Computational Astrophysics

ASTR 660, Spring 2021
計算天文物理

Lecture 7

Initial Value Problems

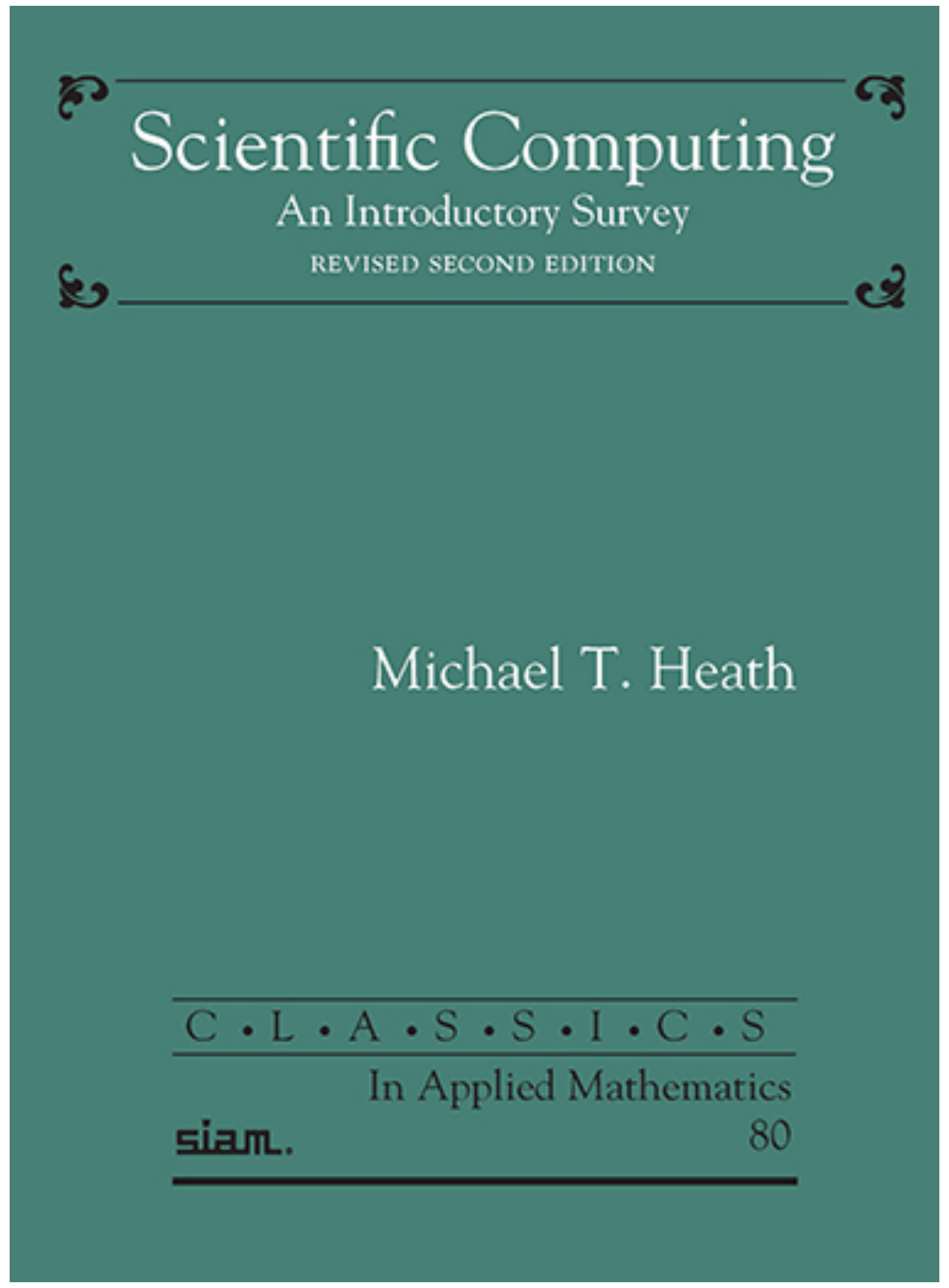
Instructor: Prof. Kuo-Chuan Pan
kuochuan.pan@gapp.nthu.edu.tw

Class website

https://kuochuanpan.github.io/courses/109ASTR660_CA/

Plan for today

- Ordinary Differential Equations (ODEs)
- ODE: Initial Value Problems
- Direct N-body method
- Lab: Solar system simulation



Reference:
"Scientific Computing: An introductory survey", Michael Heath

https://books.google.com.tw/books?id=f6Z8DwAAQBAJ&hl=zh-TW

Ordinary Differential Equation (ODE)

Ordinary Differential Equations

- Ordinary differential equation (ODE): all derivatives are with respect to single independent variable, often representing time
- Order determined by highest-order derivative of solution function appearing in ODE
- Higher-order ODE can be transformed into several equivalent first-order system
- Most ODE software is designed to solve only first-order equations

Example: higher-order ODE

$$y''' = f(t) \longrightarrow \begin{bmatrix} y' = y_1 \\ y_1' = y_2 \\ y_2' = f(t) \end{bmatrix}$$

$$F = ma = mx'' \longrightarrow \begin{bmatrix} x' = v \\ v' = a = F/m \end{bmatrix}$$

Recall the Angry bird and binary problems in lecture 02

Ordinary Differential Equations

- General first-order system of ODEs has form

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

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

- Function f is given and we wish to determine y

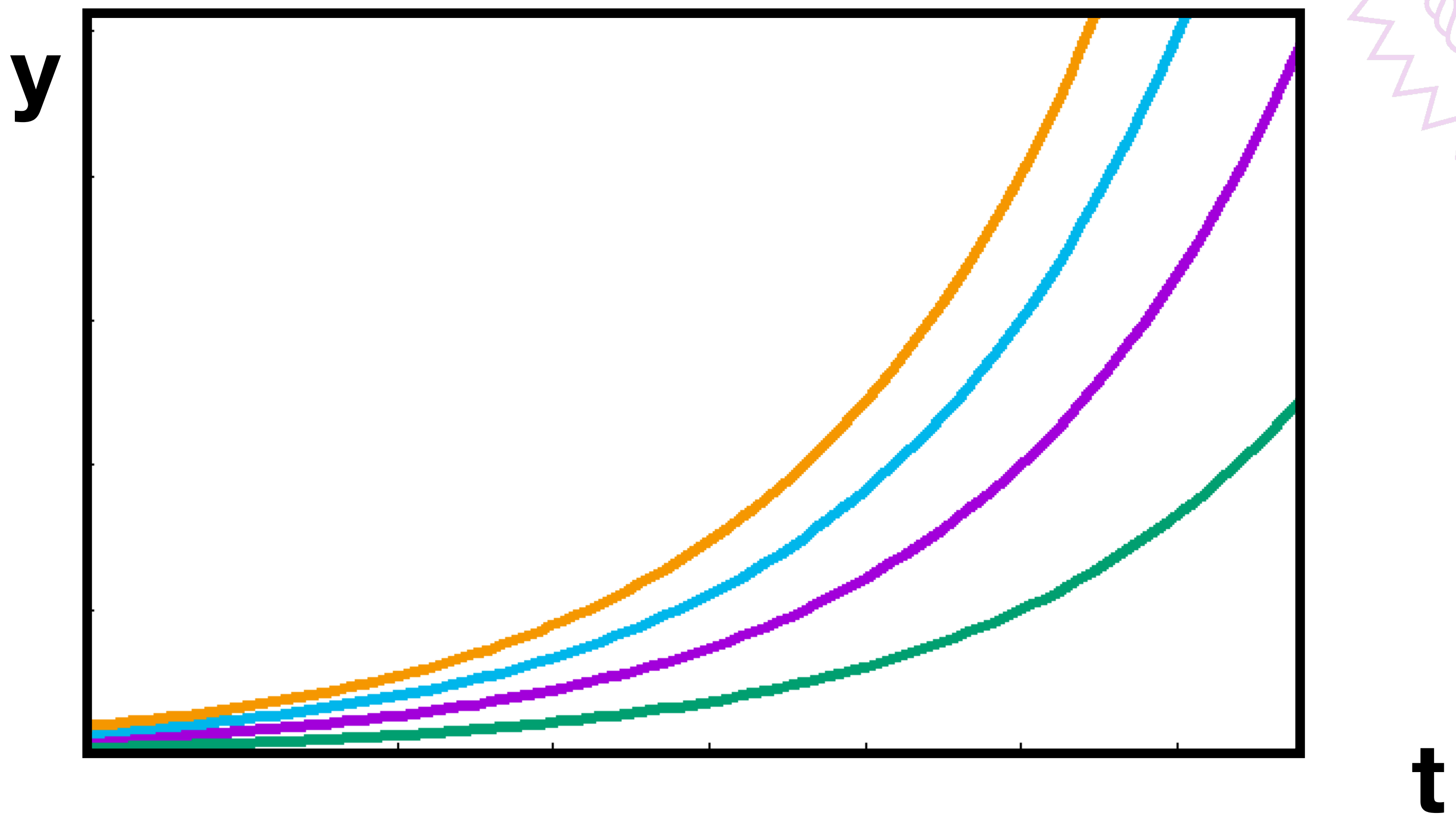
Ordinary Differential Equations

- By itself, ODE does not determine unique solution function
- This is because ODE merely specifies slope of solution function at each point, but not actual value of y at any point
- Therefore, requires an initial value to solve the specific solution function
- That is why we called “Initial Value Problems (IVP)”

Example: Ordinary Differential Equations

- Consider scalar (n=1) ODE

$$y' = y$$



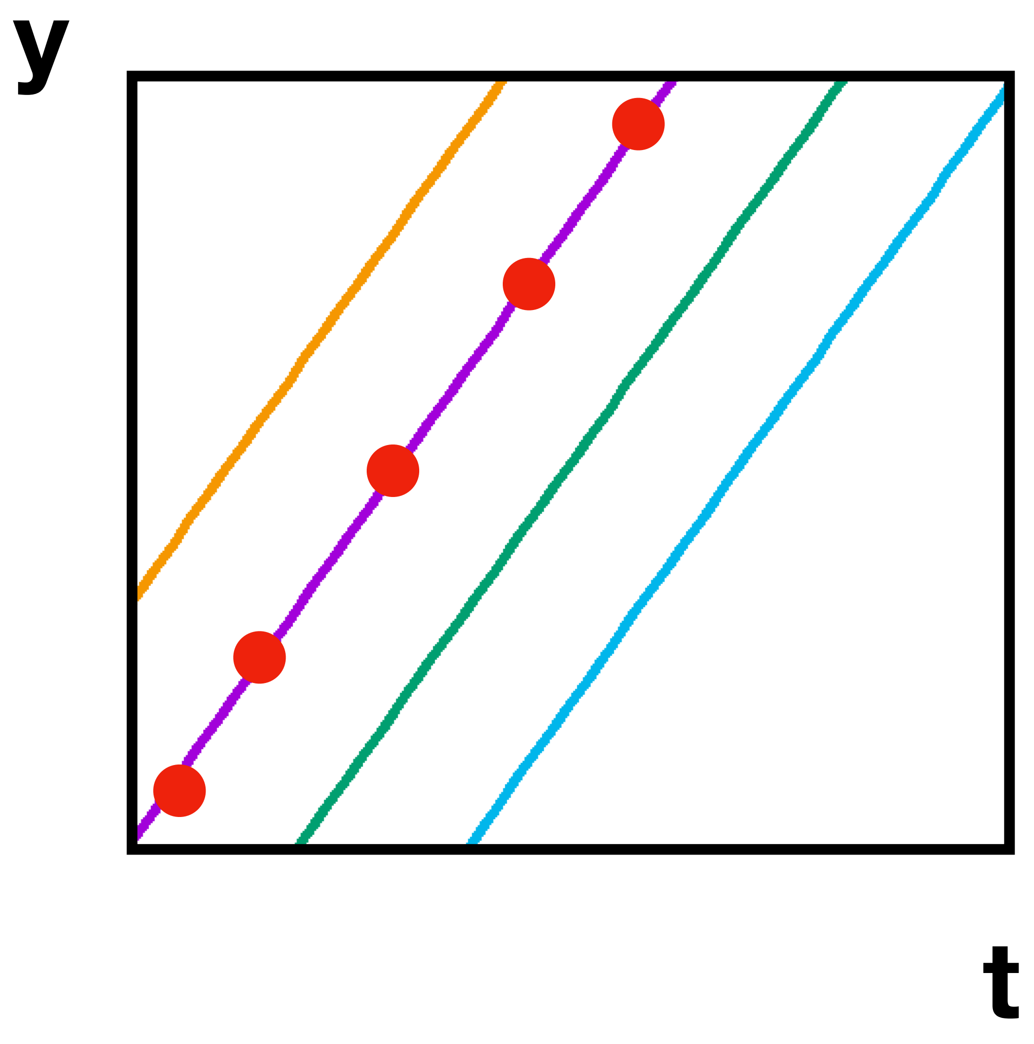
- Family of solutions is given by y=c exp(t), where c is an arbitrary real constant
- In this example, if $t_0=0$ $y=y_0$, then c $=y_0$, which means that solution is $y(t) = y_0 \exp(t)$

Stability of solutions

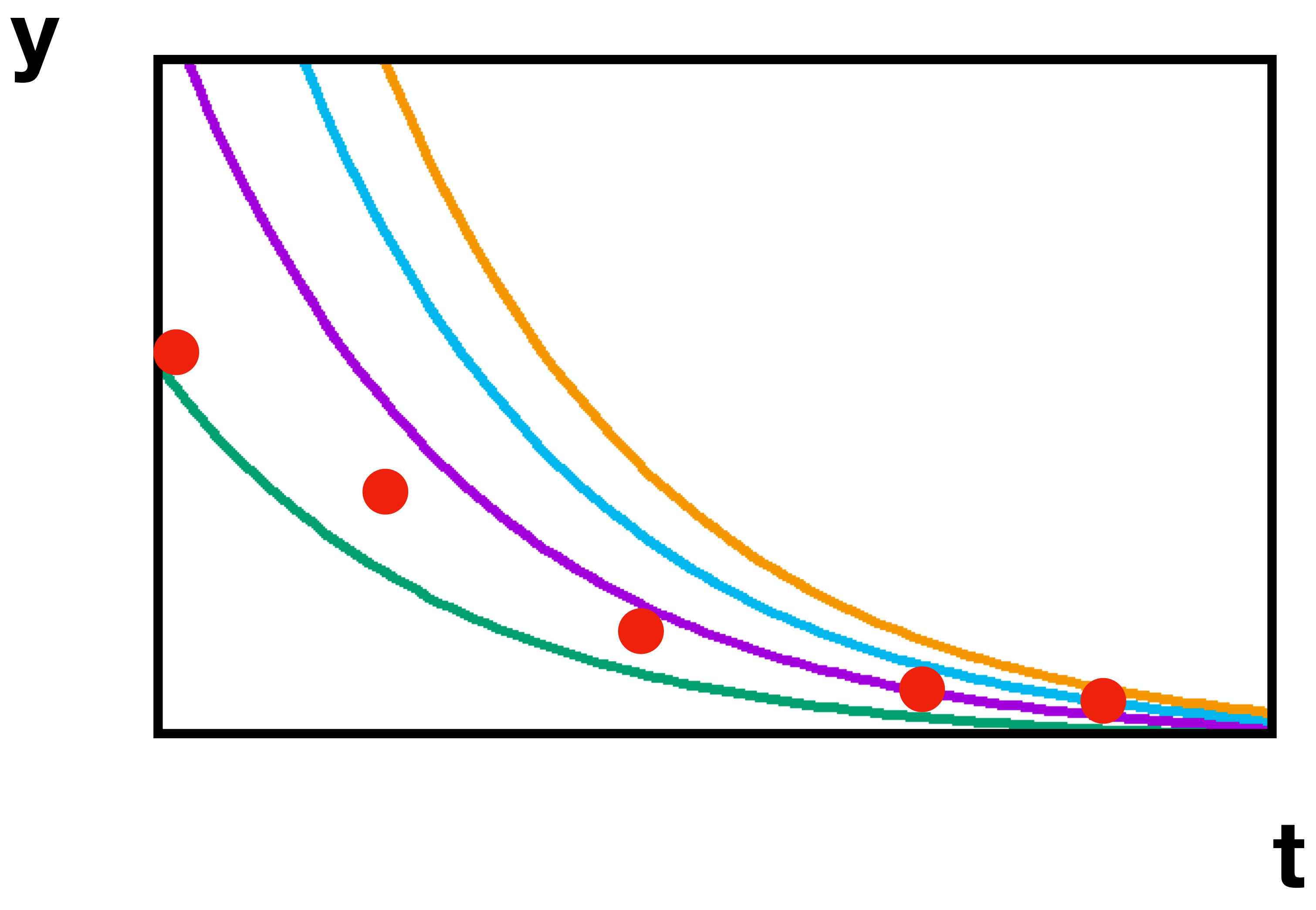
- Stable: if solutions resulting from perturbations of initial value remain close to original solution
- Asymptotically stable: if solutions resulting from perturbations converge back to original solution
- Unstable: if solutions resulting from perturbations diverge away from original solution without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

Stability of solutions

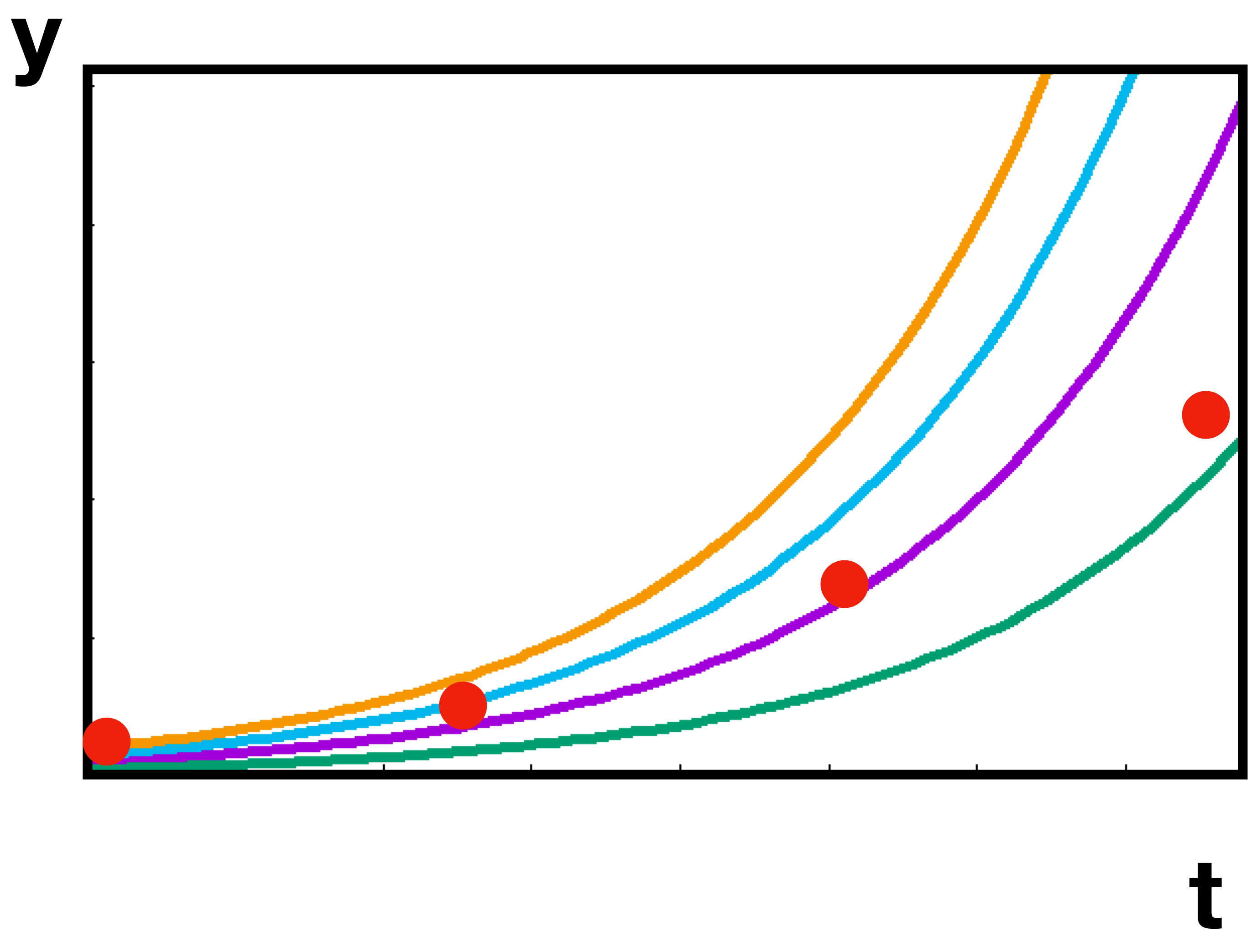
Stable



Asymptotically Stable



Unstable



Stiff ODEs

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

Errors in Numerical Solution of ODEs

Recall lecture 01

- Truncation error: due to mathematical approximations
- Rounding error: due to inexact representation of real numbers and arithmetic operations upon them

In practice, truncation error is the dominant factor

Errors in Numerical Solution of ODEs

- Global error: difference between computed solution and true solution
- Local error: error made in one step of numerical method
- Global error is not necessary sum of local errors

Numerical Solution of ODEs $\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y})$

- Consider Taylor series:

$$y(t+h) = y(t) + y'(t)h + \frac{y''(t)}{2}h^2 + \frac{y'''(t)}{6}h^3 + \ldots$$

- Euler's method: consider only first order term
- Advances solution by extrapolating along straight line whose slop is give by f(t,y)
- Euler's method is single-step method

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

Explicit and implicit methods

- (forward) Euler's method is explicit. It uses only information at time t_k to advance solution to time t_{k+1}
- Larger stability region can be obtained by using information at time t_{k+1}, which makes method implicit.
- Backward Euler method is implicit

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

Implicit methods

- Backward Euler method is implicit

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

- Typically, we use iterative method such as Newton's method to solve for y_{k+1}
- Good starting guess for iteration can be obtained from explicit method or from solution at previous time step

Example Implicit methods

- Consider ODE:

$$y' = -y^3 \text{ with initial condition } y(0) = 1$$

- Using backward Euler with step size h=0.5, we obtain implicit equation

$$y_1 = y_0 + hf(t_1, y_1) = 1 - 0.5y_1^3$$

- Can be solved by Newton's method
- Starting guess of y_1 can be obtained by explicit method, such as Euler, which gives $y_1 = y_0 - 0.5y_0^3 = 0.5$

Implicit methods

- Takes extra efforts (more expensive)
- But implicit methods generally have significantly larger stability region than comparable explicit methods

Example: Stability

- Consider ODE: $y' = \lambda y$

Forward Euler

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

$$y_k = \underbrace{(1+h\lambda)}_{\text{Growth factor}}{}^k y_0$$

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

Backward Euler

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

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

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

Hold for any h when Re(lambda) <0

Higher-order methods

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

$$y_{k+1} = y_k + h_k(f(t_k, y_k) + f(t_{k+1}, y_{k+1}))/2$$

Numerical Methods for ODEs

- Single-step methods (Taylor series, Runge-Kutta, Extrapolation)
- Multistep methods
- Multivalue methods

Taylor Series Methods

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \ldots$$

- Euler's method can be derived from Taylor series expansion
- Higher-order can be achieved by retaining more terms in Taylor series
- For example

$$y_{k+1} = y_k + h_k y'_k + \frac{h_k^2}{2} y''_k$$

Difficult

Runge-Kutta Methods

- Runge-Kutta methods are single-step methods similar in motivation to Taylor series methods, but do not require computation of higher derivatives
- Instead, Runge-Kutta methods simulate effect of higher derivatives by evaluating f several times between t_k and t_{k+1}

Heun's Method (RK2)

- Simplest example is second-order Heun's method (or Runge-Kutta 2)

$$y_{k+1} = y_k + \frac{h_k}{2}(k_1 + k_2)$$

$$\begin{aligned} k_1 &= f(t_k, y_k) \\ k_2 &= f(t_k + h_k, y_k + h_k k_1) \end{aligned}$$

- Similar to implicit trapezoid method, but remains explicit

Forth-order Runge-Kutta Method (RK4)

- Best-known Runge-Kutta method is the classical RK4

$$y_{k+1} = y_k + \frac{h_k}{6}(k_1 + 2k_2 + 2k_3 + k4)$$

$$\begin{aligned} k_1 &= f(t_k, y_k) \\ k_2 &= f(t_k + h_k/2, y_k + (h_k/2)k_1) \\ k_3 &= f(t_k + h_k/2, y_k + (h_k/2)k_2) \\ k_4 &= f(t_k + h_k, y_k + h_k k_3) \end{aligned}$$

- Analogous to Simpson's rule

Forth-order Runge-Kutta Method (RK4)

Pros

- No history of solution prior to time t_k (self-starting)
- Easy to change step size
- Easy to program

Cons

- No error estimate
- Inefficient for stiff ODEs

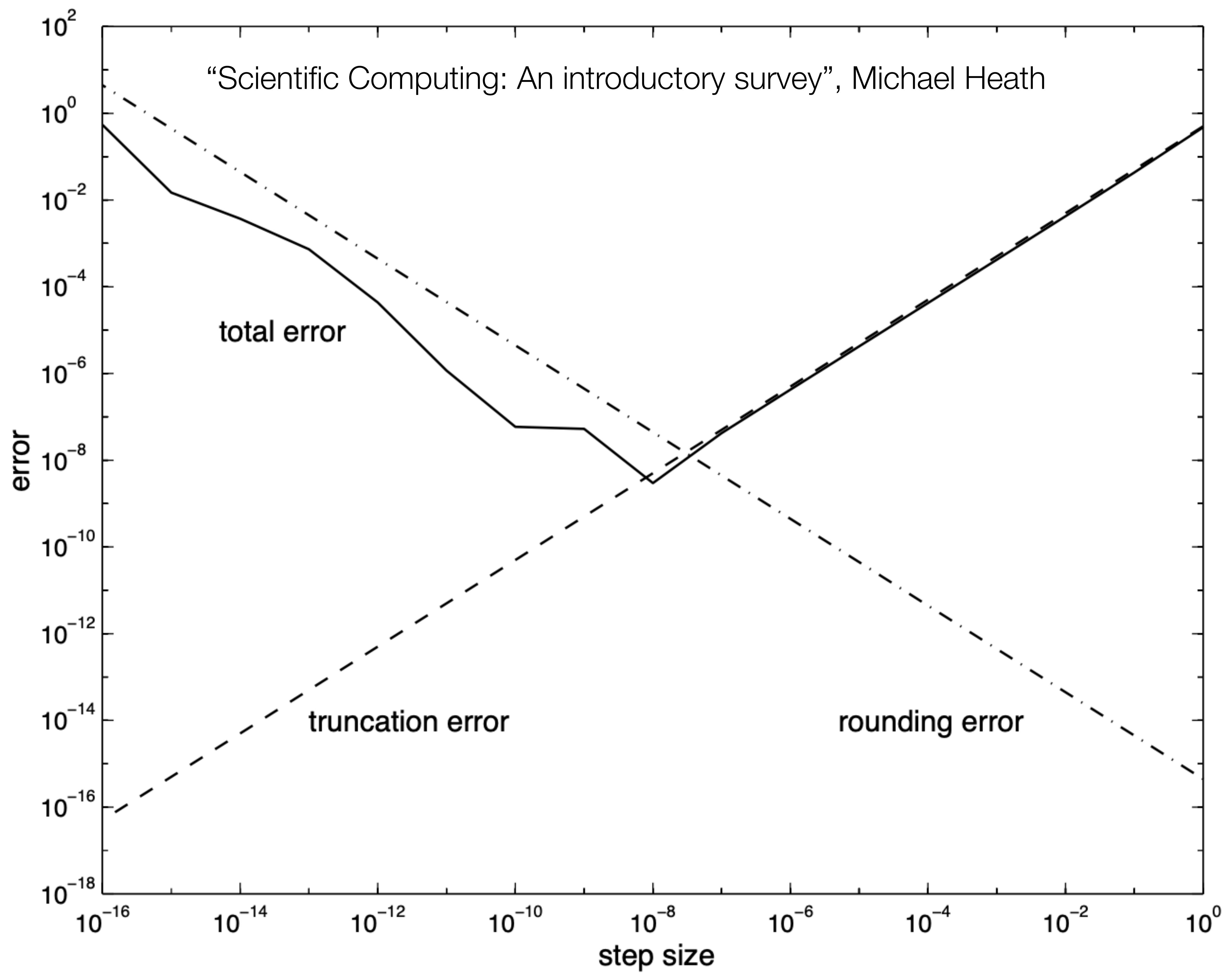
Exercise: Runge Kutta Methods

Modify your angry bird simulation code

Step 1. Refresh the Euler's method
Step 2. Implement the RK 2 method
Step 3. Implement the RK 4 method

Truncation error and rounding error

Recall Lecture 01



Extrapolation Method

- Use single-step method to integrate ODE over given interval $t_k <= t <= t_{k+1}$ using different step size h_i, and yielding results dented by $Y(h_i)$
- Gives discrete approximation to function $Y(h)$, where $Y(0) = y(t_{k+1})$
- Extrapolation methods are capable of achieving very high accuracy but less efficient and less flexible than other methods for ODEs

Multistep methods

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

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

- Alpha and beta are determined by polynomial interpolation. If beta_0 =0, method is explicit, other wise it is implicit

Multistep methods

- Simplest second-order explicit two-step method:

$$y_{k+1} = y_k + h_k \underline{(3y'_k - y'_{k-1})}/2$$

require two starting values

- Simplest second-order implicit method is trapezoid method

$$y_{k+1} = y_k + h_k(y'_{k+1} + y'_k)/2$$

Predictor-Corrector Method

- Implicit methods are usually more accurate and stable than explicit methods, but require starting guess for y_{k+1}
- Starting guess is conveniently supplied by explicit method, so the two are used as predictor-corrector pair
- One could use corrector repeatedly until some convergence tolerance is met (expensive)
- In practice, only use fixed number of corrector steps

Example: Predictor-Corrector Method

$y' = -2ty^2$ With initial value y(0) = 1

(1) Pick a h=0.25, use RK2 to obtain y_1= 0.9375 at t_1=0.25

(2) Use $y_{k+1} = y_k + h_k(3y'_k - y'_{k-1})/2$ Two-step explicit method:

(3) $\hat{y}_2 = y_1 + \frac{h}{2}(3y'_1 - y'_0) = 0.7727$ **The predicted value**

(4) $\hat{y}'_2 = -0.05971$

(5) $y_2 = y_1 + \frac{h}{2}(y'_2 + y'_1)$ **The corrected solution!**

Implicit trapezoid method

Multistep methods

Pros

- Good local error estimate can be determined from difference between predictor and corrector
- Being based on interpolation, they can provide solution values at output points other than integration points
- Can be effective for stiff ODEs

Cons

- Not self-starting, since several previous values are needed initially
- Changing step size is complicated
- Relatively complicated to program

Multi-value methods

- Like multistep methods, multivalue methods are also based on polynomial interpolation, but avoid some implementation difficulties associated with multistep methods
- One key idea motivating multivalue method is observation that interpolating polynomial itself can be evaluated at any point, not just at equally spaced intervals

Example: Multi-value methods

- Consider four-values method for solving scalar ODE

$$y' = f(t, y)$$

- Instead of representing interpolating polynomial by its value at four different points, we represent it by its value and first three derivatives at single point t_k

$$\mathbf{y}_k = \begin{bmatrix} y_k \\ h y'_k \\ (h^2/2) y''_k \\ (h^3/6) y'''_k \end{bmatrix}$$

Example: Multi-value methods

- By differentiating Taylor series

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \ldots$$

- Corresponding values at next point t_{k+1} = t_k+h are given approximately by transformation

$$\hat{y}_{k+1} = By_k$$

corrector

$$y_{k+1} = \hat{y}_{k+1} + \alpha r$$

r is a fixed vector

$$\alpha = h(y'_{k+1} - \hat{y}'_{k+1})$$

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If r = [3/8, 1, 3/4, 1/6].T -> implicit fourth-order Adams-Moulton method

Summary: IVP

- Numerical solution of ODE IVP is table of approximate values of solution function at discrete points, generated by simulating behavior of system governed by ODE step by step
- Accuracy can be improved by using higher-order methods, and stability region can be expanded by using implicit methods
- Implicit methods are especially important for solving stiff ODEs, which have widely disparate time scales
- Import families of ODE methods include Runge-Kutta and multistep/multivalue methods

Direct N-body Method

N- Body Problem

- The classical astrophysical “N-Body” problem:

$$\frac{d^2 x_i}{dt^2} = - \sum_{j=1;j\neq i}^{N} \frac{G m_j (x_i - x_j)}{|x_i - x_j|^3}$$

1. Calculating the net force on a given particle
2. Determining the new position of the particle at an advanced time

N- Body Problem

- If we write:

$$w_i = [x_i, v_i] = (w_{i1}, w_{i2}, w_{i3}, w_{i4}, w_{i5}, w_{i6})$$

- It becomes n=6 IVPs

Exercise: Solar System Simulator

- Create a model file model.txt that contains the particles

```
Sun      1.989e33  0.000e00
Mercury  3.302e26  0.390e00
Venus    4.869e27  0.720e00
Earth    5.974e27  1.000e00
Mars     6.419e26  1.520e00
Jupiter  1.899e30  5.200e00
Saturn   5.685e29  9.580e00
Uranus   8.683e28  1.920e01
Neptune  1.024e29  3.005e01
```

name mass [g] Distance [AU]

- Modify the binary simulations code to read the model

Exercise: Solar System Simulator

- Extra:
- Try adding comets, asteroid, or satellites
- Extent to 3D forces
- GR effects?
- ….(more) …

Exercise: Solar System Simulator

- The program we write is a very simple N-body code
- How to improve the accuracy?
- How to improve the efficiency?
- Is our initial condition correct?
-

N-Body code for large N

- Direct N-body code cannot handle large N simulation
- Force calculation takes ~$O(N^2)$
- Tree Method (Barnes & Hut, 1986) ~ O(N logN)

MENU

nature

International journal of science

Letter | Published: 04 December 1986

A hierarchical O(N *log* N) force-calculation algorithm

Josh Barnes & Piet Hut

Nature **324**, 446–449 (1986) | Download Citation

Problem Set 6

https://kuochuanpan.github.io/courses/109ASTR660_CA/

Next lecture

- Boundary Value Problems