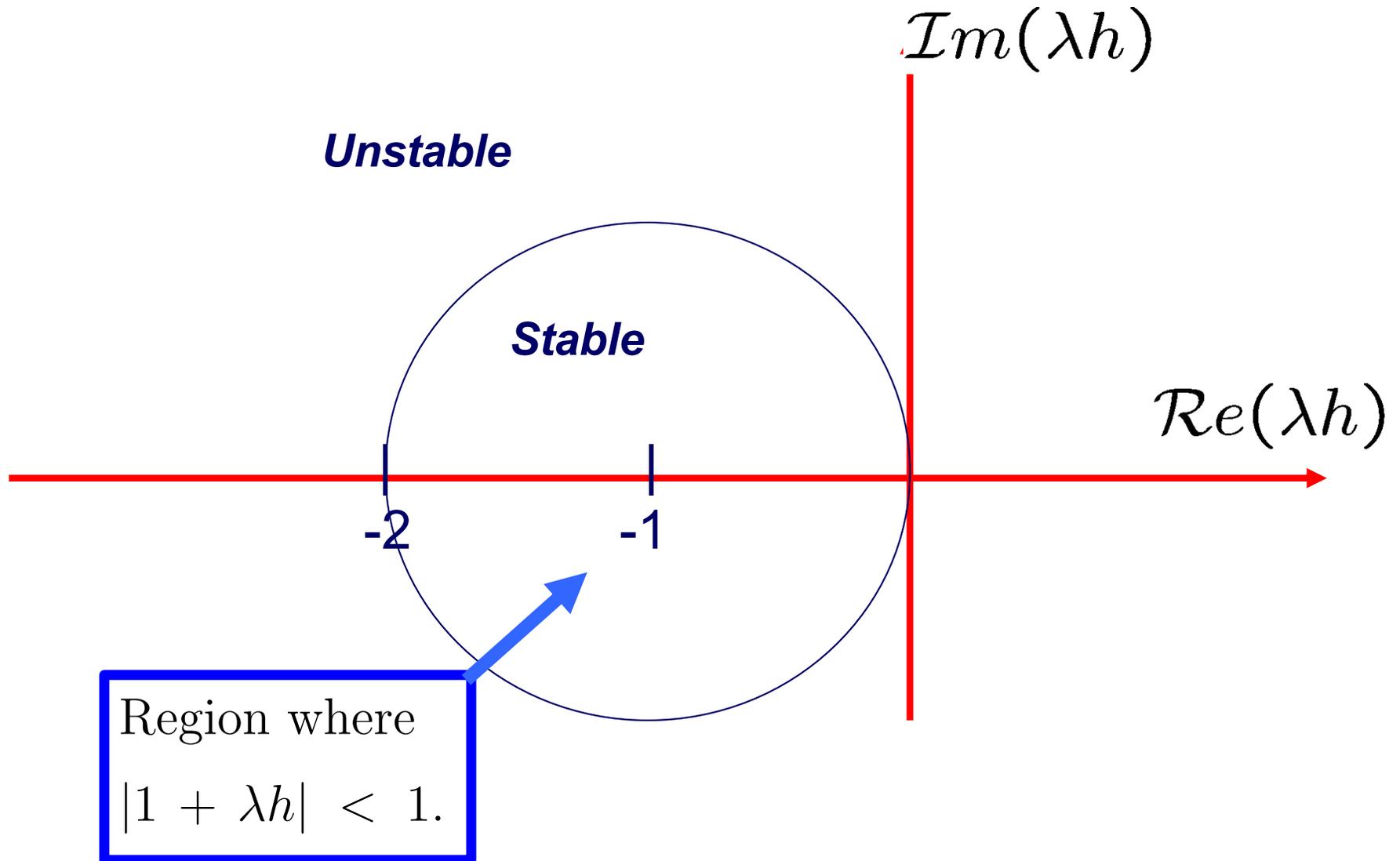
- If $\operatorname{Re}(\lambda) < 0$, then exact solution $y_0 e^{\lambda t}$ decays to zero as t increases, which is also the case if

$$|G_{EF}(h\lambda)| := |1 + h\lambda| < 1$$

where $G_{EF}(h\lambda) := 1 + h\lambda$ denotes the *growth factor* for Euler forward

- $|G_{EF}|$ is a measure of the growth (or decay) of the Euler forward method for the *model problem* on each step
- We also have the *analytical growth factor*, $\tilde{G} = e^{h\lambda}$, which reflects the growth/decay of the analytical solution after a single step of size h
- Note that $|G_{EF}(h\lambda)| < 1$ for any $h\lambda$ in the unit circle of the complex plane that is centered at $(-1, 0)$

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-')

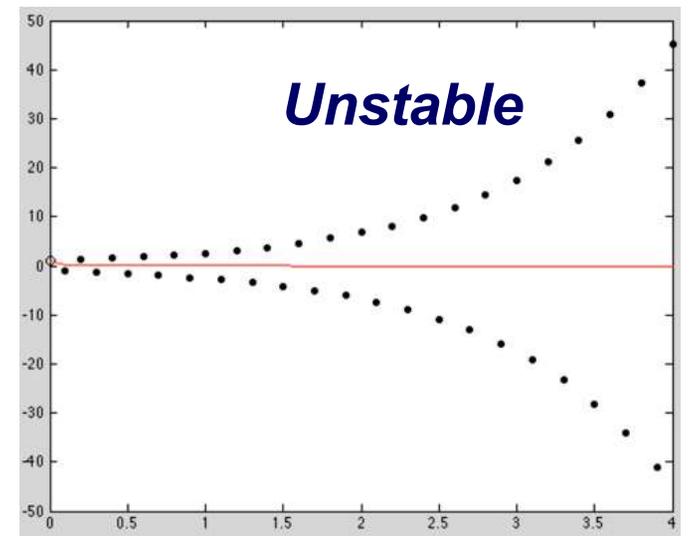
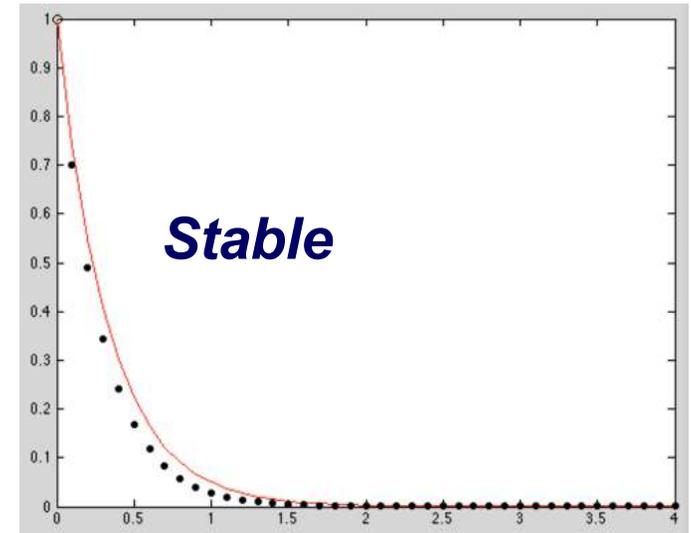
for k=1:nsteps;

    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;
```



Growth Factor, continued

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

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

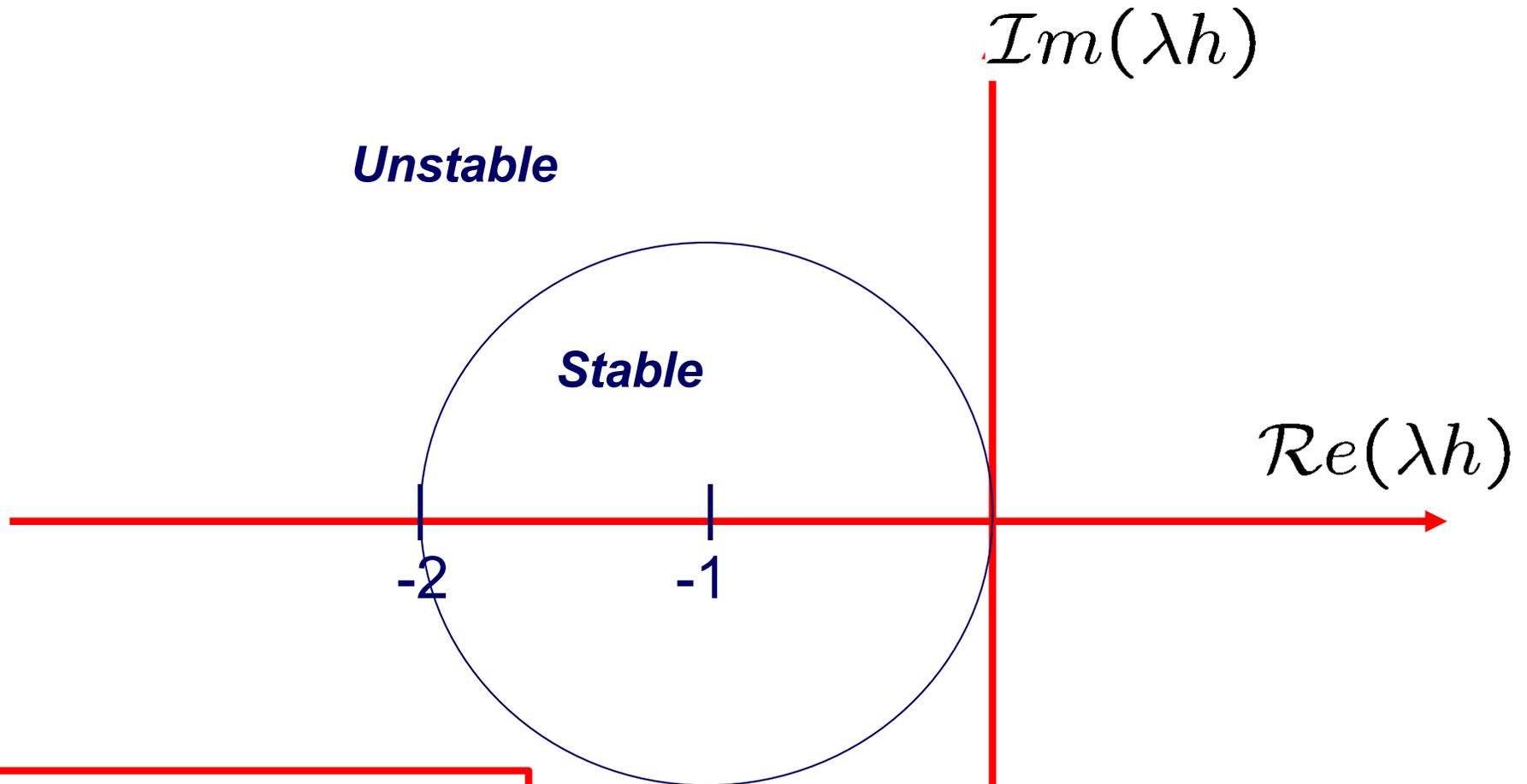
for Euler forward to be stable

- Growth factor $G_{EF} = 1 + h\lambda$ agrees with series expansion for \tilde{G}

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

through terms of first order in h , so Euler forward is first-order accurate

Stability Region for Euler's Method



Why complex plane?

Recall: Orbit Example

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

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

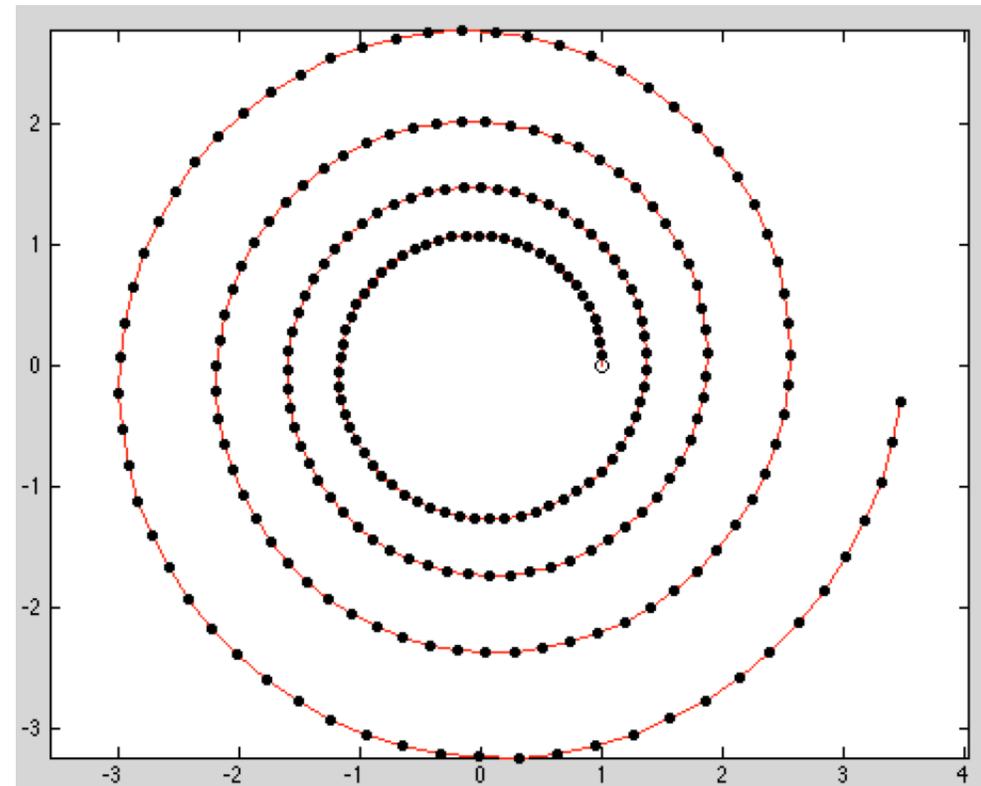
$$\begin{aligned} |A - \lambda I| &= \begin{vmatrix} -\lambda & -1 \\ 1 & -\lambda \end{vmatrix} \\ &= \lambda^2 + 1 = 0 \end{aligned}$$

$$\lambda = \pm i$$

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

Euler Forward Example: orbit.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
```



Stopped Here

Quick ODE Review

- Introduced Euler-forward
- Explored behavior for linear homogeneous system, $\mathbf{y}' = \mathbf{A}\mathbf{y} + \text{IC}$
- Behavior governed by eigenvalues of \mathbf{A} (or, \mathbf{J}_f for nonlinear systems)
 - If $\text{Re}(\lambda_j) \leq 0$ for $j = 1, \dots, n$ - stable
 - If $\text{Re}(\lambda_j) < 0$ for $j = 1, \dots, n$ - asymptotically stable
 - If $\text{Re}(\lambda_j) > 0$ for *any* j - unstable
- Therefore, we consider **model problem**, $y' = \lambda y + \text{IC}$
- Advancing a single step with Euler forward and the analytic solution yields

$$\text{EF: } y_{k+1} = y_k + h\lambda y_k = (1 + h\lambda) y_k = G_{EF} y_k$$

$$\text{Exact: } y_{k+1} = y_k e^{h\lambda} = \tilde{G} y_k$$

Review, continued

- Growth factors:

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

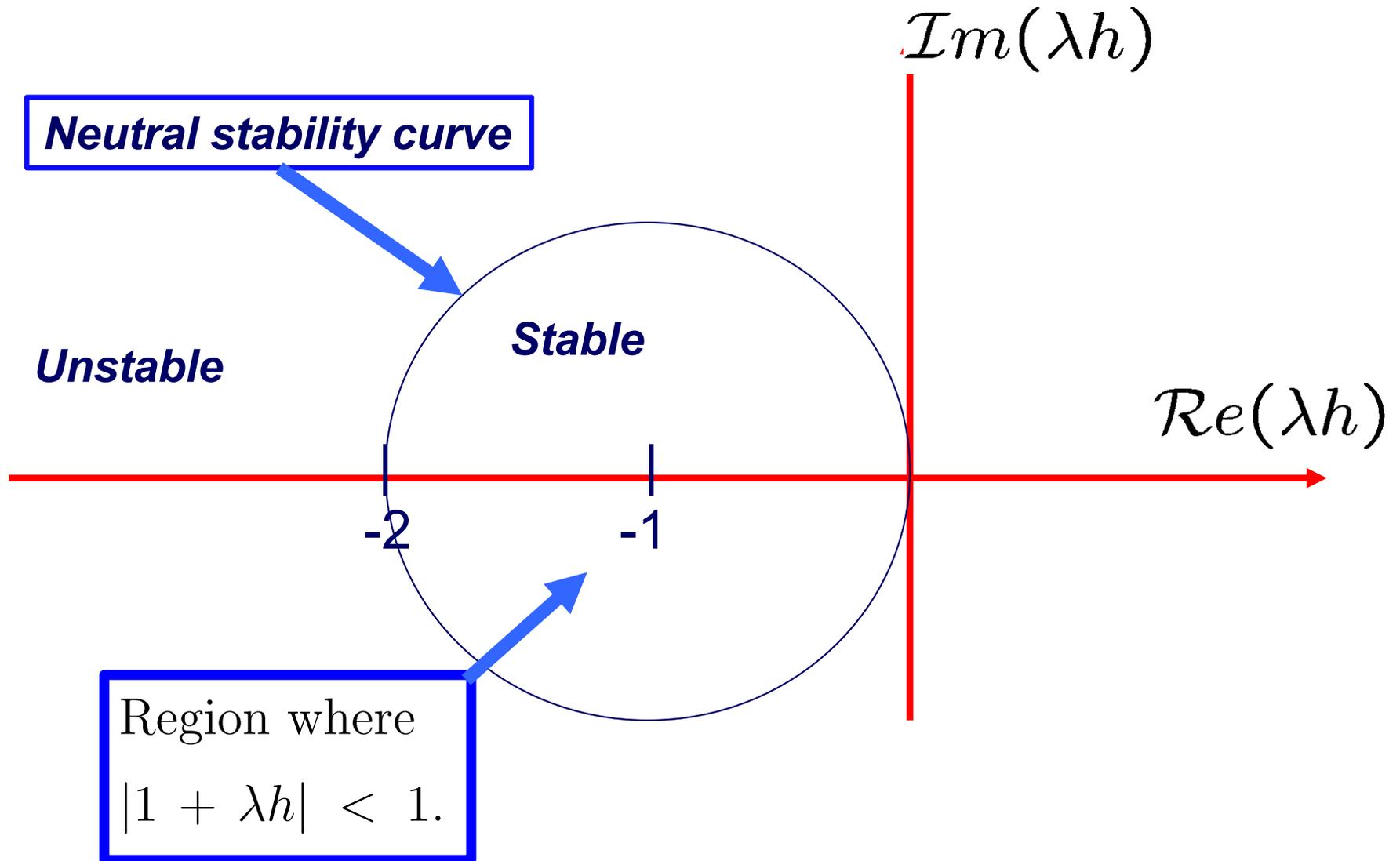
$$G_{EF} = 1 + h\lambda$$

- G_{EF} agrees with analytical \tilde{G} for first two terms, but on *each step*, incurs a *local truncation error* (LTE) of $O(h^2)$.
- So, $\text{LTE}=O(h^2) \longrightarrow$ *global truncation error* (GTE) is $O(h)$
- Euler forward is first-order accurate in time \longrightarrow error at final *fixed* time, $nh = T$, (not dependent on h) is $O(h)$
- Note that for any λ with $\text{Re}(\lambda) < 0$, $|\tilde{G}| < 1$, so ODE is *stable*
- EF is only *conditionally stable*, meaning that if $\text{Re}(\lambda) < 0$, then G_{EF} will be stable if

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

which corresponds to the requirement that $h\lambda$ be in the unit-radius circle centered at $(-1, 0)$ in the complex $h\lambda$ -plane

Stability Region for Euler's Method



Next Up:

- Shortcomings of EF
 - Accuracy
 - Stability
- Adaptive timestepping

Shortcomings of EF: Accuracy and Stability

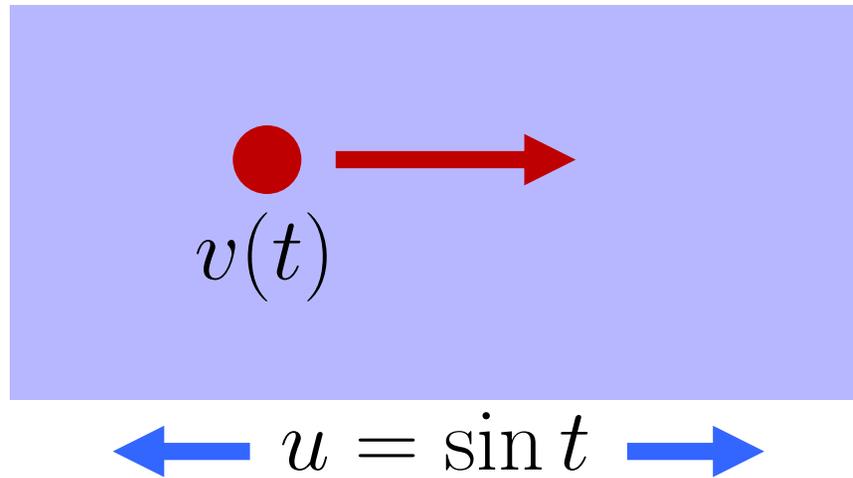
- Formally, the limited accuracy and stability of EF can be resolved by using smaller stepsize, h
- However, if final time T is relatively large or if the accuracy tolerance requirements are tight, using small steps may not be practical
- Also some systems can be arbitrarily *stiff*, which means that *stability* dictates the use of a small timestep is required because of one or more fast but unimportant timescales in the problem
 - Recall, *all* eigenvalues must be in the stability region to avoid blow-up
- Related to accuracy is the question of adaptive timestepping, which we'll touch upon at the end of this section

Stability of ODE Timesteppers

- When is stability important or, more to the point, when is lack of stability problematic?
- Very often you need relatively small timesteps for *accuracy*
- If that timestep size is close to the value required for *stability*, then there is no particular need to abandon an explicit (i.e., conditionally stable) timestepper in favor of an implicit one.
- However, if you have system with *disparate time scales*, where you are interested in the slow (e.g., long) time scale, but stability is constrained by a short time scale, then implicit methods can offer significant advantages.
- ODE systems with disparate timescales are referred to as *stiff*

Examples of Stiff Systems

- Consider the equation for a particle in a fluid-field box that is being moved back and forth.
- Let $u = \sin t$ be the velocity of the box+fluid.



Examples of Stiff Systems

- Consider the equation for a particle in a fluid-field box that is being moved back and forth.
- Let $u = \sin t$ be the velocity of the box+fluid.
- A simple model for the force on the particle is the difference between its velocity, $v(t)$, and the fluid velocity
- Newton's 2nd law says that the acceleration of the particle times its mass is proportional to the applied force

$$m \frac{dv}{dt} = c \cdot (u - v) = -cv + c \sin(t)$$

- Note that if $m = 0$ we must have $v = u$, that is the particle velocity equals that of the surrounding fluid.
- In this case, the relevant time scale is $\approx 2\pi$, which is the period of motion of the box
- In fact, as $m \rightarrow 0$, this is the behavior that is observed,

Stiff ODE Example, continued

- For $m \neq 0$, however, we have the solution (from chatgpt)

$$v(t) = \frac{s(s \sin(t) - \cos(t))}{s^2 + 1} + \frac{a(s^2 + 1) + s}{s^2 + 1} e^{-st}$$

where a is the initial condition and $s = c/m$ is termed the *Stokes number*

- For small mass (e.g., dust in the air), we can have $s = c/m \gg 1$, which means that the particle velocity matches that of the surrounding fluid in a very short time
- This is an example of a stiff system: we have a slow (long) time scale, which is the period of motion, and a fast (short) timescale, which is the required for the discrepancy between v and u to decay
- Stability of Euler forward dictates that $|hs| < 2$, which means $\Delta t \ll 1$ in the current context, even though the interesting time scale is $\approx 2\pi$
- Let's look at an example

stiff_sin.m

Sloshing Particle Example, continued

- First, notice the exponential dependency on the Stokes number, s ,

$$v(t) = \frac{s(s \sin(t) - \cos(t))}{s^2 + 1} + \frac{a(s^2 + 1) + s}{s^2 + 1} e^{-st}$$

- As s increases, the transient time diminishes, and the solution rapidly locks onto the sinusoidal driving velocity, $\sin(t)$
- Thus, for relatively long-time behavior (e.g., out to $T = 10$), we don't need to track the short transient near $t = 0$
- For **stability** reasons, however, EF forces us to take a small timestep, $h < 2/s$, when $s \gg 1$.
- Thus, the stability limitations are more severe than the long-time accuracy requirements—this is an examples of stiffness, i.e., disparate time scales
- If we are *interested* in the short-time behavior, we need a small timestep for *both* accuracy and stability – such a system (or problem) is *not* stiff

```

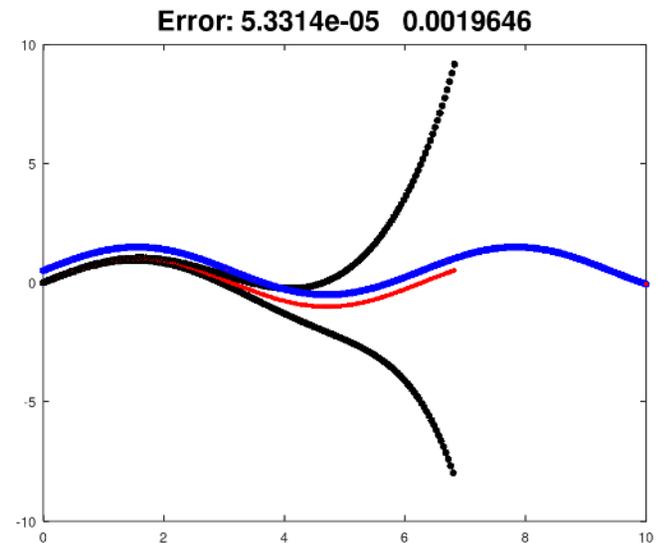
%% Stiff Example  m.v' = mu*(u-v), with u=sin(t) -->
%% This is a model of a particle of mass m in a
%% sloshing fluid with viscosity mu
%% The relevant (inverse) time scale is c = (mu/m)
clear all; hdr; hold off;
c=99; h=0.02; Tfinal = 10.; nsteps=ceil(Tfinal/h);
C = c/(c*c+1);
%% Euler Fwd
emx = 0; y = 1.00; %% IC
for k=0:nsteps; t = h*k; t1=t+h;
    u = sin(t); u1=sin(t1);
    v = C*(c*u1-cos(t1)) + (1+C)*exp(-c*t1); %% Exact
    f = -c*(y-u);
    y = y + h*f;
    err = abs(v-y); emx = max(emx,err);
    plot(t,y,'k.',ms,9,t,v,'r.',ms,6);
    title(['EF Error: ' num2str([err emx])],fs,20);
    hold on; if abs(y) > 9; break; end; drawnow;
    xlabel('time, t,fs,20); ylabel('Solution',fs,20);
end;

```

```

%% Euler Backward (1+cy) y = y + f_k+1
eb = 0;
if eb > 0;
    emx = 0; y = 1.00; %% IC
    for k=0:nsteps; t = h*k; t1=t+h;
        u = sin(t1);
        v = C*(c*u1-cos(t1)) + (1+C)*exp(-c*t1); %% Exact
        f = c*u;
        y = (y + h*f)/(1+c*h);
        err = abs(v-y); emx = max(emx,err);
        plot(t,y+.5,'b.',ms,9,t,v+.5,'r.',ms,6);
        title(['EB Error: ' num2str([err emx])],fs,20);
        hold on; if abs(y) > 9; break; end; drawnow;
        xlabel('time, t,fs,20); ylabel('Solution',fs,20);
    end;
end;

```



Rationale for Implicit Methods

- The preceding example was a linear, scalar, *inhomogeneous* ODE, there the inhomogeneity, $u(t) = \sin(2\pi t/\tau)$ set one time scale (the period, τ) and the eigenvalue, s , set the other, which was stepsize-limiting when $s \gg \tau^{-1}$
- In this example, s was the governing eigenvalue and we understand the limitations on h imposed by the requirement that $hs = h\lambda$ be within the stability region in the $h\lambda$ -plane
- Stability limitations can be bypassed by using **implicit methods**, in which we evaluate part or all of the *rhs* of the ODE at time t_{k+1} , rather than t_k , as was done with EF

Euler Backward (EB)

- The simplest implicit scheme, known as *Euler Backward* (EB) timestepping, is

$$\frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{\Delta t} = \mathbf{f}(t_{k+1}, \mathbf{y}_{k+1}) + O(\Delta t)$$

- Rearranging, setting $h = \Delta t$ and dropping the error term, we have the implicit system,

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

which will require a nonlinear system solve if \mathbf{f} is nonlinear in \mathbf{y}_{k+1}

- If \mathbf{f} is *linear* of the form $\mathbf{f}(t, \mathbf{y}) = \mathbf{A}(t)\mathbf{y} + \mathbf{g}(t)$, we still require a *system solve* involving the matrix $\mathbf{I} - h\mathbf{A}$, which is generally more expensive than the simple matrix-vector product, $\mathbf{A}\mathbf{y}_k$, required of an explicit method such as EF

Properties of Euler Backward

- To understand EB, we once again consider the model problem $y' = f$ with $f = \lambda y$
- For EB, we have

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

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

which will be *stable* if

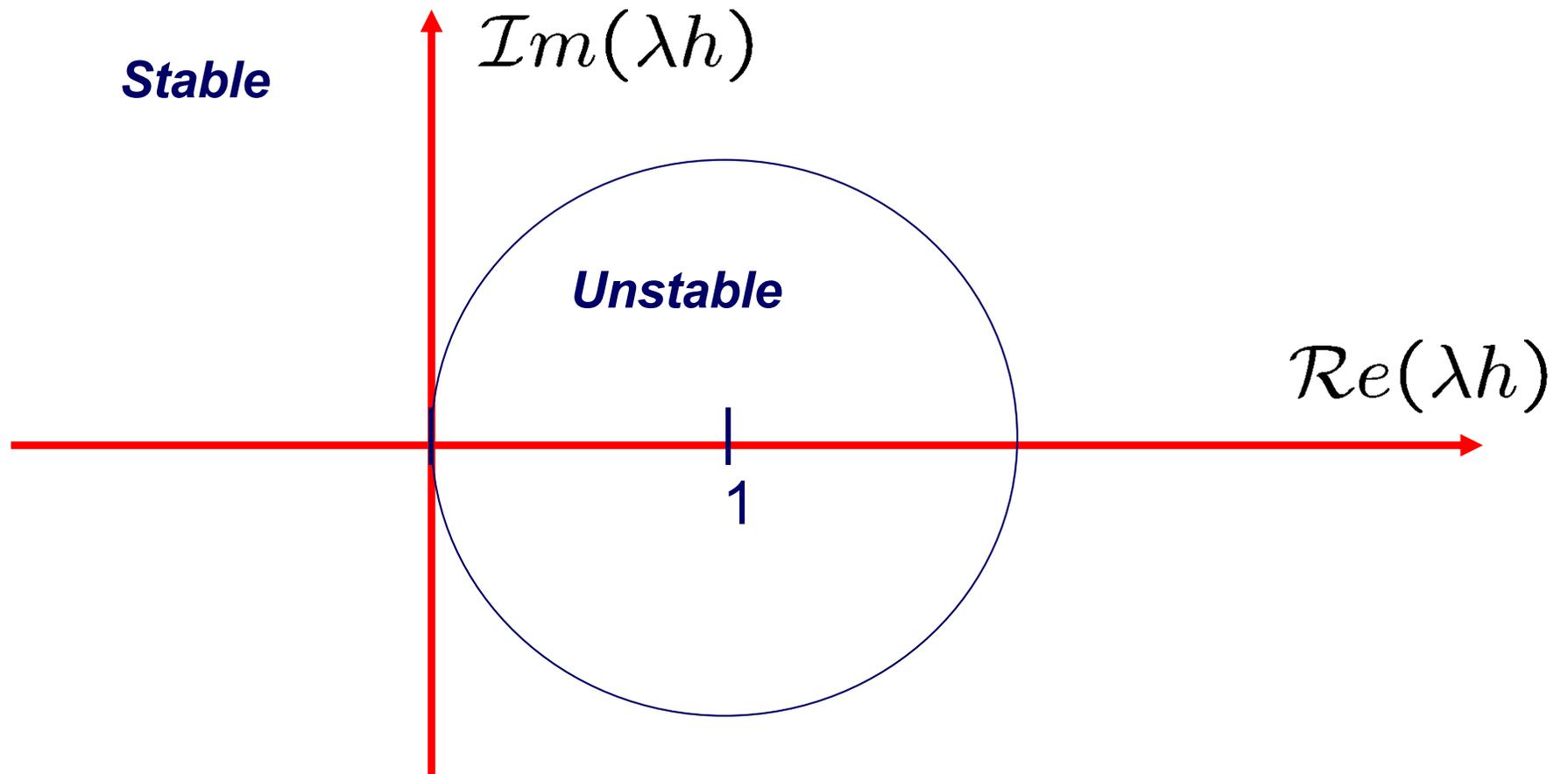
$$|G_{EB}| := \left| \frac{1}{1 - h\lambda} \right| \leq 1$$

where G_{EB} is the growth factor associated with Euler backward

- The neutral stability curve, $|G_{EB}| \equiv 1$, is the unit-radius circle centered at $(1, 0)$ in the complex $h\lambda$ plane
- Any point *outside* the circle will have $|G_{EB}| < 1$ and EB will be *stable* for these cases
- In particular, EB will be stable for *any* $Re(\lambda) \leq 0$, so we say that EB is **unconditionally stable**

stiff_sin.m

Stability Region for Backward Euler Method



Recall: Orbit Example

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

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

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

$$\lambda = \pm i$$

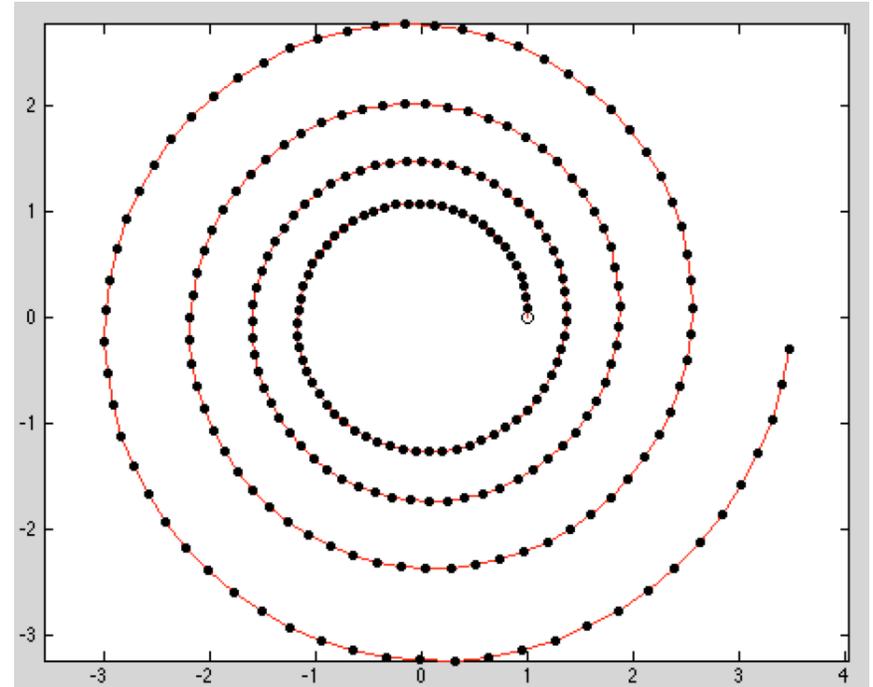
- ***Even though ODE involves only reals, the behavior can be governed by complex eigenvalues.***

Orbit Example: *orbit_ef_eb.m*

- **Recall our EF orbit example**

```
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
```

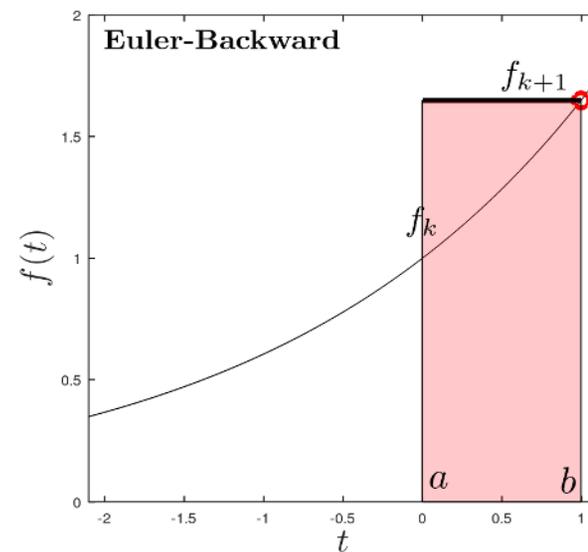
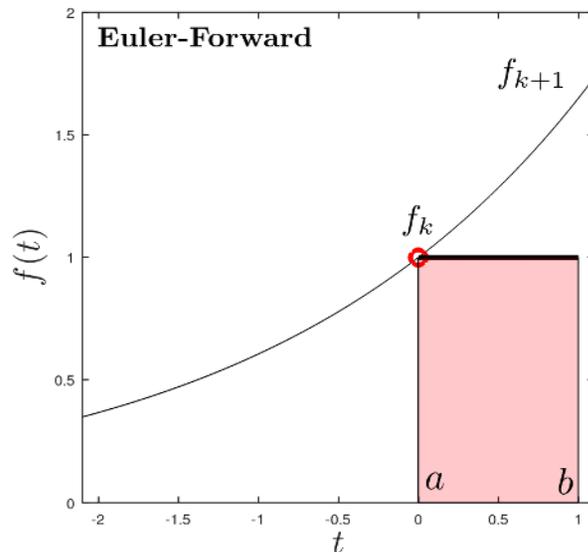
- **Imaginary eigenvalues outside the stability region**



- **What happened to radius vs time?**
- **What would we expect for Euler-Backward??**
 - **Where are the $h\lambda$ values with respect to the EB neutral stability curve?**

Higher-Order Timesteppers

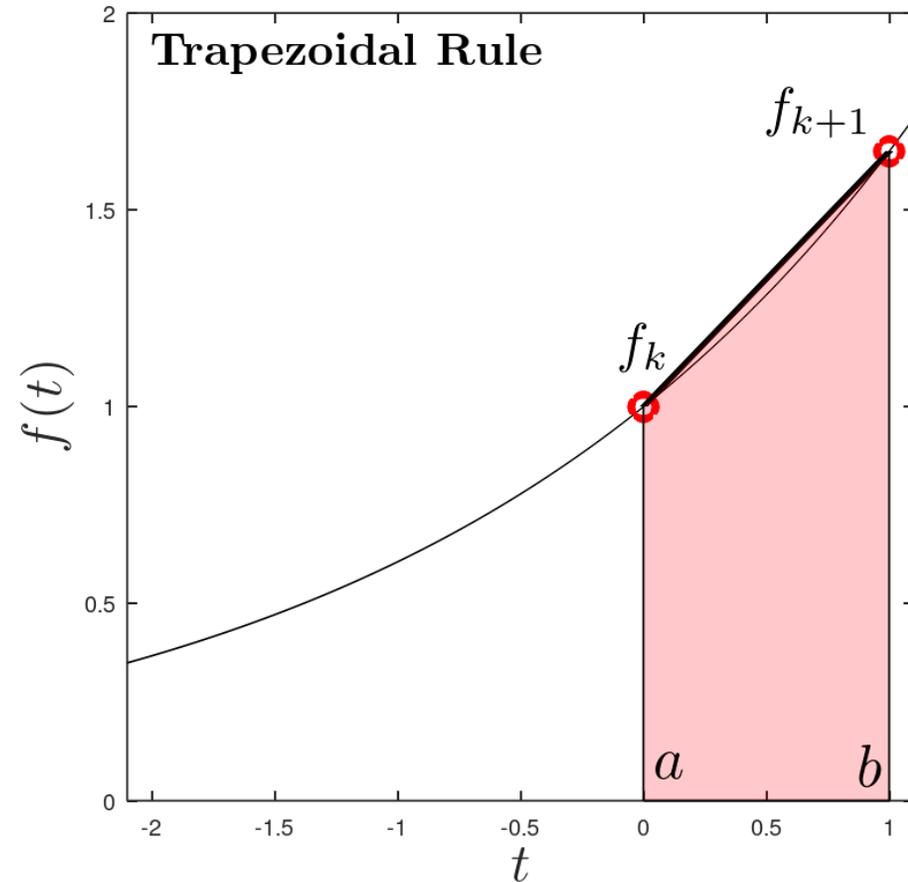
- EB, while stable, still suffers from $O(h)$ accuracy
- More rapidly convergent schemes can considerably reduce costs
- Although we derived EF and EB from forward/backward differences, we can also view them as integration-based rules using 0th-order (i.e., constant) polynomial approximations to f .
- Yields “rectangle rule” with $f(t_k, y_k)$ for EF and $f(t_{k+1}, y_{k+1})$ for EB



- We can see that the LTE is $O(\Delta t^2)$ in either case, which means that these methods are *first-order* (i.e., $\text{GTE} = O(\Delta t)$)

Trapezoidal Rule

- One can improve the accuracy by using the *trapezoidal rule*, which clearly has an LTE of $O(\Delta t^3)$, meaning that trapezoidal rule is $O(\Delta t^2)$ accurate (i.e., second-order accurate)



- The trapezoidal update reads

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

which is the average of EB and EF

Trapezoidal Rule, continued

- The trapezoidal rule is *implicit*, so cost is same as EB, but accuracy is much improved

- If $\mathbf{f} = \mathbf{A}\mathbf{y} + \mathbf{g}(t)$ is *linear* we can write the update as

$$\left(\mathbf{I} - \frac{h}{2}\mathbf{A}\right)\mathbf{y}_{k+1} = \left(\mathbf{I} + \frac{h}{2}\mathbf{A}\right)\mathbf{y}_k + \frac{1}{2}(g_k + g_{k+1})$$

which entails solving a system of the form $\mathbf{H}\mathbf{y} = (\mathbf{I} - \frac{h}{2}\mathbf{A})\mathbf{y} = \hat{\mathbf{g}}$ at every step, as opposed to the EB update of the form $\mathbf{H}\mathbf{y} = (\mathbf{I} - h\mathbf{A})\mathbf{y} = \hat{\mathbf{g}}$

Trapezoidal Rule, continued

- From the preceding update, we can see that the Trapezoidal rule for the *model problem*, $y' = \lambda y$ yields

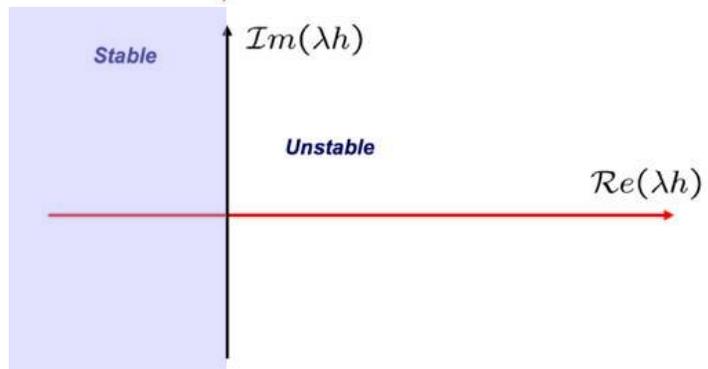
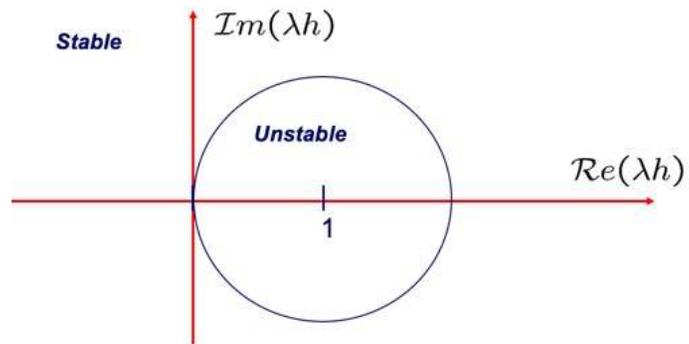
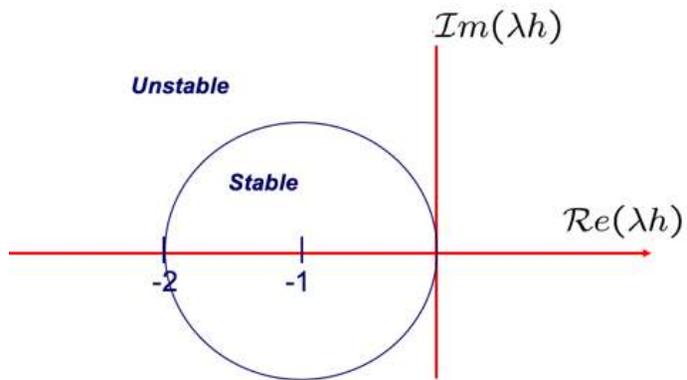
$$\begin{aligned}y_{k+1} &= \frac{1 + \frac{1}{2}h\lambda}{1 - \frac{1}{2}h\lambda} y_k \\ &= G_T y_k\end{aligned}$$

with

$$G_T = \frac{1 + \frac{1}{2}h\lambda}{1 - \frac{1}{2}h\lambda}$$

- The growth factor has modulus $\equiv 1$ whenever $Re(\lambda) = 0$, which is consistent with the analytical growth factor, $\tilde{G} = e^{h\lambda}$
- Trapezoidal rule seems perfect, but has one major flaw in that it is not L -stable, which we discuss shortly

Stability Regions: $|G| < 1$



- Euler Forward
- Euler Backward
- Trapezoidal Rule

Growth Factors

- **Growth factor \mathbf{G}** is the multiplier of the solution to the homogeneous problem $y' = \lambda y$ that you observe after one timestep of size Δt .
- They are one of the fundamental properties of a method (or *timestepper*) for numerical solution of ODEs.
- For small $\lambda\Delta t$ they should approximate the analytical solution, $e^{\lambda\Delta t}$:

$$y_{k+1} = e^{\lambda\Delta t} y_k = \tilde{\mathbf{G}} y_k.$$

- For stability, they must have $|G| \leq 1$.
- We look at some examples of growth factors in the next couple of slides.

Growth Factors

$$\mathbf{Exact:} \quad \tilde{\mathbf{G}} = e^{\lambda\Delta t} = 1 + \lambda\Delta t + \frac{(\lambda\Delta t)^2}{2} + \frac{(\lambda\Delta t)^3}{3!} + \frac{(\lambda\Delta t)^4}{4!} + \dots$$

$$\mathbf{EF:} \quad \mathbf{G} = 1 + \lambda\Delta t$$

$$\mathbf{EB:} \quad \mathbf{G} = \frac{1}{1 - \lambda\Delta t} = 1 + \lambda\Delta t + (\lambda\Delta t)^2 + (\lambda\Delta t)^3 + \dots$$

$$\mathbf{Trap:} \quad \mathbf{G} = \frac{(1 + \frac{1}{2}\lambda\Delta t)}{(1 - \frac{1}{2}\lambda\Delta t)} = 1 + \lambda\Delta t + \frac{(\lambda\Delta t)^2}{2} + \frac{(\lambda\Delta t)^3}{4} + \dots$$

- For a p th-order method (i.e., GTE is $O(\Delta t^p)$), $\mathbf{G}(\lambda\Delta t)$ will agree with $\tilde{\mathbf{G}}(\lambda\Delta t)$ up through the power $(\lambda\Delta t)^p$.
- $\mathbf{G}(\lambda\Delta t)$ and $\tilde{\mathbf{G}}(\lambda\Delta t)$ will *differ* in the coefficient in front of $(\lambda\Delta t)^{p+1}$, which indicates that the *local truncation error* (LTE) is $(\lambda\Delta t)^{p+1}$.
- **EF** and **EB** agree up to the linear terms, implying that they are of order $p = 1$, while **Trap** agrees to the $(\lambda\Delta t^2)$ term, implying it is of order $p = 2$.

Growth Factors

$$\text{Exact:} \quad \tilde{\mathbf{G}} = e^{\lambda\Delta t} = 1 + \lambda\Delta t + \frac{(\lambda\Delta t)^2}{2} + \frac{(\lambda\Delta t)^3}{3!} + \frac{(\lambda\Delta t)^4}{4!} + \dots$$

$$\text{EF:} \quad \mathbf{G} = 1 + \lambda\Delta t$$

$$\text{EB:} \quad \mathbf{G} = \frac{1}{1 - \lambda\Delta t} = 1 + \lambda\Delta t + (\lambda\Delta t)^2 + (\lambda\Delta t)^3 + \dots$$

$$\text{Trap:} \quad \mathbf{G} = \frac{(1 + \frac{1}{2}\lambda\Delta t)}{(1 - \frac{1}{2}\lambda\Delta t)} = 1 + \lambda\Delta t + \frac{(\lambda\Delta t)^2}{2} + \frac{(\lambda\Delta t)^3}{4} + \dots$$

- **Note** that these Taylor series expansions are accurate for $\lambda\Delta t \rightarrow 0$, but not for large $\lambda\Delta t$, where the behavior can be quite different from $e^{\lambda\Delta t}$.
- The difference between \mathbf{G} and $\tilde{\mathbf{G}}$ in the large and small $\lambda\Delta t$ limits are the principal characterizations of a timestepper.
 - Behavior for small $\lambda\Delta t$ governs *order of accuracy*.
 - Behavior for large $\lambda\Delta t$ governs *stability of the timestepper*.

Growth Factors

- **Q:** What are the first 5 terms in the Taylor series for the growth factor of a 4th-order Runge-Kutta scheme?

- **A:** Same as the first 5 terms in $\tilde{\mathbf{G}}$.

$$\mathbf{G} = 1 + \lambda\Delta t + \frac{(\lambda\Delta t)^2}{2} + \frac{(\lambda\Delta t)^3}{3!} + \frac{(\lambda\Delta t)^4}{4!} + c_5(\lambda\Delta t)^5 + \dots,$$

with $c_5 \neq \frac{1}{5!}$.

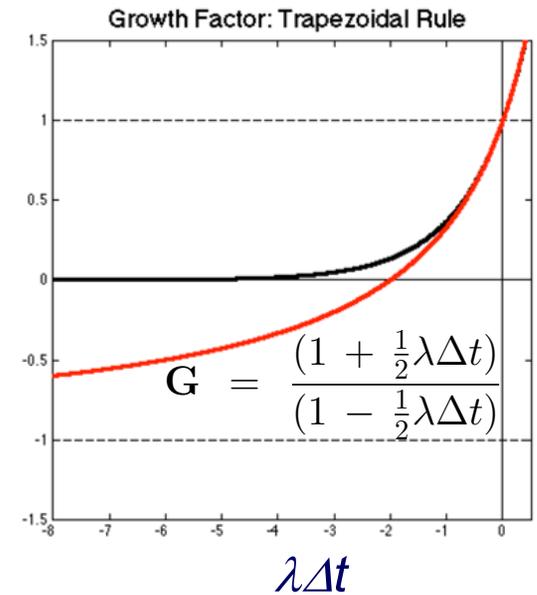
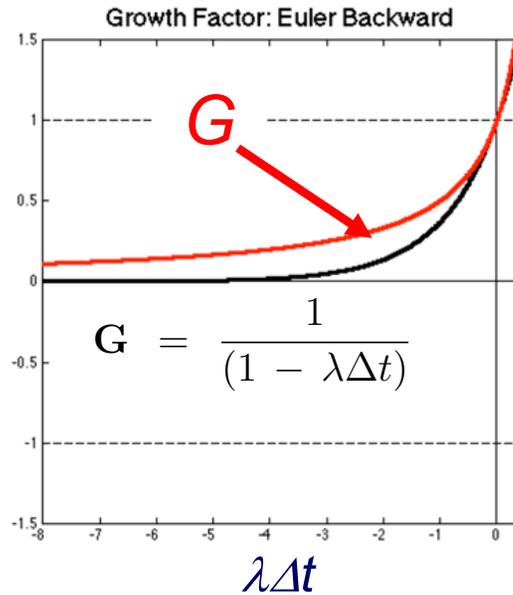
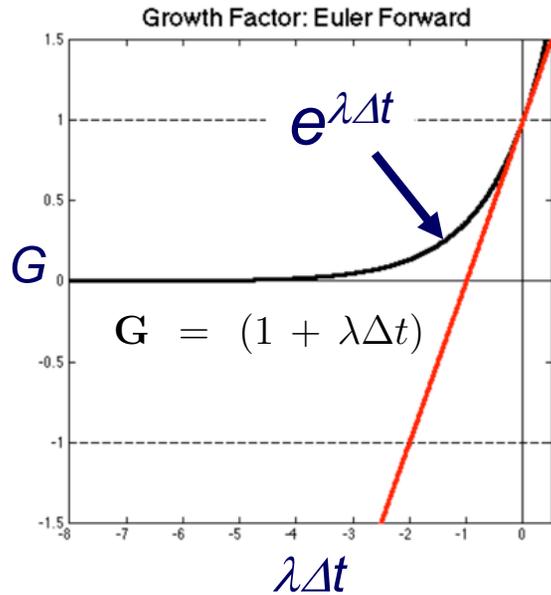
- The scheme is 4th-order:

→ GTE is $O(\Delta t^4)$

→ LTE is $O(\Delta t^5)$

∴ Taylor series for \mathbf{G} agrees with $\tilde{\mathbf{G}}$ up to and including (Δt^4) term, but not (Δt^5) term.

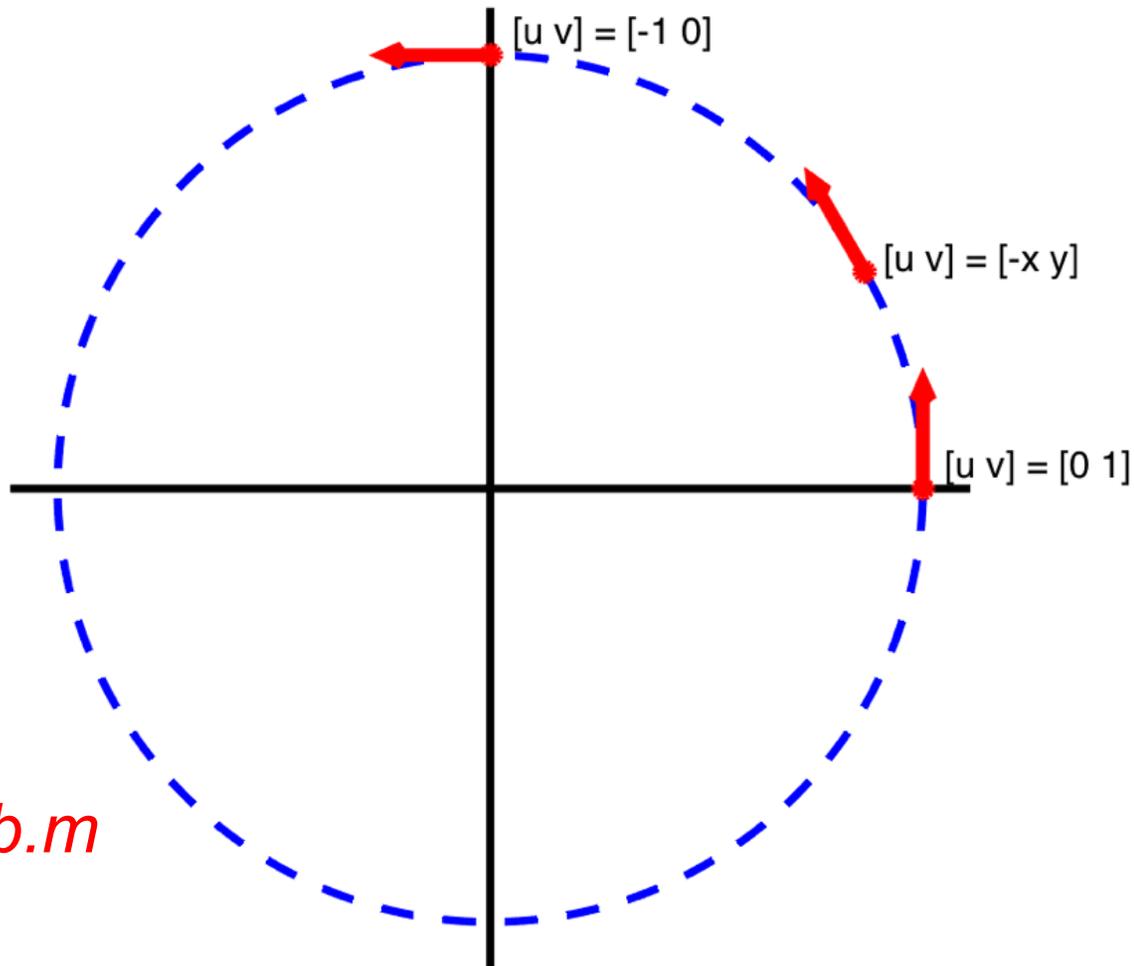
Growth Factors for Real λ



- Each growth factor approximates $e^{\lambda\Delta t}$ for $\lambda\Delta t \rightarrow 0$
- For EF, $|G|$ is not bounded by 1
- For Trapezoidal Rule, local (small Δt) approximation is $O(\Delta t^2)$, 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 $\lambda\Delta t \rightarrow -\infty$.

Example: Orbit Problem

$$\begin{bmatrix} u \\ v \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix}}_{\frac{dy}{dt}} = \underbrace{\begin{bmatrix} -x \\ y \end{bmatrix}}_{f(\mathbf{y})} = \underbrace{\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}}_{A\mathbf{y}} \begin{bmatrix} x \\ y \end{bmatrix}$$



orbit_ef_eb.m

EF, EB, and Trap: Orbit Example

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

$$\mathbf{EF:} \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{\Delta t} = A\mathbf{y}_k$$

$$\mathbf{EB:} \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{\Delta t} = A\mathbf{y}_{k+1}$$

$$\mathbf{Trap:} \quad \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{\Delta t} = \frac{1}{2} [A\mathbf{y}_k + A\mathbf{y}_{k+1}]$$

Solve for \mathbf{y}_{k+1} :

$$\mathbf{EF:} \quad \mathbf{y}_{k+1} = [I + \Delta t A] \mathbf{y}_k$$

$$\mathbf{EB:} \quad \mathbf{y}_{k+1} = [I - \Delta t A]^{-1} \mathbf{y}_k$$

$$\mathbf{Trap:} \quad \mathbf{y}_{k+1} = \left[I - \frac{\Delta t}{2} A \right]^{-1} \left[I + \frac{\Delta t}{2} A \right] \mathbf{y}_k$$

Eigenvalues of A : $\lambda = \pm i$

orbit_ef_eb.m

```
if eb>0;                %% Euler Backward
    pause;
    y=y0;
    I=eye(2);
    H=I-dt*A; Hi=inv(H);
    for k=1:n;
        yo=y;
        y=Hi*y;
        plot([yo(1) y(1)],[yo(2) y(2)],'b-',lw,2); hold on;
        plot([yo(1) y(1)],[yo(2) y(2)],'b.',ms,9);
        drawnow
    end;
    plot(y(1),y(2),'kx',lw,2); axis square; hold on
    title('Euler Forward/Backward',fs,14)
end;

if trap > 0;           %% Trapezoidal Rule
    pause
    y=y0;
    I=eye(2);
    Hl=I-0.5*dt*A; Hr=I+0.5*dt*A;
    G = inv(Hl)*Hr;
    for k=1:n;
        yo=y;
        y=G*y;
        plot([yo(1) y(1)],[yo(2) y(2)],'g.',ms,9); drawnow
    end;
    plot(y(1),y(2),'kx',lw,2); axis square; hold on
    title('Trapezoid, EF, EB',fs,14)
end;
```

Alternative Timesteppers: $\frac{d\mathbf{u}}{dt} = \mathbf{f}$

- Higher order? $O(\Delta t^k)$, $k = 2, 3, \dots$
- Stepsize?
Large step generally preferable, but total cost is $n_{\text{steps}} \times \text{work-per-step}$.
- Implicit?
- Explicit?
- Single-step (multistage)
- Multi-step
Requires bootstrapping to get $(\mathbf{u}^{n-j}, \mathbf{f}^{n-j})$, $j = 1, \dots, k$.

For all choices, we are very concerned with the stability region for the method, as well as the asymptotic accuracy.

BDF k Formulas, GTE = $O(h^k)$

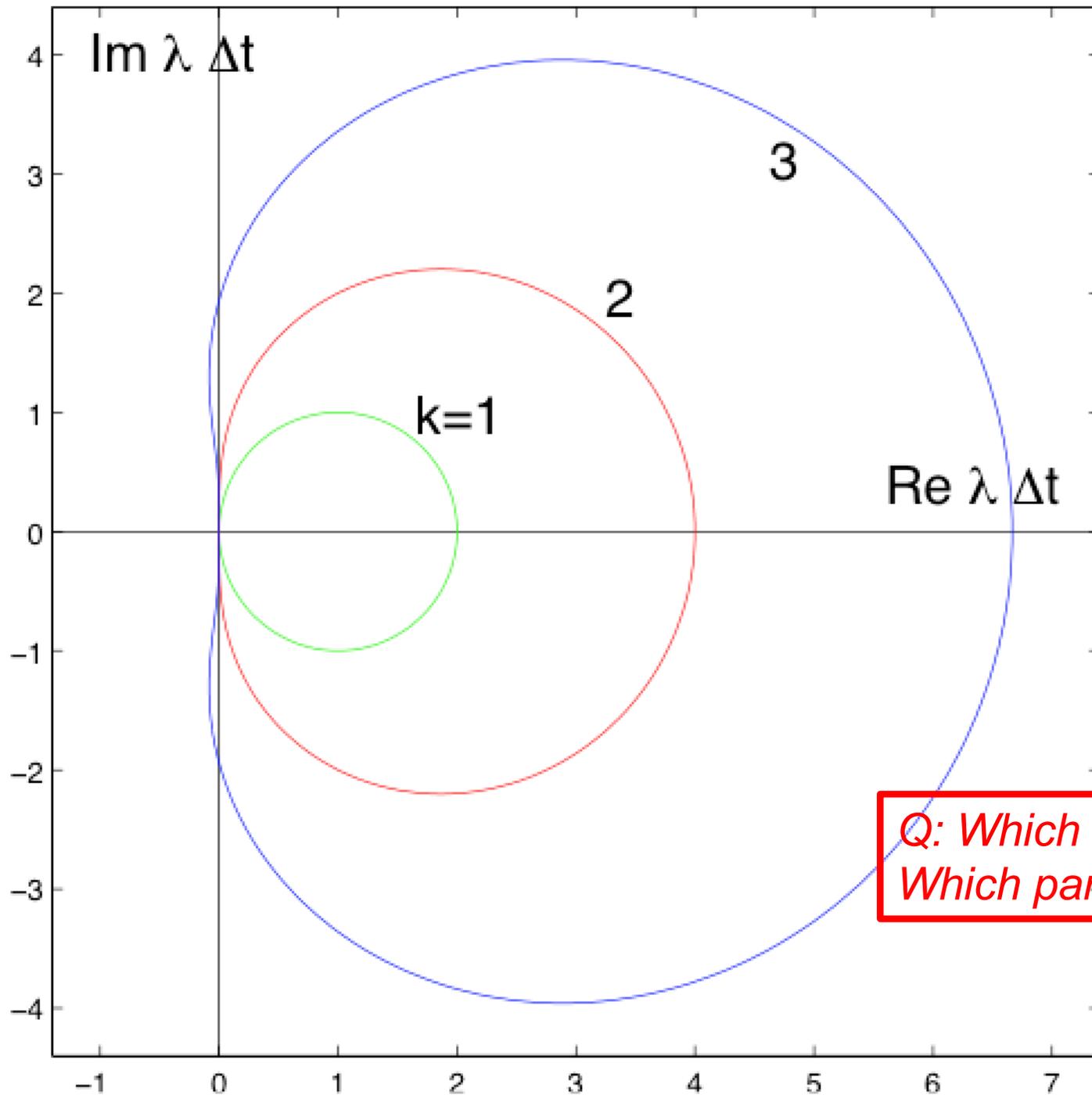
$$\text{BDF1: } \left. \frac{\partial u}{\partial t} \right|_{t^n} = \frac{u^n - u^{n-1}}{\Delta t} + O(\Delta t)$$

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

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

- These methods are L -stable: $|G| \rightarrow 0$ as $h\lambda \rightarrow -\infty$
- k th-order accurate
- Implicit
- Unconditionally stable only for $k < 3$ (here, $k :=$ order of method)
- Multi-step: require data from previous timesteps

BDFk Neutral Stability Curve



*Q: Which is stable?
Which part is unstable?*

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

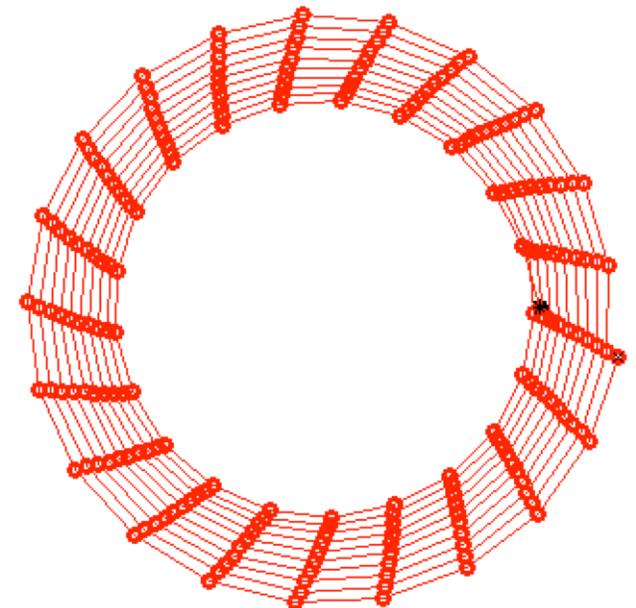
y=y0; y2=y; y1=y; y3=y;

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
```



STOPPED HERE

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

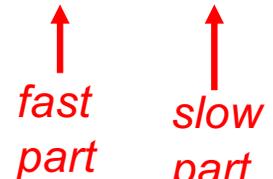
- Trap: $(I - \frac{h}{2}J)\mathbf{y}_{k+1} = (1 + \frac{h}{2}J)\mathbf{y}_k$

- 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

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

- 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:

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

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

- 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.

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 --- small particle example).
 - More accuracy
 - For $k > 2$, stability region 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:

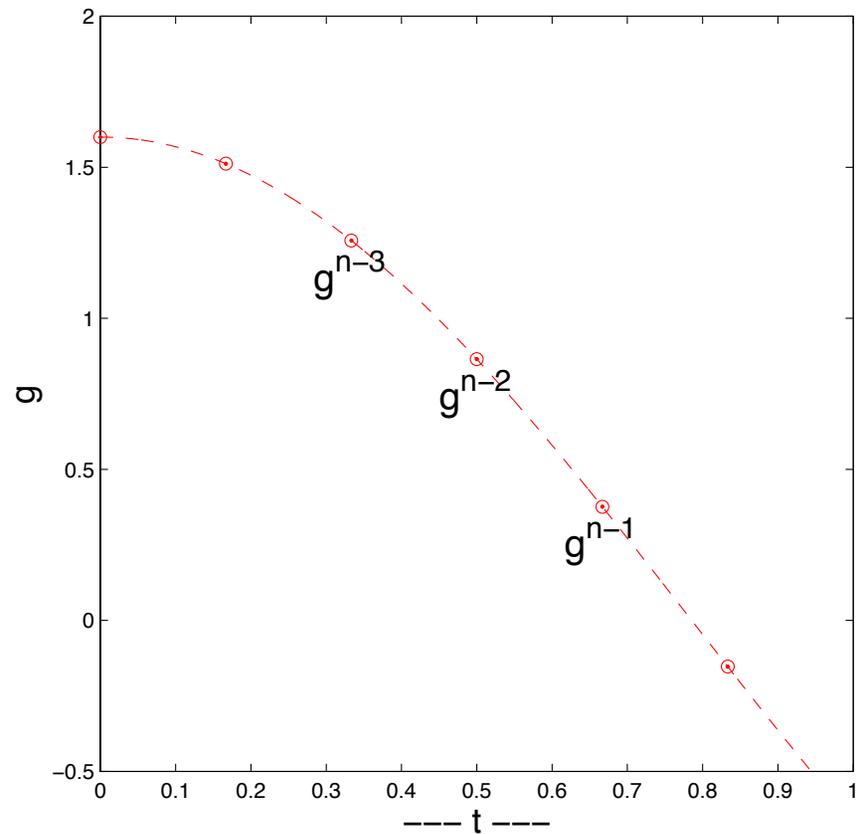
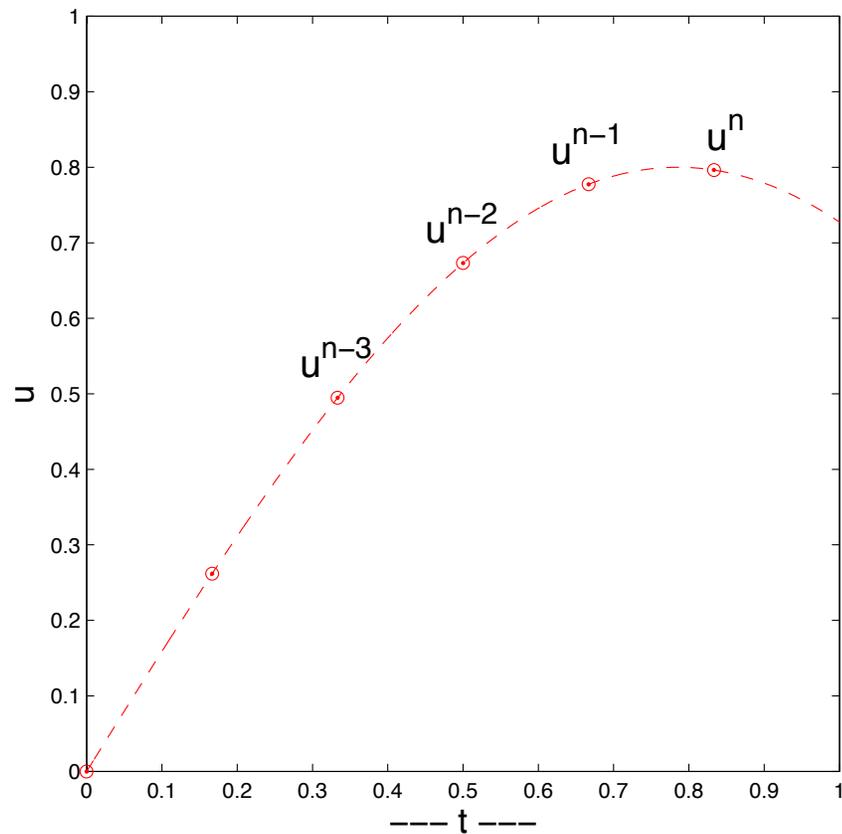
$$y_{k+1} = \frac{2}{3} \left[\frac{4y_k - y_{k-1}}{2} + \Delta t(2f_k - f_{k-1}) \right] + O(\Delta t^3)$$

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

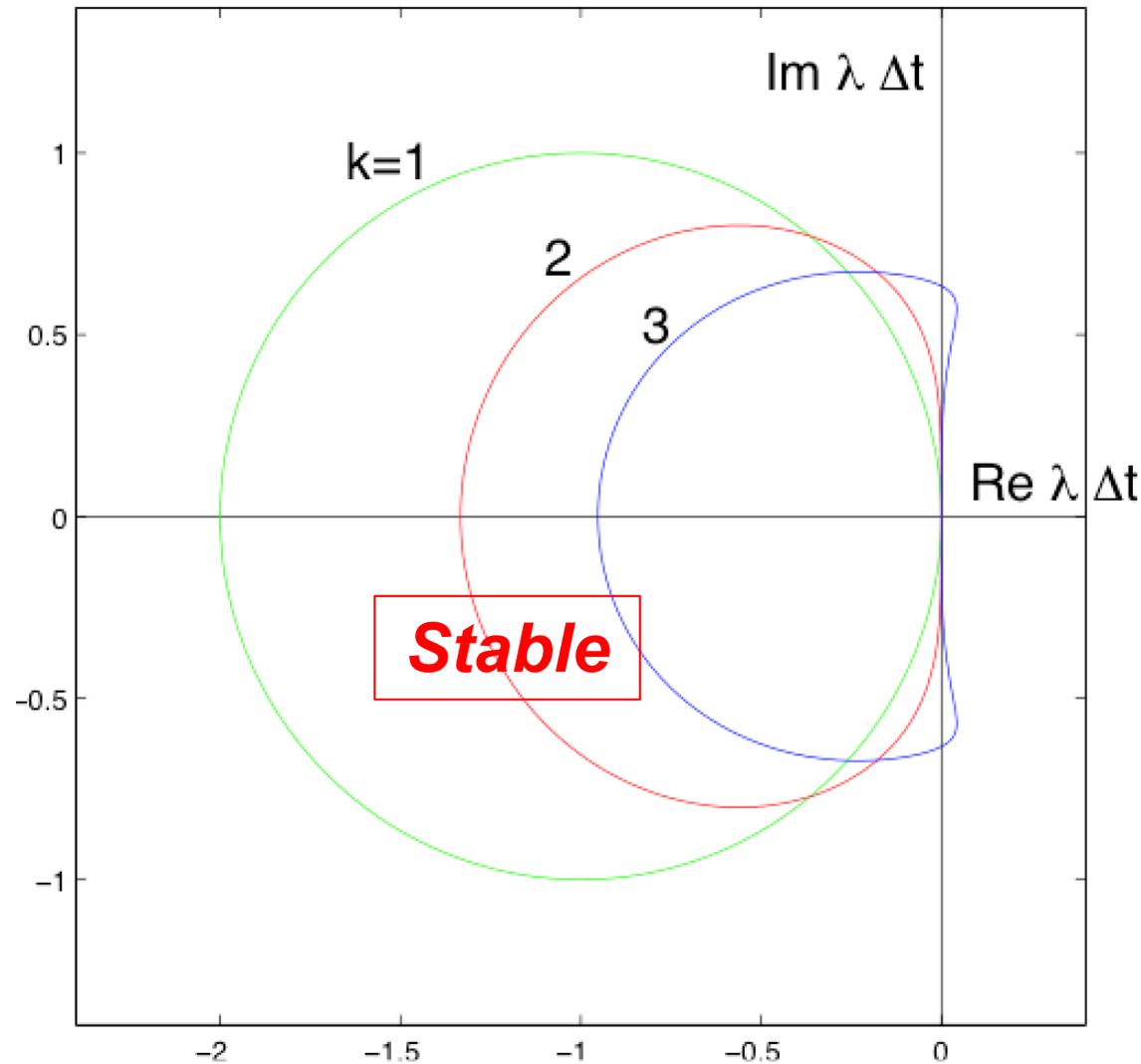
BDF k -EXT k Example

$$\frac{3u^n - 4u^{n-1} + u^{n-2}}{2\Delta t} + O(\Delta t^2) = (2g^{n-1} - g^{n-2} + O(\Delta t^2)) + f(u^n, t^n)$$

- Use *old known values* of $u(t)$, plus unknown u^n to estimate $u'(t^n)$
- Use *old known values* of $g(t, u)$ to estimate *rhs*, g^n



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 (i.e., those having imaginary eigenvalues).

Higher-Order Explicit Timesteppers: k th-order Adams-Bashforth

- Adams-Bashforth methods are a somewhat simpler alternative to BDF k /EXT k .
- Time advancement via integration:

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

- AB1:

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

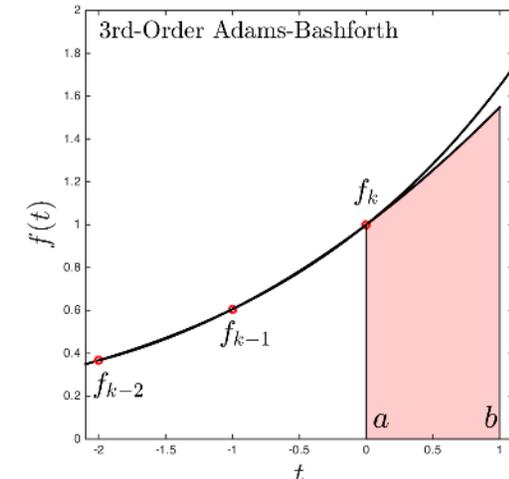
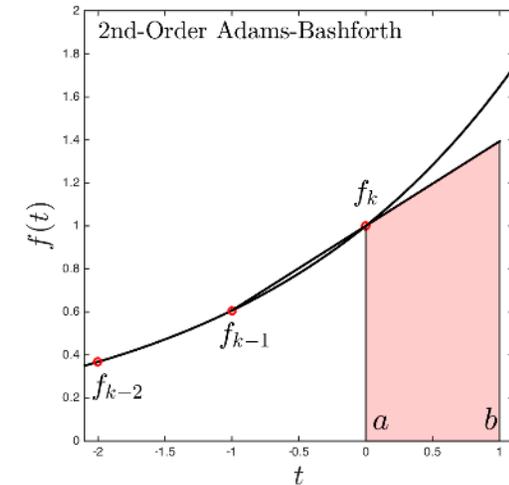
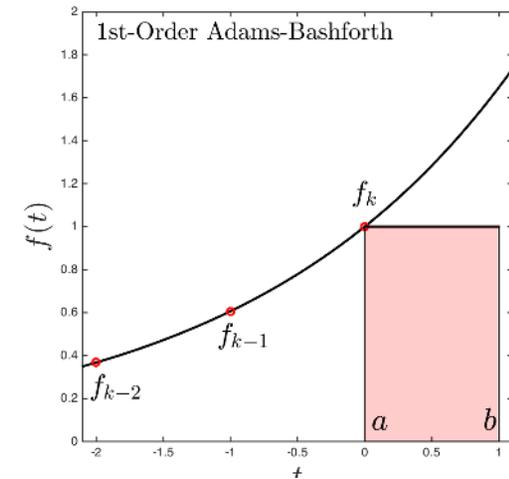
- AB2:

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

- AB3:

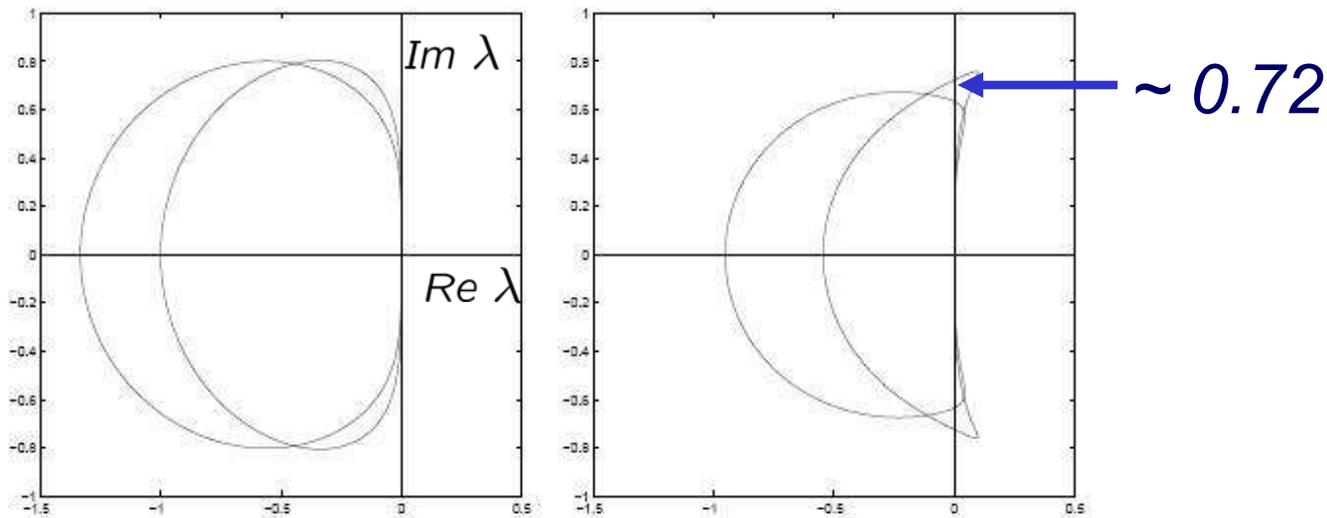
$$\int_{t_k}^{t_{k+1}} \mathbf{f}(t, \mathbf{y}) dt = h \left(\frac{23}{12} \mathbf{f}_k - \frac{16}{12} \mathbf{f}_{k-1} + \frac{5}{12} \mathbf{f}_{k-2} \right) + O(h^4) \text{ (if } h \text{ is constant)}$$

- 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)



Stability Region for AB2/BDF2-Ext2 (left) and AB3/BDF3-Ext3 (right)

- Example: If $\lambda = i$, we need $\Delta t < 0.72$ for AB3 to be stable
- For BDF3/Ext3, need $\Delta t < \sim 0.61$

Determining the Neutral-Stability Curve

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

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

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

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

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

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

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

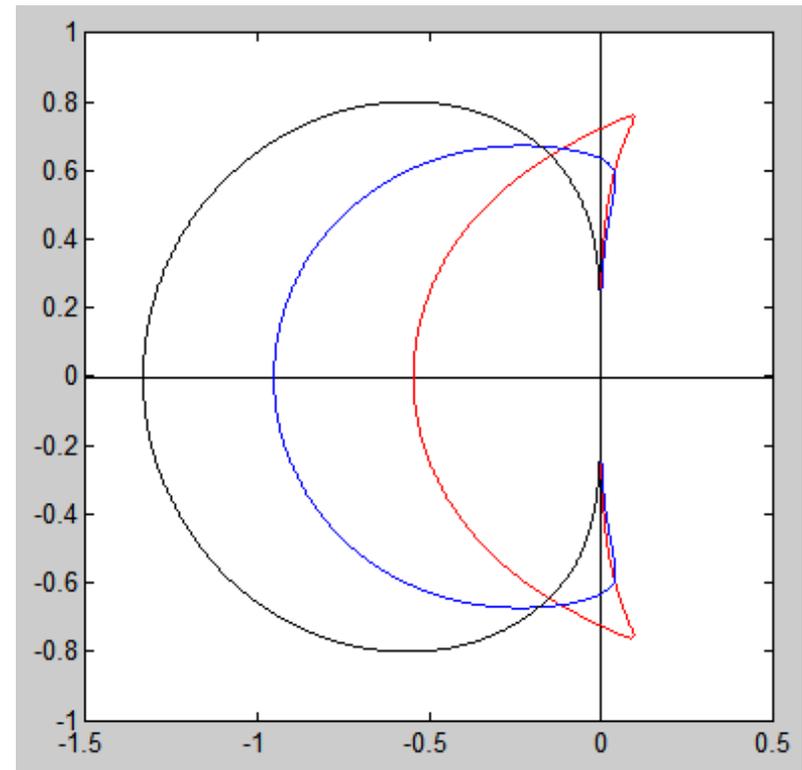
Matlab Code: stab.m

```
ymin=-1; ymax=1; ep=1.e-13; yaxis=[-ymax*ii ymax*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([-ymax-.5 ymax-.5 -ymax ymax]);

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
```



Runge-Kutta Methods

- Runge-Kutta methods are derived from integration rules of the form

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

and can be either *implicit* or *explicit*, with explicit being most common.

- RK schemes can be high order
- They are *multi-stage* rather than *multi-step*, meaning that they evaluate \mathbf{f} at multiple points (stages) on the interval $[t_k, t_{k+1}]$, but do not require information from prior timesteps, as would be the case for a multi-step method
- As such, RK schemes are *self-starting*
- It is relatively easy to change the step size and they are relatively easy to program

Classic Fourth-Order Runge-Kutta Method

- The most common RK scheme is explicit RK4, which is 4th-order accurate
- The update begins with four function evaluations,

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

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

$$\mathbf{k}_3 = \mathbf{f}(t_{k+\frac{1}{2}}, \mathbf{y}_k + \frac{h_k}{2}\mathbf{k}_2)$$

$$\mathbf{k}_4 = \mathbf{f}(t_{k+1}, \mathbf{y}_k + h_k\mathbf{k}_3)$$

where $h_k := t_{k+1} - t_k$ and $t_{k+\frac{1}{2}} := t_k + h_k/2$

- These stages are then combined to advance the solution

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

which is a quadrature rule similar to the $O(h^4)$ accurate Simpson's rule.

RK4 Example

rk4a.m

```
Tfinal=pi/2;

for kk=1:10; nsteps=2^kk; dt=Tfinal/nsteps;

    % nsteps=ceil(Tfinal/dt); dt=Tfinal/nsteps;

    ftype=0; y=0; % Scalar ODE y=sin(t) --> y'=cos(t);, y(t=0)=0
    t=0; % Intial time = 0

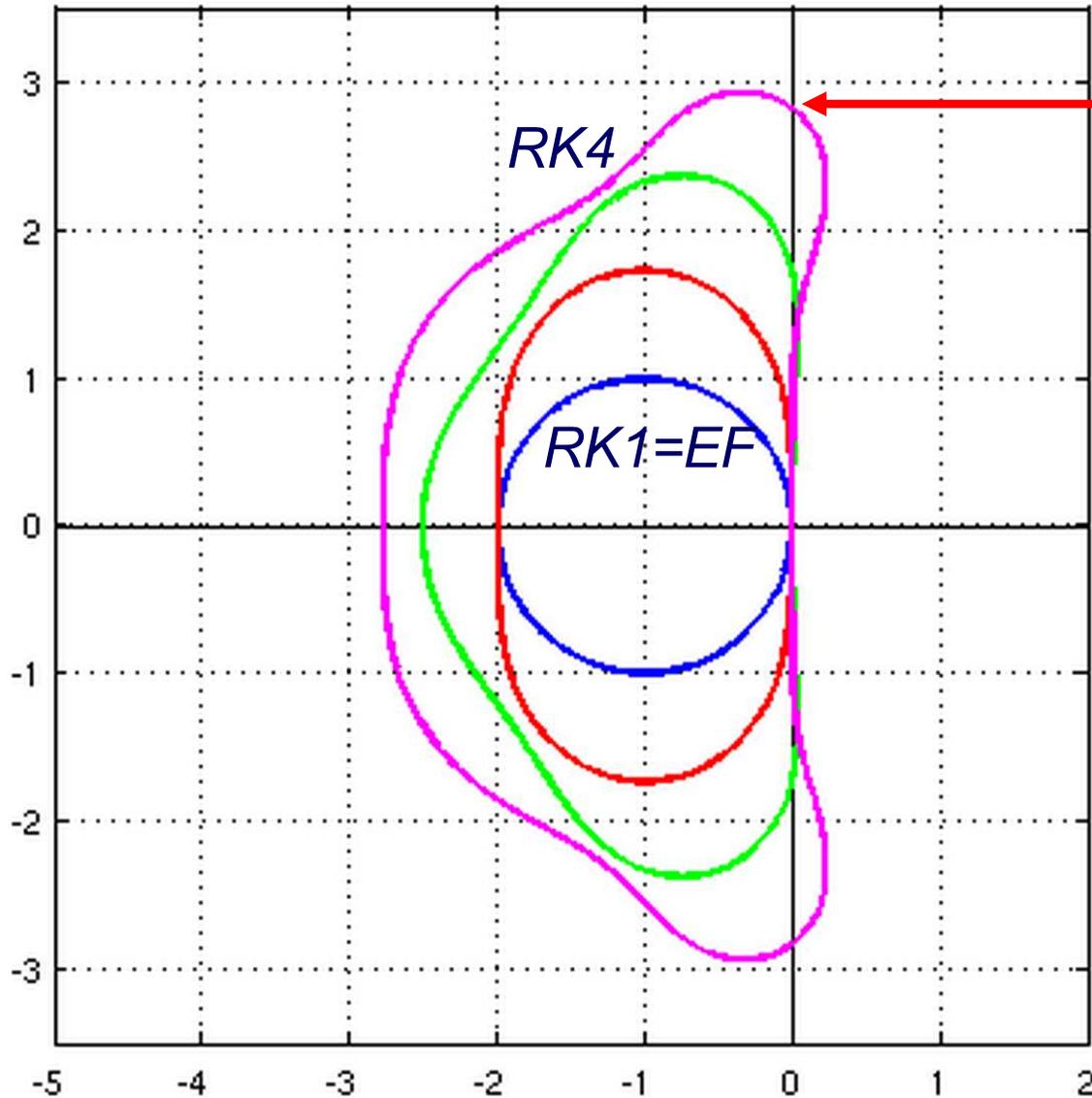
    for k=1:nsteps

        d2 = dt/2;          % half-step
        k1 = rkf(ftype,t,y);
        k2 = rkf(ftype,t+d2,y+d2*k1);
        k3 = rkf(ftype,t+d2,y+d2*k2);
        k4 = rkf(ftype,t+dt,y+dt*k3);
        y = y + dt*(k1+2*(k2+k3)+k4)/6;

        t      = t+dt;
        exact  = sin(t);
        error(k) = abs(exact-y);
    end;
    nn(kk) = nsteps;
    dn(kk) = dt;
    en(kk) = error(k);
end;

model = dn.^4;
loglog(dn,model,'k-',lw,2,dn,en,'r--',lw,2);
```

Stability Regions for RK1—4



- 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)

Time Step Selection

- Assuming $\lambda\Delta t$ 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.

Adaptive RK4

- *Adaptive* timesteppers will adjust the timestep size based on *local error estimates*
- These schemes work remarkably well and maintain stability and accuracy in an efficient way
- If the system is *stiff*, you still need an implicit method or the scheme will be forced to take small timesteps
- For non-stiff systems, RK4 is an excellent starting point

Adaptive RK4, continued

- The error estimates are based on using two timesteppers, with *different local error behavior*, to advance from t_k to t_{k+1}

- For example, with RK4, we have local truncation error,

$$LTE_{RK4} \sim c_5 h^5 + O(h^6)$$

If we also have a 5th-order method, we could take a single step and have an error

$$LTE_{RK5} \sim c_6 h^6 + O(h^7) \ll LTE_{RK4}$$

- We can estimate c_5 by taking the difference between the two solutions
- RK5 schemes typically require 6 stages, so it would appear that we need a total of 10 function evaluations to get the error estimate.
- However, *embedded methods*, such as Fehlberg's RK4/5, allow one to evaluate both an RK4 and an RK5 update with only *six* function evaluations

Adaptive RK4, continued

- As an alternative to embedded schemes can simply take three RK4 steps, one with step size h_k to yield $\hat{\mathbf{y}}_{k+1}$, and two with step size $h_k/2$ to yield \mathbf{y}_{k+1}
- Since the first stage evaluation is the same whether using h_k or $h_k/2$, we need only 11 function evaluations per step with this approach
- Let \mathbf{u}_{k+1} be the “true” (unknown) solution and take the difference

$$\hat{\mathbf{y}}_{k+1} = \mathbf{u}_{k+1} + c_5 h^5 + O(h^6)$$

$$\mathbf{y}_{k+1} = \mathbf{u}_{k+1} + 2c_5 \frac{h^5}{32} + O(h^6)$$

$$\Delta_k := \hat{\mathbf{y}}_{k+1} - \mathbf{y}_{k+1} \approx \frac{15}{16} c_5 h^5$$

- Thus, we have a way of estimating the local error: $|c_5 h^5| \approx \|\Delta_k\|_*$, where $\|\cdot\|_*$ is some appropriately weighted measure of the difference

Adaptive RK4, continued

- For an adaptive scheme, we typically want to bound the *global error* at time $t_{final} = T$ by, say, tol
- If Δ_k is our local error (LTE), committed on *each step*, then we need

$$\Delta_k \cdot n \lesssim tol$$

- For stepsize \tilde{h} , $n \approx T/\tilde{h}$ steps and therefore require

$$\frac{\Delta_k \cdot T}{\tilde{h}} \approx \frac{c_5 \tilde{h}^5}{\tilde{h}} \lesssim tol$$

- Solving for requisite stepsize, \tilde{h} ,

$$\tilde{h} \approx \left(\frac{tol}{T \cdot c_5} \right)^{\frac{1}{4}} \approx \left(\frac{tol \cdot h_k^5}{T \cdot \Delta_k} \right)^{\frac{1}{4}} = S \cdot h_k,$$

with

$$S = \left(\frac{tol \cdot h_k}{T \cdot \Delta_k} \right)^{\frac{1}{4}}$$

Adaptive RK4, continued

- So, if $h_k > \tilde{h}$ (current step too large),
 - Do not advance \mathbf{y}_k
 - Set $h_k = \tilde{h}$ and retest.
- If current step $h_k \leq \tilde{h}$, then
 - Update $\mathbf{y}_k \longrightarrow \mathbf{y}_{k+1}$ (using the *accurate* $h_k/2$ update)
 - Set $h_{k+1} = \tilde{h}$
- Notes:
 - i.* Cannot go wrong by setting $h_{k+1} = \tilde{h}$ because the step size will be tested on next adaptive update step
 - ii.* Our estimates are conservative because they are based on the inaccurate h_k update, rather than the more accurate $h_k/2$ update employed in the actual update

Adaptive RK4, continued

- Should view adaptive process as:
 - i.* Estimate c_5 and adjust h
 - ii.* Advance, if LTE is deemed small enough
 - iii.* Else, go to *i.*

Adaptive Examples

- Here, we will consider two examples
- The first is the orbit example we've seen before, for which we know the exact answer
 - This is a *kinematic* example in that the motion is *prescribed* (cinématique, in French)
 - It is a first-order system with two unknowns, $[x(t), y(t)]$
- The second is to predict the orbit of a satellite orbiting a planet
 - This is a *dynamic* example, in which the motion is determined by external forces (here, gravity)
 - Here, we have 4 unknowns $[x, y, \dot{x}, \dot{y}]$, that is, two components of position and two of velocity

Adaptivity for Orbit Example

- This is the case we saw before, with IC $\mathbf{y}_0 = [x_0 \ y_0]^T$ and

$$\underbrace{\begin{bmatrix} x' \\ y' \end{bmatrix}}_{\mathbf{y}'} = \underbrace{\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{\mathbf{y}}$$

- The ODE describes the position of a particle traveling in a circle with time period $\tau = 2\pi$.
- So, for any integer $k > 0$, we can expect the particle to return to its original position at time $t_{final} = T = k \cdot 2\pi$.
- We can choose $\Delta t = T/n$ and study the convergence $err_T = \|\mathbf{y}_n - \mathbf{y}_0\|_2$ as a function of n
- Or, for the purposes of this exercise, choose tol and see if the error at time T is $\leq tol$, per our adaptive strategy

Example: Adaptivity for Dynamic Case

With help from a couple of Newton's Laws, we have

- $\mathbf{a} = \frac{1}{m}\mathbf{F}$

- $\mathbf{F} = -\frac{GMm}{r^2} \frac{\mathbf{r}}{\|\mathbf{r}\|}$

- With position $[x \ y]^T$ and velocity $[x' \ y']^T = [u \ v]^T$, the system is

$$\underbrace{\begin{bmatrix} x' \\ y' \\ u' \\ v' \end{bmatrix}}_{\mathbf{y}'} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{c}{r^3} & 0 & 0 & 0 \\ 0 & -\frac{c}{r^3} & 0 & 0 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x \\ y \\ u \\ v \end{bmatrix}}_{\mathbf{y}}$$

- Note that gravity point in the direction $-\mathbf{r}$.

- Here we just take $c = 1$

Example: Adaptivity for Dynamic Case, continued

- This case, in general, has *elliptic orbits*, with the origin at one of the foci
- We do not know the exact answer in this case, but there are metrics one could readily check
- The key point is that if the eccentricity is large then the velocity variation is large and the system can easily go unstable if Δt is fixed
- Adaptive timesteppers are essential for this system
- Large steps are most efficient when the velocity is relatively small
- Small steps are required for stability when the velocity is large, which occurs when r is small, as we might expect by inspecting \mathbf{A}

Example: Setting tol

- Here, our solution has *mixed units*, involving *position* and *velocity*
- How then, should we set the tolerance?
- Suppose the (unkown) velocities are enormous? What is the correct tolerance?
- Fortunately, we do not need to set tolerances on all quantities if we are interested only in, say, the final position
- Therefore, we can define $\Delta_k = \|\hat{\mathbf{y}}_{k+1}[1 : 2] - \mathbf{y}_{k+1}[1 : 2]\|_2$, which is just a measure of the LTE for *position*
- More generally, we could consider an $m \times m$ diagonal matrix \mathbf{D} and set

$$\Delta_k = \left(\delta \mathbf{y}_{k+1}^T \mathbf{D} \delta \mathbf{y}_{k+1} \right)^{\frac{1}{2}},$$

with $\delta \mathbf{y}_{k+1} := \hat{\mathbf{y}}_{k+1} - \mathbf{y}_{k+1}$, such that \mathbf{D} scales each error indicator

- One could also use a scaled ∞ -norm, if desired

RK4 Example

```
Tfinal = 5*2*pi;
dt = Tfinal; ftype=1;

% tol = 1.e-4;

y0 = [1 ; 0 ];
y = y0;

t=0;k=0;
while t < Tfinal; k=k+1;

    d2 = dt/2;          % half-step
    k1 = rkf(ftype,t,y);
    k2 = rkf(ftype,t+d2,y+d2*k1);
    k3 = rkf(ftype,t+d2,y+d2*k2);
    k4 = rkf(ftype,t+dt,y+dt*k3);
    y1 = y + dt*(k1+2*(k2+k3)+k4)/6;

    d2=dt/2; d4=d2/2; % 1st of 2 small steps, Re-use k1
    k2=rkf(ftype,t+d4,y+d4*k1);
    k3=rkf(ftype,t+d4,y+d4*k2);
    k4=rkf(ftype,t+d2,y+d2*k3);
    y2=y + d2*(k1+2*(k2+k3)+k4)/6;

    d3=d2+d4;          % Second of 2 small steps
    k1=rkf(ftype,t+d2,y2);
    k2=rkf(ftype,t+d3,y2+d4*k1);
    k3=rkf(ftype,t+d3,y2+d4*k2);
    k4=rkf(ftype,t+dt,y2+d2*k3);
    y2=y2+ d2*(k1+2*(k2+k3)+k4)/6;

% Local error estimate--based on position
delta=norm(abs(y2-y1),2);

% Check error estimate against GTE target
C = ( (dt*tol) / (delta*Tfinal) ).^0.25;

% Check error estimate against GTE target
C = ( (dt*tol) / (delta*Tfinal) ).^0.25;

if C < 1.0; % Current dt is too large
    dto= dt;
    dt = 0.9*C*dt; % Reset dt, but don't advance solution
% disp(['Reduce: ' int2str(k) ' ' num2str([dt dto])])
else % dt ok or too small
    t = t+dt; y = y2;
    k=k+1; yk(:,k)=y; ek(k)=delta; dk(k)=dt; tk(k)=t;
    C = min(1.5,C); % Allow only 10% dt increase
    dt=C*dt; dt=min(dt,Tfinal-t); % Don't exceed Tfinal
end;
mk = mod(k,iostep);
if mk==0
    plot(y(1),y(2),'r.',ms,12);
    axis equal; hold on; drawnow;
end;
tim(k) = t;
dtk(k) = dt;
ktp(k) = k;
end;
tim=tim(1:k);
dtk=dtk(1:k);
ktp=ktp(1:k);

error =norm(abs(y0-y),2);
disp(['Final step: ' int2str(k) ' Error & tol: ' num2str([error tol])])
```

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