Lecture 7:
Solving ODEs with Initial Values:
Stability Issues

some of this material is from “Scientific Computing” by Heath.

Initial Value ODEs

- Given some initial conditions and an ordinary differential equation, evolve the system in time (or whatever the independent variable is).

$$\frac{d\vec{r}}{dt} = \vec{f}(\vec{r}, t)$$

$$\vec{r}(0) = \vec{r}_0$$

- Note: as demonstrated in the first lab, we can always write an nth order ODE as n 1st order ODEs, so I'm not losing any generality by only considering the first order equations.

Scipy.integrate.odeint

- Python has a built in ODE solver called “odeint” located in the scipy.integrate module. (Aside: This module also contains a bunch of integration functions that can do gaussian quadrature, simpson’s rule etc.).

- more info on the module can be found here:

  http://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html

- It really functions as a black box and you don’t know how accurate your solution is (since you don’t know what method was used).

- If that doesn’t matter to your specific application, then just use odeint. However, if it does matter, then you can write your own ODE solver with the method that you want.

Choosing a Method

- Accuracy and speed are two important factors in choosing an ODE integrator method. However there is a third factor: STABILITY

- When choosing a method, you will want to consider the stability of the function being integrated AND the stability of the method used to integrate the function.

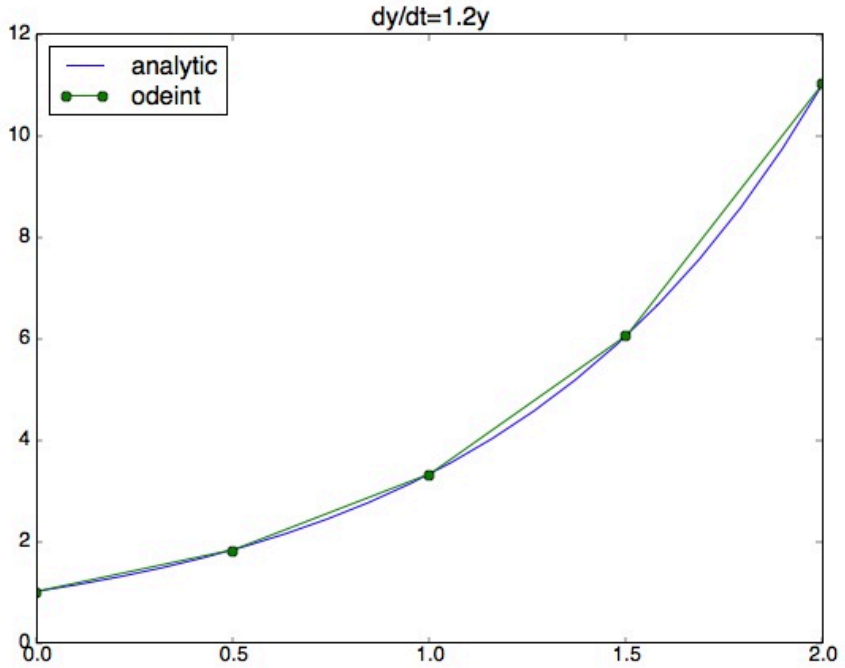
- Accuracy: how close do you stay to real solution?
- Stability: how do nearby solutions diverge from each other?

- Example:

$$\frac{dy}{dt} = 1.2y$$

$$y(0) = 1.0$$

used program: ex_odeint.py



adapted from Scinet Scientific Computing course

Function Stability

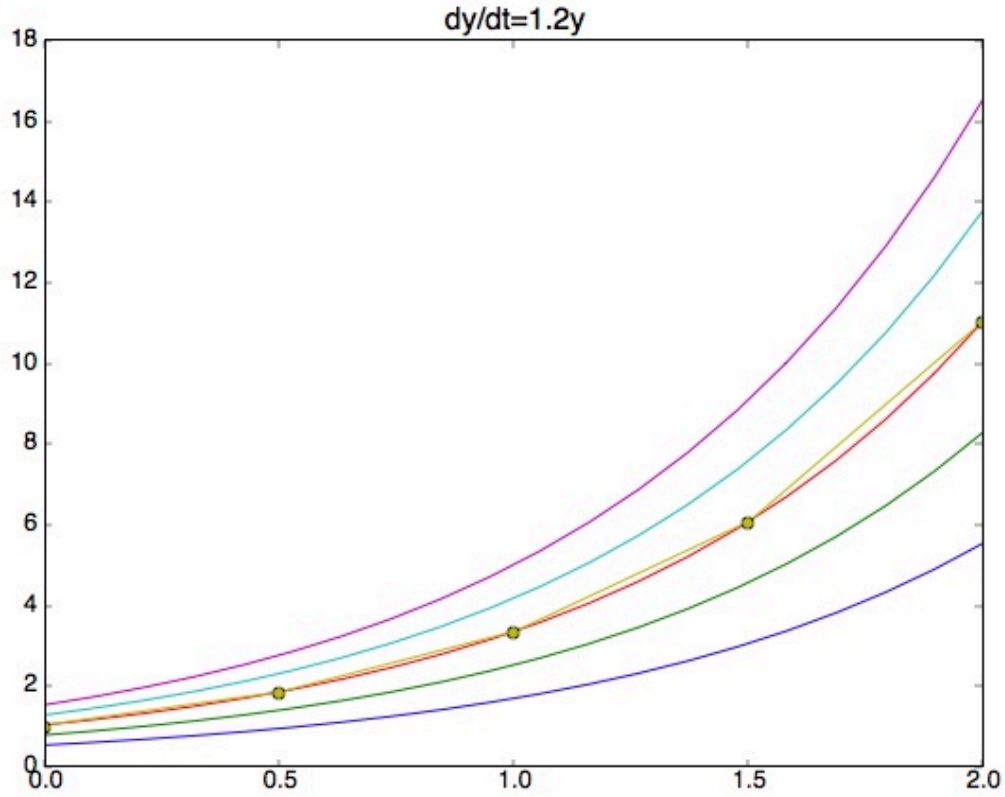
- Some systems are inherently challenging to integrate
- small deviations pull you further away from the solution
- since small errors will always creep in, this is very challenging for correctness.

- Notice how initial conditions that were close together spread further apart in time.

$$\frac{dy}{dt} = 1.2y$$

$$y(0) = 1.0$$

used program: ex_odeint2.py



adapted from Scinet Scientific Computing course

Function Stability

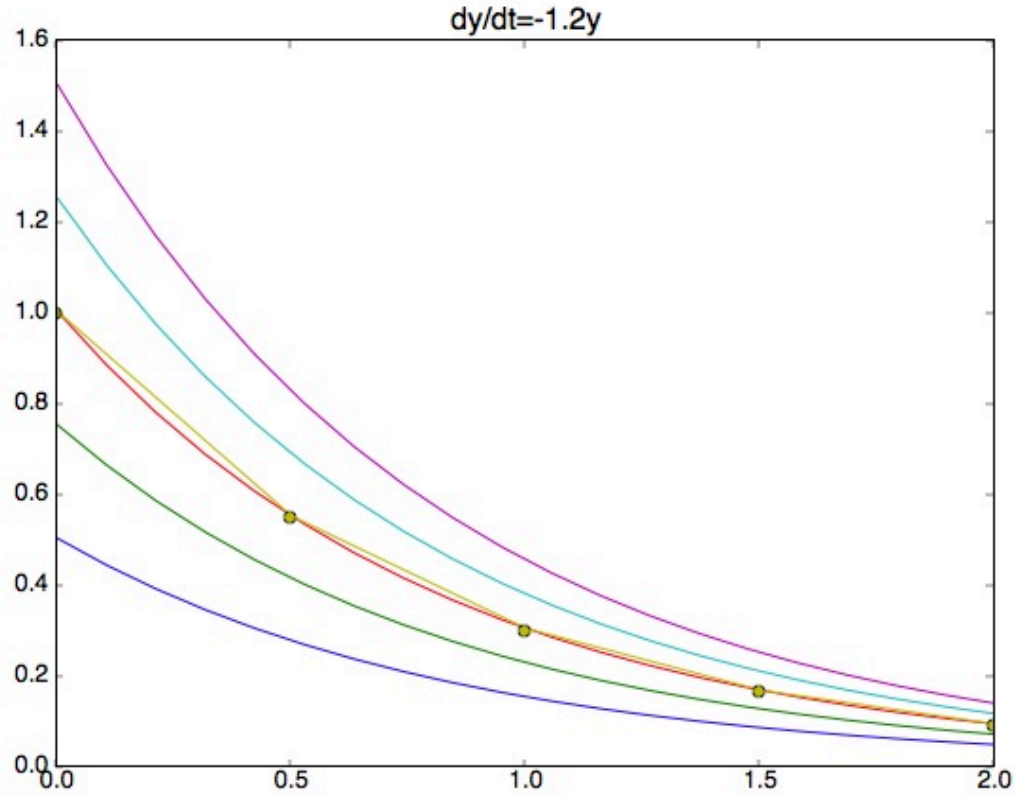
- Some systems are inherently forgiving to integrate
- small deviations don't pull you further away from the solution
- these are "well-behaved functions".

- Notice how initial conditions that were further apart converge in time.

$$\frac{dy}{dt} = -1.2y$$

$$y(0) = 1.0$$

used program: ex_odeint3.py



adapted from Scinet Scientific Computing course

Determining Function Stability

- The solution to the ODE: $$\frac{dy}{dt} = f(y,t)$$

  is STABLE if: $\forall \quad \epsilon > 0 \quad \exists \quad \delta > 0 \quad \text{s.t.}$

  if $\hat{y}$ satisfies the ODE and: $\| \hat{y}(t_0) - y(t_0) \| \leq \delta$

  then: $\| \hat{y}(t) - y(t) \| \leq \epsilon \quad \forall\, t \geq t_0$

- In english: if the initial value is perturbed, then the perturbed solution remains close to the original solution.

Determining Function Stability

- You can determine stability by looking at eigenvalues of ODE.
- Simple example (of the form we considered on previous pages):

$$\frac{dy}{dt} = \lambda y$$

- Solution: $y(t) = y_0 \exp(\lambda t)$
- For $\lambda > 0$ all nonzero solutions grow exponentially and hence diverge. → UNSTABLE
- For $\lambda < 0$ all solutions converge → STABLE.

Function Stability for Linear Systems

- You can determine stability by looking at eigenvalues of ODE.
- More general case: Consider a linear system of ODEs:

$$\frac{d\vec{r}}{dt} = A\vec{r}$$

- where *A* is a matrix of constants (since the system is linear).
- STABLE if ALL eigenvalues of *A* satisfy: $\mathrm{Re}(\lambda_i) \leq 0$
- UNSTABLE if ANY eigenvalue of *A* satisfies: $\mathrm{Re}(\lambda_i) > 0$

Specific Example

$$\frac{dx_1}{dt} = 3.0x_3 + 4.0x_2$$
$$\frac{dx_2}{dt} = x_2 - x_1$$
$$\frac{dx_3}{dt} = -x_3$$

becomes:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 & 4 & 3 \\ -1 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

- Solve for eigenvalues of matrix:

[0.5+1.93649167j 0.5-1.93649167j -1.0+0.j]

Since at least 1 of the eigenvalues has a real part > 0, these equations are unstable.

Function Stability for Nonlinear Systems

- Can't write in matrix format. But you can do the following:
- LINEARIZE the ODE using truncated Taylor series to get the Jacobian matrix of the RHS vector.
- Then determine if the linearized system is LINEARLY stable by looking at the eigenvalues of the Jacobian matrix.

$$\frac{d\vec{r}}{dt} = \vec{f}(\vec{r},t)$$ linearize f to get new system:

$$\frac{d\vec{s}}{dt} = \vec{g}(\vec{r},t)$$ then calculate the Jacobian of g and find its eigenvalues

Specific Example

$$\frac{dx_1}{dt} = x_3{}^2 + \cos(x_2)$$
$$\frac{dx_2}{dt} = x_2 - x_1 x_3$$
$$\frac{dx_3}{dt} = -\exp(x_3)$$

Jacobian matrix: $J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix}$

- Plugging in:

$$J_f = \begin{bmatrix} 0 & -\sin(x_2) & 2x_3 \\ -x_3 & 1 & -x_1 \\ 0 & 0 & -\exp(x_3) \end{bmatrix}$$

- Linearize:

$$J_f = \begin{bmatrix} 0 & -x_2 & 2x_3 \\ -x_3 & 1 & -x_1 \\ 0 & 0 & -1-x_3 \end{bmatrix}$$

Specific Example cont.

- Linearize:

$$J_f = \begin{bmatrix} 0 & -x_2 & 2x_3 \\ -x_3 & 1 & -x_1 \\ 0 & 0 & -1-x_3 \end{bmatrix}$$

- If the Jacobian had only included constants, then you could determine the stability of the ODE for all solutions.

- Because this Jacobian depends on the vector, you can only determine if it is LOCALLY stable, by plugging in values for the vector and checking the eigenvalues of the Jacobian.

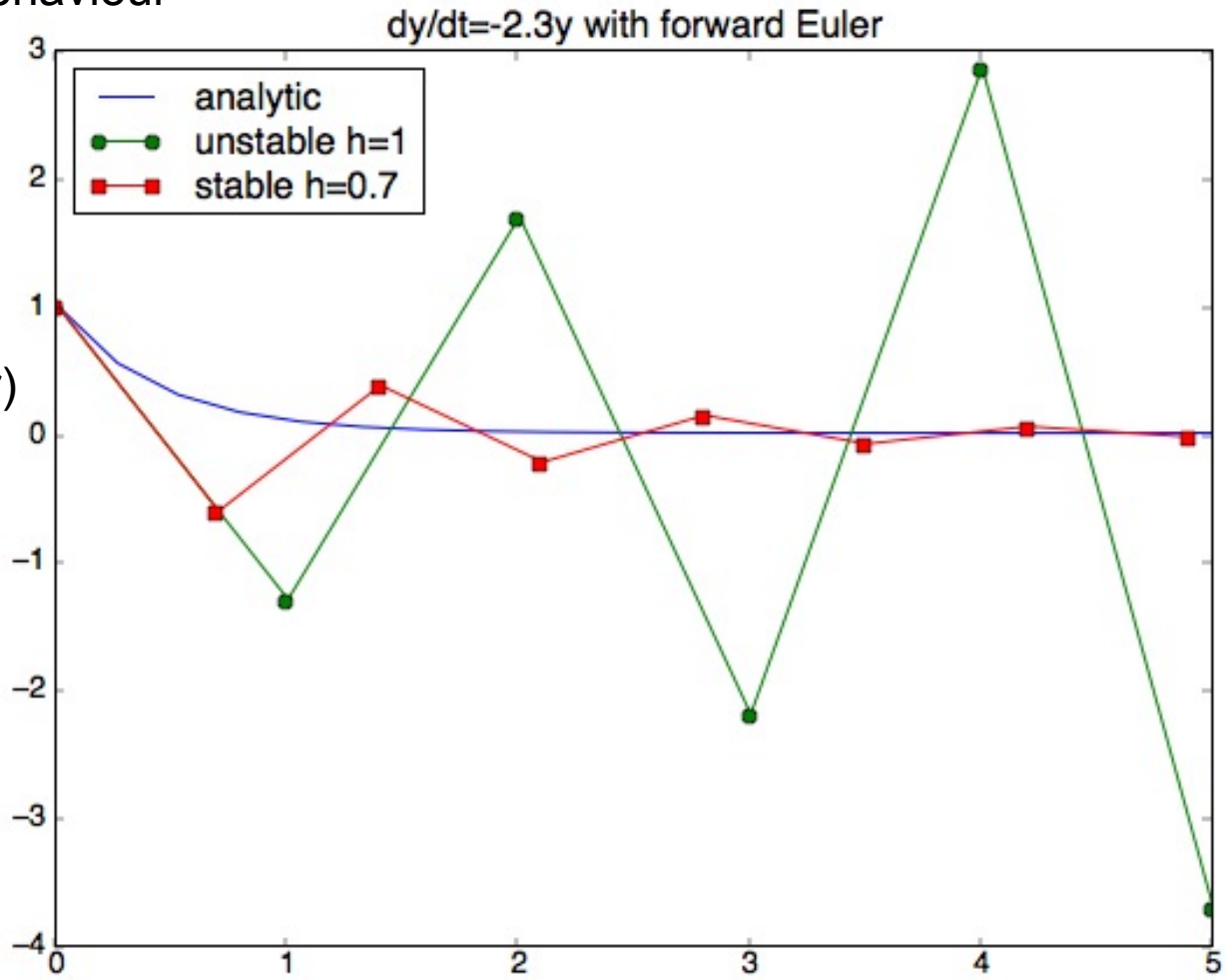
Function Stability

- So what does this all mean?
- If you have stable functions, then you probably don't have to worry so much about the accuracy of the solution.
- Small errors don't tend to build up to produce a really wrong answer.
- If you have unstable functions, then you have to be concerned with the accuracy of your solution at each step. This means you will want to use a very accurate method (i.e. high order method) and smaller time steps in order to reduce the build up of your errors.

Method Stability

- Even with perfectly well-behaved, stable functions, methods can be unstable
- Errors grow without bound

- Often see oscillatory behaviour



- Example: function is very stable

- Method (Forward Euler) very unstable with h =1 but stable at h=0.7

- Need to determine stability of METHODS. Typically this depends on the ODE and the step size

Method Stability

- Methods suffer from 2 sources of error:

  1. Rounding error
  2. Truncation (or Discretization) error

- In most practical situations, truncation error is the dominant factor in determining the accuracy of numerical solutions of ODEs, so we will only consider that for now.

- For truncation error at the "kth" step, need to consider:

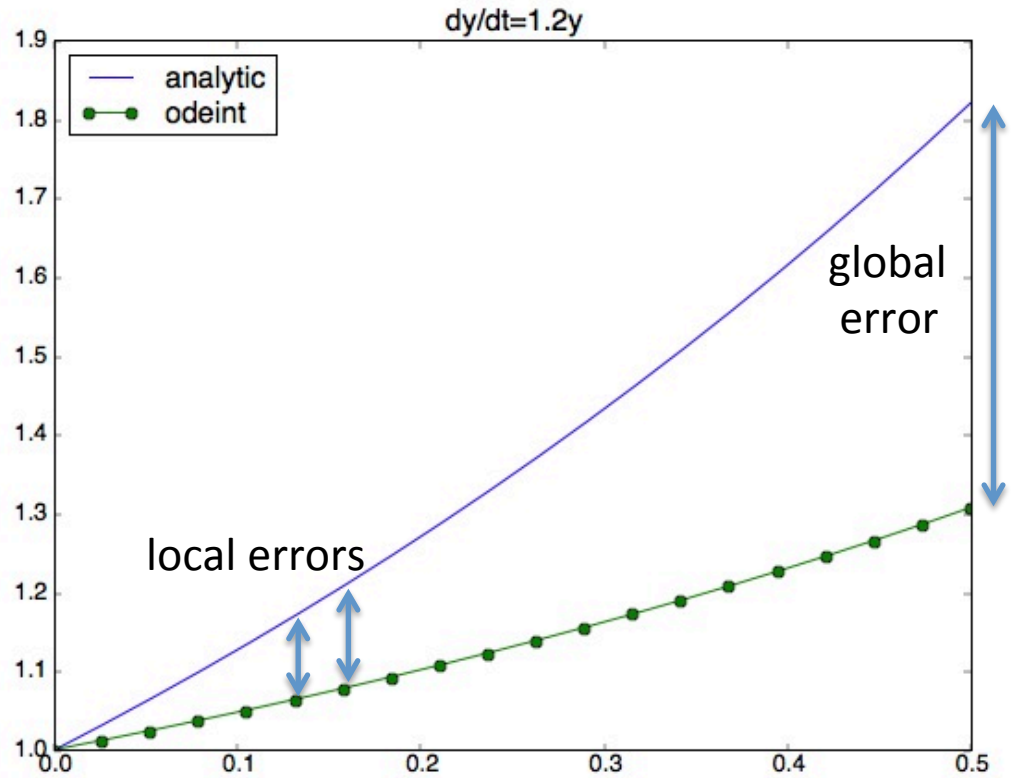
  1. GLOBAL error: the difference $e_k = y_k - y(t_k)$ where y_k is the computed solution at t_k and $y(t)$ is the true solution of the ODE passing through the initial point (t_0, y_0).

  2. LOCAL error: the error made in one step of the numerical method

     $l_k = y_k - u_{k-1}(t_k)$ where u_{k-1} is the sol'n of the ODE passing through the previous point (t_{k-1}, y_{k-1})

Method Stability

- the global error is of primary interest (what we care about is how much of an error we made in the total evolution)
- the local error is the one we can easily estimate and control



Method Stability

- The global error is not simply the sum of the local errors
- if the solutions are diverging, the local errors get magnified over time making a larger global error than just the sum of the local errors
- if the solutions are converging, the global error may be less than the sum of the local errors.
- in order to assess the effectiveness of a numerical method, we need to characterize both its local error (i.e. its accuracy) and the compounding effects over multiple steps (i.e. its stability).
- The ACCURACY of a numerical method is said to be of order *p* if:

$$l_k = O(h_k^{p+1})$$

Method Stability

- The STABILITY of a method is analogous to the stability of functions.
- A numerical method is said to be STABLE if small perturbations do not cause the resulting numerical solution to diverge away without bound.
- Such divergence of numerical solutions could be caused by the instability of the solution to the ODE or by the numerical method itself (even if the solution to the ODE is stable).
- Simple example: Consider Euler's method after a single step of size h_k:

$$y_{k+1} = y_k + h_k f(y_k)$$

- Now consider a specific ODE (i.e. a specific f). Say, our previous example (with lambda = -2.3):

$$\frac{dy}{dt} = \lambda y$$

Method Stability

- For this specific example, after 1 time step, Euler's method gives:

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

- Applying this from k=0 to k gives: $y_k = (1 + h_k \lambda)^k y_0$

- The quantity $(1 + h_k \lambda)$ is called the GROWTH FACTOR.

- In order for Euler's method to be stable, the magnitude of the growth factor must be:

$$|1 + h_k \lambda| \leq 1$$

- So, first notice that only for negative lambda is this even possible. Next, for a given lambda, a minimum step size must be chosen for the method to be stable. Notice for lambda = -2.3, an h=1 is unstable whereas an h = 0.7 is stable

Method Stability

- To determine the accuracy and stability of Euler's method for a general system:

$$\frac{dy}{dt} = f(y,t)$$

- use a Taylor expansion:

$$y(t+h) = y(t) + hy'(t) + O(h^2) = y(t) + hf(y(t),t) + O(h^2)$$

- Take $t=t_k$ and $h=h_k$:

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

- Now subtract this from the expression for y_{k+1} from Euler's method:

$$y_{k+1} = y_k + h_k f(y_k)$$

- to get:

Method Stability

$$y_{k+1} - y(t_{k+1}) = (y_k - y(t_k)) + h_k [f(y_k, t_k) - f(y(t_k), t_k)] + O(h_k^2)$$

- The difference on the LHS is the GLOBAL error e_{k+1}
- If there were no prior errors, then the first 2 terms on the RHS would be 0 and we would conclude that the error is $O(h_k^2)$. This is the LOCAL error.
- Using $l_k = O(h_k^{p+1})$ means that Euler's method is first-order accurate (since p=1).
- The global error at a given step is the sum of the local error at that step and the PROPAGATED error from previous steps. We can characterize this as follows:

Method Stability

- Using the Mean Value Theorem:

$$f(y_k, t_k) - f(y(t_k), t_k) = J_f(t_k, \alpha y_k + (1-\alpha) y(t_k))(y_k - y(t_k))$$

- where J is the Jacobian matrix of f with respect to y and alpha is between 0 and 1.

- This means we can express the global error at step k+1 as:

$$e_{k+1} = (I + h_k J_f) e_k + l_{k+1}$$

- Thus the global error is multiplied at each step by the factor ($I+h_kJ_f$) which is called the GROWTH factor (or AMPLIFICATION factor).

- The errors do not grow if the spectral radius:

$$\rho(I + h_k J_f) \leq 1$$

- which is satisfied if all the eigenvalues of h_kJ_f lie inside a circle in the complex plane of radius 1 centered at -1.

Method Stability

- If this is not the case, then the errors grow and the method is unstable, regardless of whether the solution to the ODE is stable.

- NOTE: in general, the amplification factor depends on the particular ODE being solved (since it determines the Jacobian) and so a particular numerical method's stability depends on the ODE (i.e. a method can be unstable for one ODE and stable for another).

- In choosing a step size for advancing the solution of an ODE, want to minimize computational cost by taking as large a step as possible, but we must take into account both stability and accuracy.

- To yield a meaningful solution, the step size must obey any stability restrictions imposed by the method being used. In addition, a local error estimate is needed to ensure that the desired accuracy is attained.

Method Stability

- For example, with Euler's method, the local error is approximately:

$$\frac{h_k^2}{2} y_k''$$

- If we want the local error to at most equal some tolerance (tol) then the step size should satisfy:

$$h_k \leq \sqrt{2tol / \| y_k'' \|}$$

- We don't know the second derivative, so it can be estimated using:

$$y_k'' \approx \frac{y_k' - y_{k-1}'}{t_k - t_{k-1}}$$

- Other methods of obtaining local error estimates are based on differences between results obtained using methods of different orders of accuracy or different step sizes (this is what we will do in today's lab).