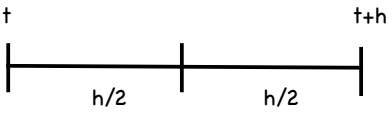
AM111 Lectures | **Sections** | **Office Hours**

- ☐ Week 5
- ☐ Week 6
- ☐ Week 7
- ☐ Week 8
- ☐ Week 9 Spring Break!
- ☐ Week 10
- ☐ Week 11
  - ☐ *Apr. 10th*
  - ☐ *Apr. 11th*
    - ☐ Last time:
      - ☐ How to get better accuracy than the Euler method
      - ☐ Higher-order Taylor methods
      - ☐ Multistage single-step methods
        - ☐ How to derive them
        - ☐ Examples: 2nd order and 4th order Runge-Kutta
    - ☐ Error Control
      - ☐ If we know that the local error is $\propto h^{m+1}$ and the error is e for a particular h, then given a global error tolerance ε, we can change the step-size from $h \to qh$ s.t. the new error $|q^m e| \simeq \varepsilon$
      - ☐ This implies $q \simeq \left|\frac{1}{2e}\right|^{1/m}$
      - ☐ One approach: Step doubling
        - ☐ over h: $e(h) = kh^{m+1}$



        - ☐ over h: $e(h/2) = 2k\left(\frac{1}{2}\right)^{m+1} h^{m+1}$
        - ☐ $y_{n+1}(h) - y_{n+1}(h/2) = \left[1 - \left(\frac{1}{2}\right)^m\right] kh^{m+1} = [2^m - 1]\, e(h/2)$
        - ☐ For a 4th order method:
          - ☐ $y_{n+1}(h) - y_{n+1}(h/2) = 15e(h/2)$
          - ☐ We estimate the error made as: $e(h/2) = \frac{y_{n+1}(h) - y_{n+1}(h/2)}{2^m - 1}$
        - ☐ This works nicely enough, and is easy to write, but has large overhead.
      - ☐ A more efficient (modern) way: Embedded Runge-Kutta method
        - ☐ Use two methods of different order (remember ode45!)
          - ☐ (1) $\tilde{e} = y(t_{n+1}) - \tilde{y}_{n+1}$ this is $O(h^{m+2})$
          - ☐ (2) $e = y(t_{n+1}) - y_{n+1}$ is $O(h^{m+1})$
          - ☐ $e = \tilde{e} + \tilde{y}_{n+1} - y_{n+1} \simeq \tilde{y}_{n+1} - y_{n+1}$
          - ☐ The way we do this efficiently is to use the slopes needed to calculate the lower-order method to help calculate the higher-order method.
          - ☐ See section 7.5 in Moler's NCM. for an example of a 2nd order method with a 3rd order error estimate.
        - ☐ Implementation:
          - ☐ Calculate y_{n+1} using the lower order method, and $\tilde{y}_{n+1}$ using the higher order method. Then:
          - ☐ $q = \left|\frac{1}{2(\tilde{y}_{n+1} - y_{n+1})}\right|^{1/m}$
          - ☐ $h = qh$
          - ☐ if q<1, repeat with the new h

- else (q >=1), continue with new h
- Examples:
  - F=@(t,y) 0; ode23(F,[0 10],1)
  - F=@(t,y) t; ode23(F,[0 10],1)
  - F=@(t,y) y; ode23(F,[0 10],1)
  - F=@(t,y) -y; ode23(F,[0 10],1)
  - F=@(t,y) sin(t); ode23(F,[0 10],1)
    - Cruddy result --Boost tolerance!
    - opts = odeset('RelTol',1e-6);
    - F=@(t,y) sin(t); ode23(F,[0 10],1,opts)
    - Much better!
  - Pendulum Equation: $\ddot{\theta} = -\frac{g}{L}\sin\theta$
    - Watch the tolerance!
  - Lorenz Equation:
    - $\dot{y}_1 = -\beta y_1 + y_2 y_3$
      $\dot{y}_2 = -\sigma y_2 + \sigma y_3$
      $\dot{y}_3 = -y_2 y_1 + \rho y_2 - y_3$
- Multistep Methods
  - Idea is to make use of previously computed $y_k,\ k \le n$, to compute y_{n+1}
  -
  - $\frac{dy}{dt} = f(t,y)$
  - $y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t,y)dt$
  - Idea: $y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} P(t,y)dt$ We will choose different P's to approximate f.
    - First order scheme: $P_1(t)$ = constant
      - $y_{n+1} = y_n + \Delta t f(t_n, y_n)$ Euler method!
    - 2nd order method:
      - $P_2(t) = f_{n-1} + \frac{f_n - f_{n-1}}{\Delta t}(t - t_n)$
      - Inserted into the integral equation, this yields:
      - $y_{n+1} = y_n + \frac{\Delta t}{2}[3f(t_n, y_n) - f(t_{n-1}, y_{n-1})]$
      - This is the 2nd order Adams-Bashforth method.
      - In general, methods which don't involve y_{n+1} (i.e. explicit methods) are called Adams-Bashforth methods.
    - When y_{n+1} is used to determine P, it's called an Adams-Moulton method
    - 1st order A-M:
      - $P_1(t) = const = f(t_{n+1}, y_{n+1})$
      - $y_{n+1} = y_n + \Delta t f(t_{n+1}, y_{n+1})$
      - This is the backward Euler Method!
    - 2nd order A-M:
      - $P_2(t) = f_n + \frac{f_{n+1} - f_n}{\Delta t}(t - t_n)$
        (note similarity to the 2nd order A-B method.)
      - $y_{n+1} = y_n + \frac{\Delta t}{2}[f(t_{n+1}, y_{n+1}) + f(t_n, y_n)]$
      - This is the Trapezoidal Method.
    - Predictor-Corrector Method
      - 2nd order example

AM111 Lectures **Sections** **Office Hours**

- Predictor (A-B): $y_{n+1}^P = y_n + \frac{\Delta t}{2}[3f_n - f_{n-1}]$
- Corrector (A-M): $y_{n+1} = y_n + \frac{\Delta t}{2}\left[f(t_{n+1}, y_{n+1}^P) + f(t_n, y_n)\right]$
- Used to model the orbital mechanics of spacecraft.

- Can we achieve arbitrary accuracy?
  - Of course not.
  - Suppose we intergrate over an interval of length $L = t_f - t_0$
  - The # of steps we take to do this integral is N=L/h.
  - The roundoff error for each step is ε.
  - So the total roundoff error is $< N\varepsilon$ (more realistically, this scales as $\sqrt{N}\varepsilon$)
  - The total error is thus: $Ch^m + \frac{L}{h}\varepsilon$ at mth order. If we make h too small, then this blows up.
- Stiffness of Problems:
  - What are some examples of stiff ODEs?
    - $\frac{dy}{dt} = \Lambda y$
    - $y(t=0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
    - $\Lambda = \begin{bmatrix} -100 & 1 \\ 0 & \frac{-1}{10} \end{bmatrix}$
    - Using the forward Euler method yields a solution which blows up at large time!
    - If we try the backward Euler method, we find that everything's just spiffy. More next time.
  -

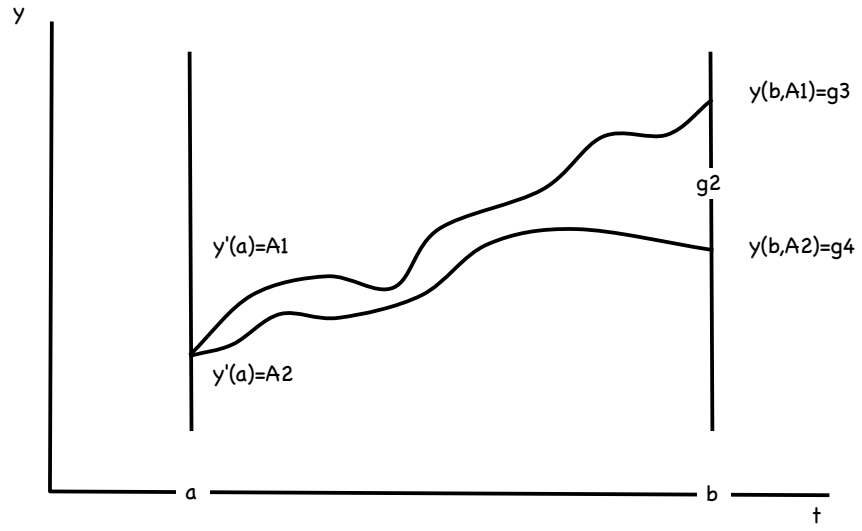
- *Apr. 13th*
  - Round-off Errors
    - Over the interval $[t_0, t_f]$ of length $L = t_f - t_0$, the discretization error is Ch^p, and the roundoff error is $\frac{L}{h}\varepsilon$.
    - For various orders, (if L= 20, C = 100, $\varepsilon = 2^{-52}$)
      - p=1: roundoff error becomes important at $N = 4.5 \times 10^{17}$
      - p=3: " N=5,647,721
      - p=5: " N=37,285
      - p = 10: " N=864
  - Stiff problems
    - example: $\frac{dy}{dt} = \Lambda y$
    - $y(0) = y_0$
    - $\Lambda = \begin{bmatrix} -100 & 0 \\ 0 & \frac{-1}{10} \end{bmatrix}$. This problem is unstable with the forward Euler method.
      - has two eigenvalues
        - $\lambda_1 = -100$
        - $\lambda_2 = -1/10$
      - The forward Euler method is only stable inside the unit circle centered on z=-1 on the complex plane.
      - So if we look at the stability condition:
        - $|1 + \lambda\Delta t| < 1$

- $= |1+\lambda_1 \Delta t| = 9 < 1$, if our timestep is too large. (as it was when we tried this before)
- If we make a smaller timestep, so that
- $|1+\lambda_1 \Delta t| = 1$ (when $\Delta t = 0.02$), we find that the solution using the forward Euler method doesn't blow up!
- Even if the timestep is slightly too large ($\Delta t = 0.021$) the problem will eventually blow up. (although it does so more slowly than before)

- **Definition:** A problem is "stiff" if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so that the numerical methods must take small steps obtain satisfactory results.
  - **Alternative Definition:** A problem is stiff if its numerical solution by some methods requires (perhaps in only a portion of the solution interval) a significant depression of the step-size to avoid instability.
- The **stiffness ratio** is the ratio of the largest and smallest (in modulus) eigenvalues of a linear system (for a general problem, these are the eigenvalues of the Jacobian matrix)
  - The stiffness ratio of our example problem was 1000.
  - For comparision, consider the world record stiffness of 10^{31} (from cosmological Big-Bang simulation)
- Looking back at section 7.12 in the NCM, we now understand ode45, ode23, and ode113. What about ode15s? It uses backward differentiation formulas (BDFs)
  - BDF
    - $\frac{dy}{dt} = f(t,y)$
    - $y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t,y)dt \simeq y_n + \int_{t_n}^{t_{n+1}} P(t,y)dt$
- ode23s is based on a modified Rosenbrock formula of order 2
  - Rosenbrock formula:
    - This is a generalized implicit Runge-Kutta formula
    - $s_i = f(t, \sum_{k=1}^{i-1} \beta_{i,k} s_k)$ is an explicit method. If we let the sum go over all the slopes, then
    - $s_i = f(t, \sum_{k=1}^{i} \beta_{i,k} s_k)$ we have an implicit method.
- Consider the problem of the size of an expanding flame. The flame depends upon oxygen to exist, so it can grow at a rate proportional to the rate at which oxygen is available to it. If the ball is of radius y, then the rate of injection of oxygen is proportional to y^2. Oxygen consumption should go as the volume of the flame, or y^3. Thus
- $\dot{y} = y^2 - y^3$
- let $y(0) = \delta$, and then run the simulation from $0 \le t \le 2/\delta$
- We notice that this problem is solved much faster with implicit methods than by explicit methods. (recall section, and NCM)
- Why is this so? To find out, let's linearize the equations (find the Jacobian)
  - $J = f_y = 2y - 3y^2$ Near equilibrium, y=1.
    - $\dot{y} = J|(y - y_c) = -1(y-1)$
  - $\lambda = -1$, so the timestep must be $\Delta t \le 2$, which seems big, but the span we're trying to cover is huge, so this severely constrains the span we can solve this over.
  - Implicit methods, by contrast, allow us to take arbitrarily huge timesteps once we're near equilibrium. They're much better for problems like these.
- Summary:
  - Single-step (multistage) methods
  - Multistep methods
    - useful for smooth problems that require high accuracy and where evaluations of f (t,y) are expensive
    - Explicit methods are easier to implement

- ☐ Implicit methods have larger regions of stability
- ☐ Boundary Value Problems:
  - ☐ A 2nd order BVP is (in general)
    - ☐ $\frac{d^2y}{dt^2} = f(t, y, \frac{dy}{dt})$
    - ☐ on $t \in [a,b]$ with the general boundary conditions
    - ☐ $\alpha_1 y(a) + \beta_1 \frac{dy(a)}{dt} = \gamma_1$

      $\alpha_2 y(b) + \beta_2 \frac{dy(b)}{dt} = \gamma_2$
  - ☐ We don't have enough information to set up an initial condition to run forward from, so what do we do?
  - ☐ Shooting Methods:
    - ☐ guess that $y(a) = \gamma_1, y'(a) = A_1$. We then run the initial condition problem forward and find out that this leads to $y(b, A_1) = \gamma_3$. This is a tad off (perhaps bigger than γ_2), so we try another guess, $y(a) = \gamma_1, y'(a) = A_2$. This gives us another result $y(b, A_2) = \gamma_4$, which might be smaller than γ_2. The general problem is to find an argument A s.t.
    - ☐



    - ☐ $y_b(A) - \gamma_2 = 0$
      - ☐ This is a zero-finding problem!
  - ☐ Finite Difference Methods
    - ☐ For example, the Schrödinger equation
    - ☐ $\frac{d^2y}{dt^2} + [V(t) - E]y = 0$
    - ☐ $y(0) = 0$
    - ☐ $y(1) = 0$
    - ☐ $\frac{d^2y}{dt^2} = \frac{y(t+\Delta t) - 2y(t) + y(t-\Delta t)}{\Delta t^2}$
    - ☐ so that
    - ☐ $\frac{y_{n+1} - 2y_n + y_{n-1}}{\Delta t^2} + (V_0 - E)y_n = 0$, with $y_1 = y_N = 0$.
    - ☐ This can be written in matrix form as

AM111 Lectures **Sections** **Office Hours**

- $\begin{bmatrix} -2+V_0\Delta t^2 & 1 & 0 & 0 \dots & 0 \\ 1 & -2+V_0\Delta t^2 & 1 & 0 \dots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & 0 & 1 \quad -2+V_0\Delta t^2 \end{bmatrix} \begin{bmatrix} y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_{N-1} \end{bmatrix} = E \begin{bmatrix} y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_{N-1} \end{bmatrix}$

- Let's try an example
  - $\frac{d^2y}{dt^2} = y^2 - 1$
  - $y(0) = 0, y(1) = 1$
  - Using a shooting method
    - $y(0) = 0, y'(0) = 1 \leftarrow$ a guess
      - This leads to $y(t=1) - 1 \simeq .45$
    - Trying $y(0) = 0, y'(0) = 2$
      - This leads to $y(t=1) - 1 \simeq$ something less than 0·
    - Thus we've bracketed the zero, and we can just solve this problem using whatever our favorite root finding method is.
  - We can also try a finite difference method:
    - This forms a linear system $A(y)y = b$ (the coefficient matrix A depends upon y)
    - This requires some iteration to solve.
    - $\frac{y_{n+1} - 2y_n + y_{n-1}}{\Delta t^2} = y_n^2 - 1$, which, in matrix form is
    - $\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \dots \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \dots & 0 & 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{N-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ A y + b
    - We have $Ay + b = \Delta t^2 c(y)$
    - This can be solved by linear iteration.
  - Newton's method:
    - Residue: $Ay + b - (\Delta t)^2(y^2 - 1)$
    - Jacobian: $A - 2(\Delta t)^2 y$
    - $J\Delta y$=-Residue
- Alternatively, we could just use the MATLAB function **bvp4c**
  - To use, we first:
    - (a) write the ODE in standard form
    - (b) write a function to describe your boundary condition
      - $y_2 = \dot{y}_1$
      - $\dot{y}_1 = y_2, \dot{y}_2 = y_1^2 - 1$
      - $y_1(0) = 0$
      - $y_1(1) - 1 = 0$
- *Apr. 14th*
- Week 12
- Week 13
- Week 14