Financial Systems Design Part II

Ordinary Differential Equations

July 2020

Slido Event: #FSD2

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Outline

- Ordinary Differential Equations (ODEs)
- Euler's Method
- Runge Kutta
- 2^{nd} Order Linear ODEs
- Error Estimation

Ordinary Differential Equations

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Ordinary Differential Equations

- Differential Equations are equations that relates a function to its derivatives
- E.g. $y'(t) + y(t) + 3 = 0$
- We are interested in what $y(t)$ is given an initial condition (else solution is not unique)
- Not always able to solve the differential equation i.e. find the exact equation for $y(t)$
- So we use computational methods to find answers
- Before moving on to the Black Scholes PDE and FDM
- Start with the simpler Ordinary Differential Equation (ODE) where there is only one variable

Ordinary Differential Equations

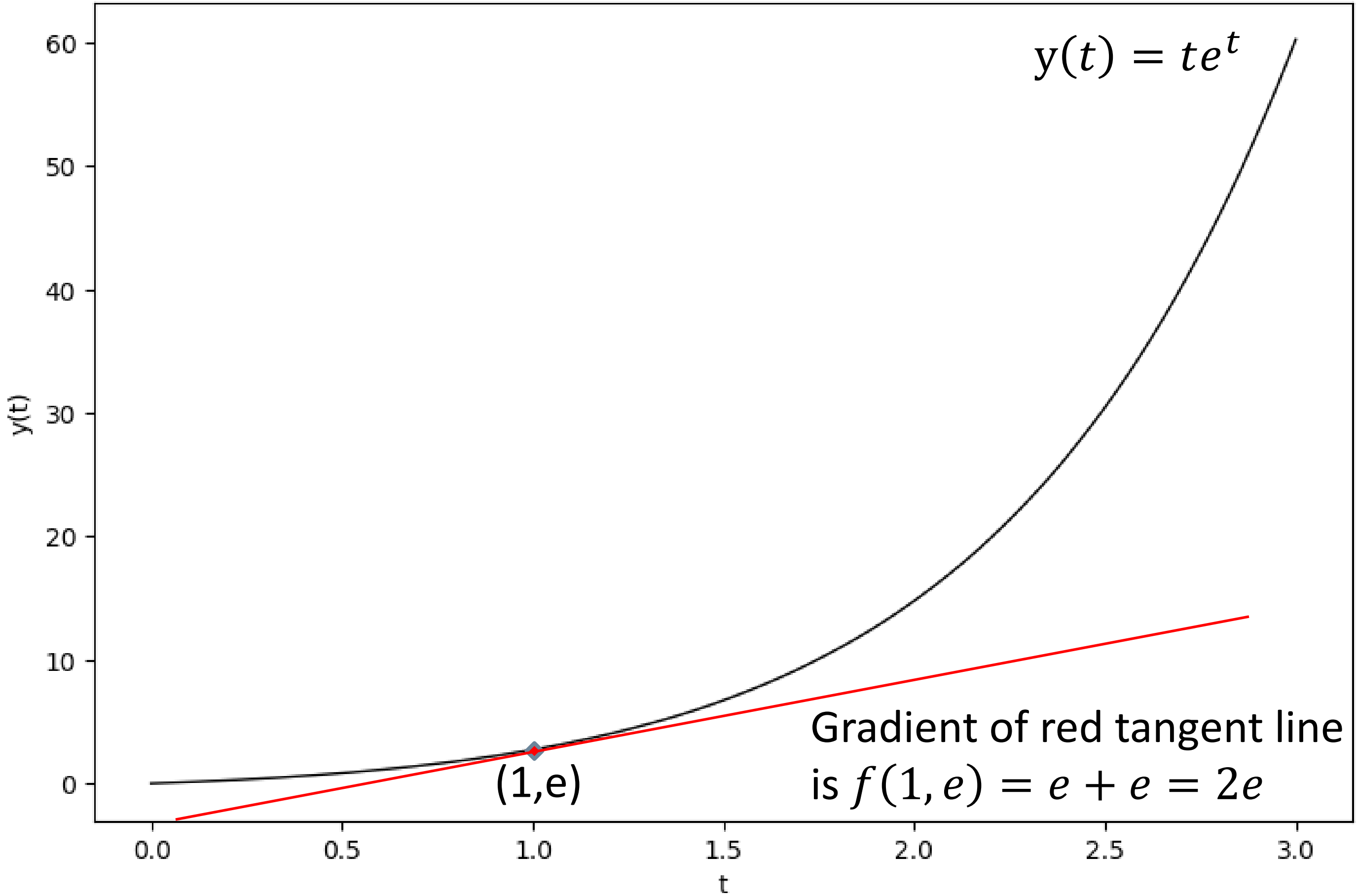
- Start with 1 dimension i.e. ODE with the general form:

$$y'(t) = f\big(t, y(t)\big),\ y(0) = y_0$$

- Given y when t = 0, we want to approximate y at a later time t
- $f\big(t, y(t)\big)$ is the derivative of y with respect to t at the point (t, y)
- Graphically, it is the gradient of the tangent at the point (t, y) i.e. the slope

Ordinary Differential Equations

$$y'(t) = f(t, y) = e^t + y \text{ and } y(0) = 0$$



Ordinary Differential Equations

- Given $y'(t) = f\big(t, y(t)\big),\ y(0) = y_0$
- Can we use the information we have about the slope to find the value of y at t?
- Remember $y'(t) = \frac{dy}{dt}$ i.e. it tells us the ratio of the change of y with respect to a change in t for small t
- So what if we use $\hat{y}_{\Delta t} = y(0) + \Delta t\, y'(0)$?

Euler Method

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Intuition

- A car which is initially beside you is driving along a straight road away from you at a constant speed of 90km/h
- How far is the car away from you after 10 hours
- Distance travelled = 90km/h x 10h = 900km
- Therefore distance away from you is 900km
- To simply into equations, we can abstract the problem
- $y(0) = 0$, speed $y'(t) = 90$ and Δt is 10
- $y(10) = y(0) + y'(0)\Delta t = 0 + 90(10) = 900$

Intuition

- If the car's speed is constantly changing and you are given a function that gives you the exact speed at any point in time i.e. $y'(t)$, how would you calculate the distance from you?
- For example, $y'(t) = 20t - 3t^2$
- Can we take the average speed at the start and end and multiply by the time elapsed?
- If the change in speeds in between is dramatic then the error will be large
- What if you did the same procedure is smaller time steps where the change in speed during each time step is small?

Intuition

- For example, at the start, the car's speed is $y'(0)$ = 0km/h
- You take the time step Δt to be 1h
- After 1h, you estimate the car to have travelled 0km/h x 1h = 0km so the distance $y_1 = y(0) + y'(0)\Delta t = 0$
- Now for the next time step, you check what is the new speed and find that $y'(1) = 20(1) - 3(1^2) = 17$
- Between 1h and 2h, you estimate the car to have travelled $y_2 = y_1 + y'(1)\Delta t = 0 + 17(1) = 17$
- Therefore estimated distance from you after 2h is 17km

slido

Join at slido.com #FSD2

What is the distance of the car from you after 3h using this procedure?

Euler’s Method

- You repeat this procedure until you have 10 intervals and sum up all the distance to get the total distance travelled
- Mathematically we can write down the procedure as follows where $y'(t) = f\big(t, y(t)\big)$

$\hat{y}_0 = y(0)$ (Initial condition or boundary condition)

$$\hat{y}_{\Delta t} = \hat{y}_0 + \Delta t f(0, \hat{y}_0)$$

$$\hat{y}_{2\Delta t} = \hat{y}_{\Delta t} + \Delta t f(\Delta t, \hat{y}_{\Delta t})$$

…

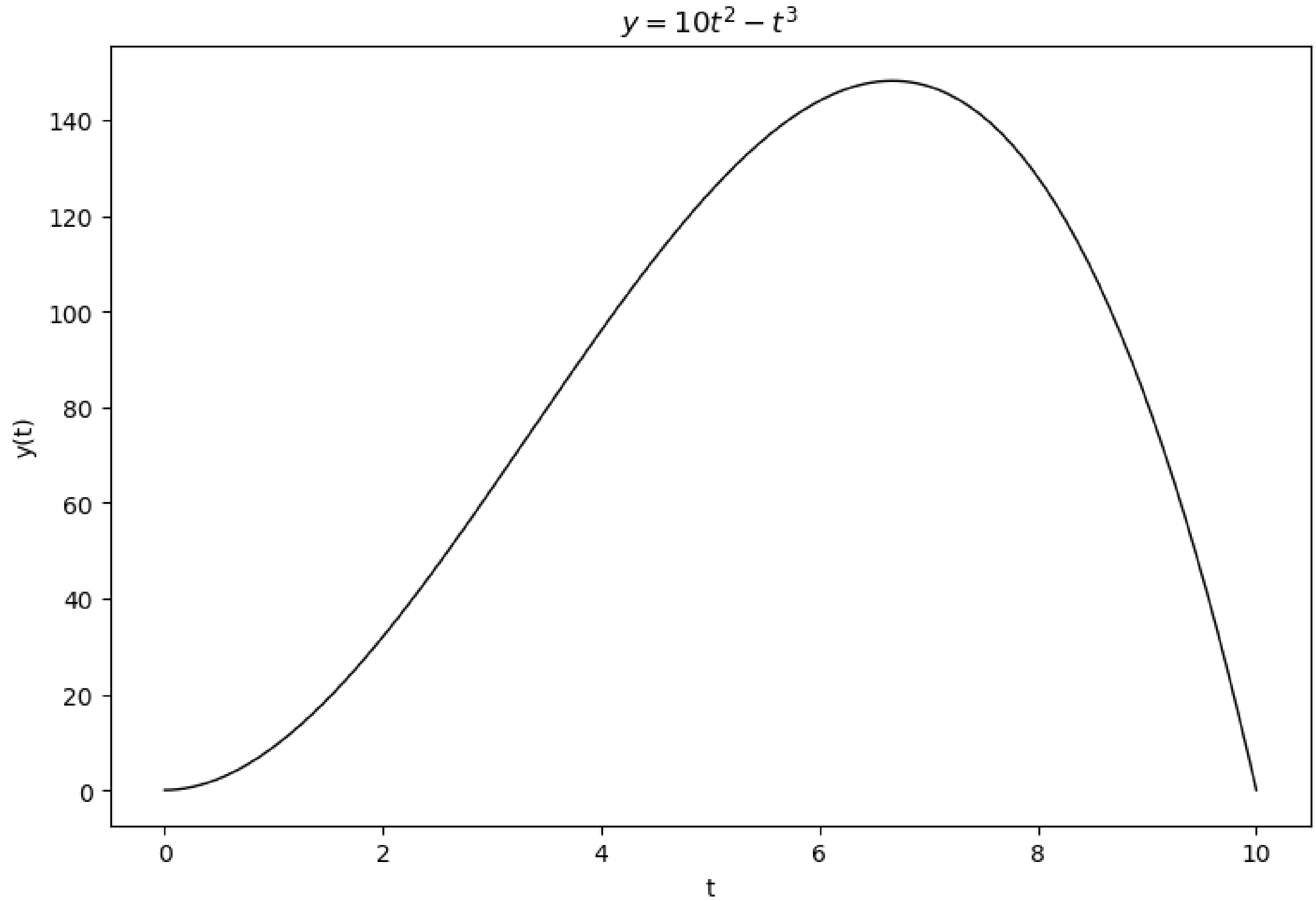
$$\hat{y}_{(i+1)\Delta t} = \hat{y}_{i\Delta t} + \Delta t f(i\Delta t, \hat{y}_{i\Delta t})$$

- This is EULER’S METHOD

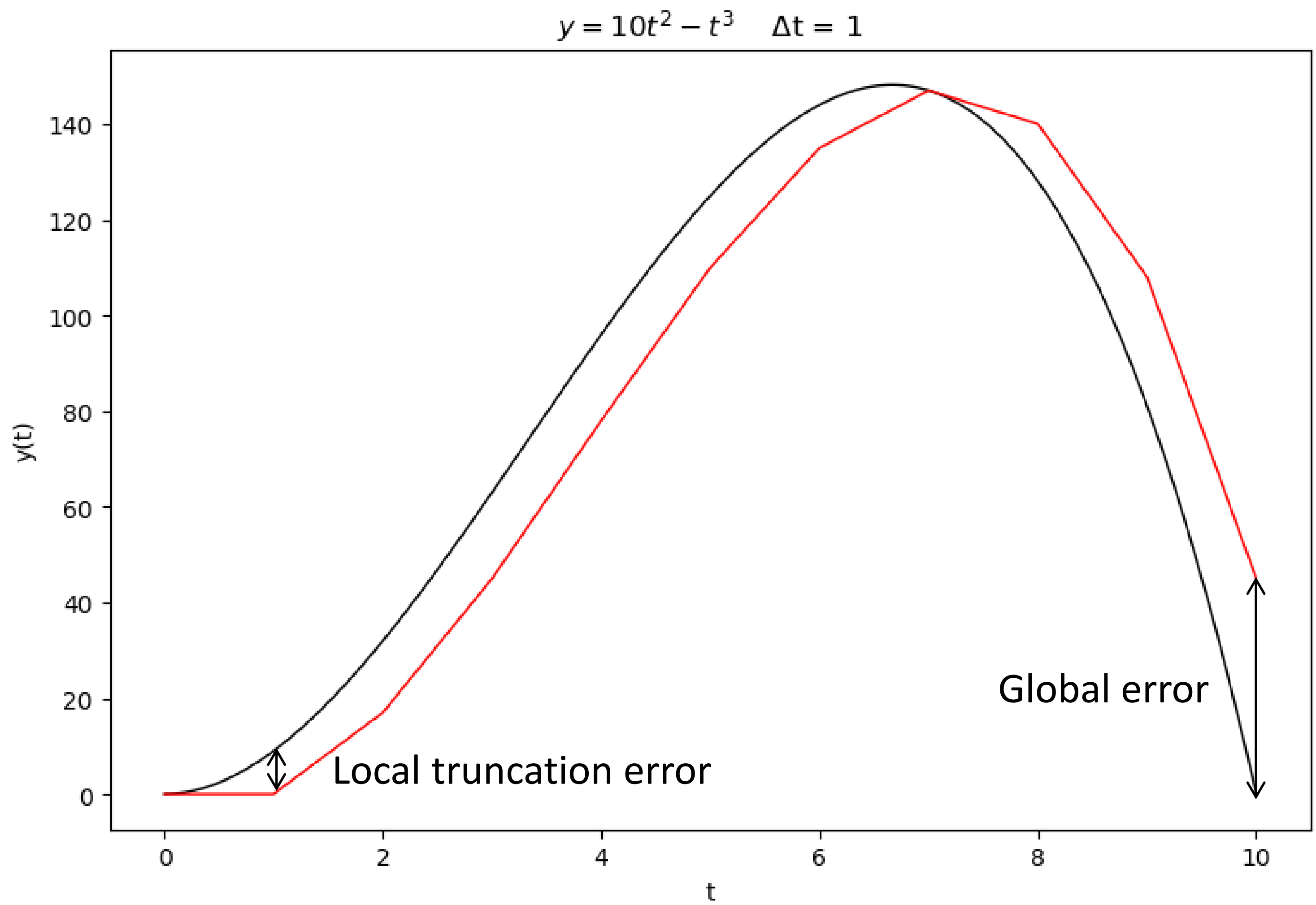
Euler's Method

- $y'(t) = 20t - 3t^2$
- $y(0) = 0$ and we want to know $y(10)$
- Of course, we know that through integration that $y(t) = 10t^2 - t^3$
- Sometimes we cannot find the analytical solution in real world problems
- Using Euler's method, we try to reconstruct the curve point by point with different step size Δt

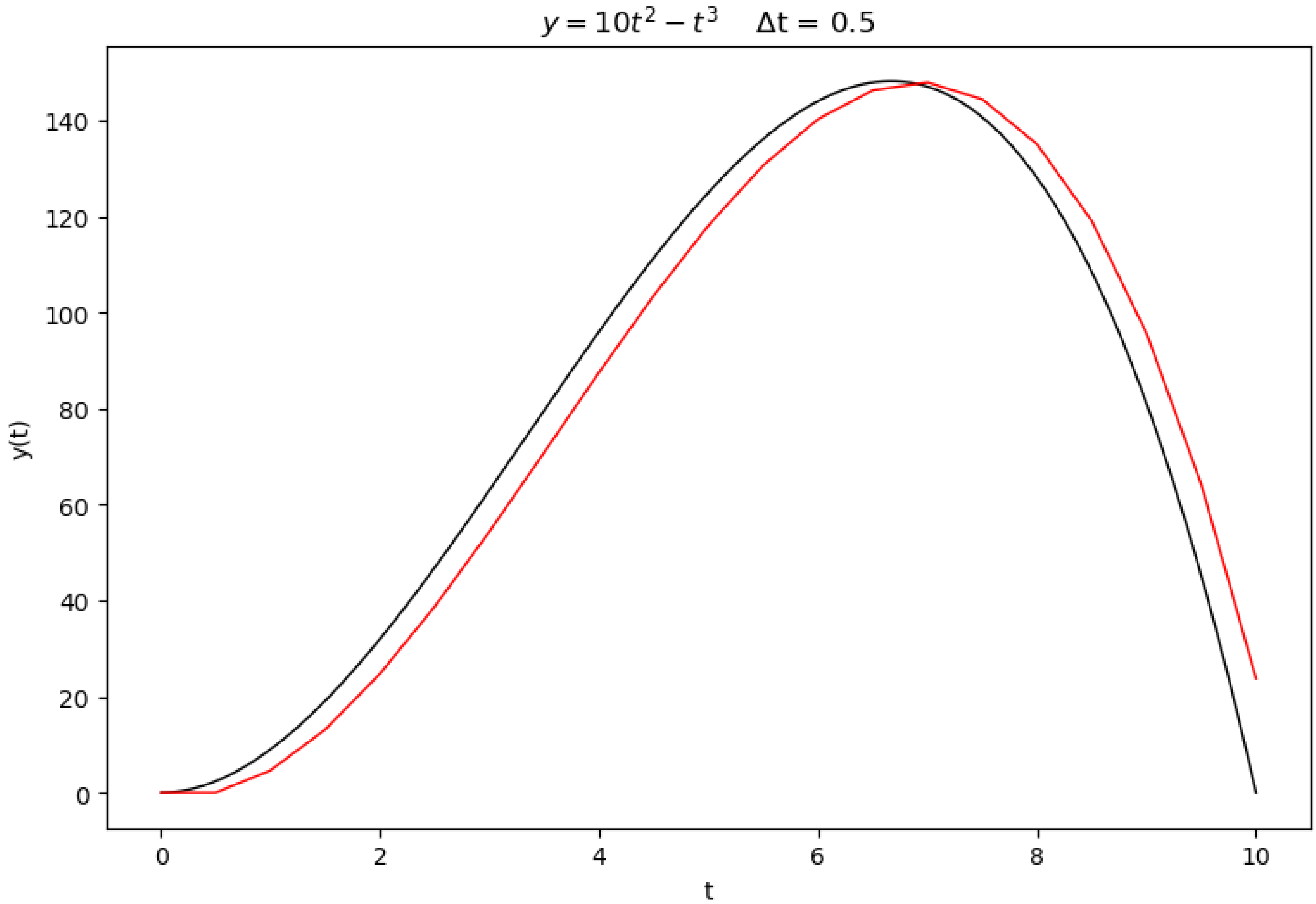
Euler's Method



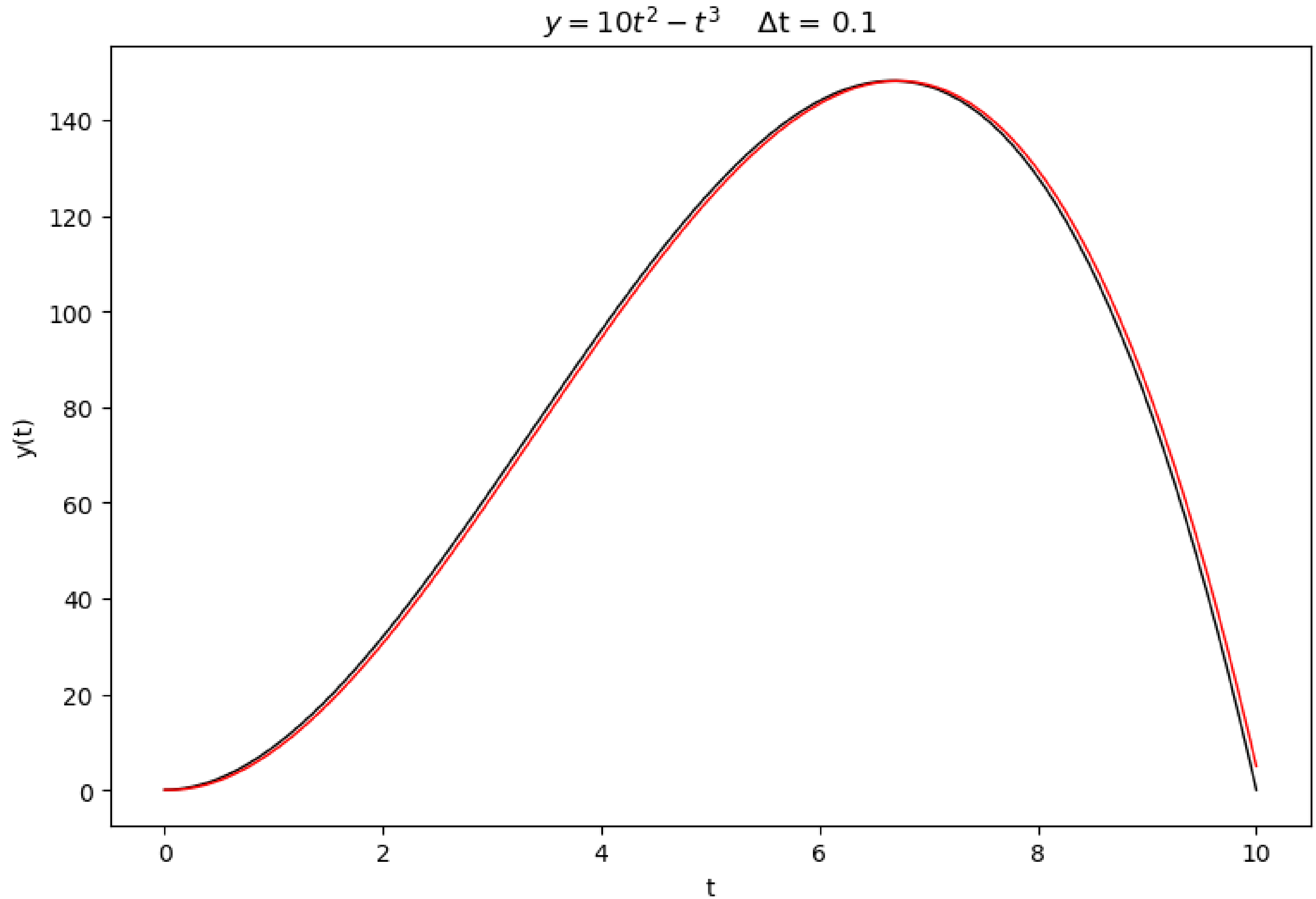
Euler's Method



Euler's Method



Euler's Method



Euler's Method

- We can use Taylor's series to compare it to the true solution and find the error using this method
- This is our estimate

$$\hat{y}_{t+\Delta t} = y(t) + \Delta t y'(t)$$

- Taylor's expansion about the point t

$$y(t + \Delta t) = y(t) + \Delta t y'(t) + \frac{1}{2}\Delta t^2 y''(t) + \cdots$$

Euler's Method

- We can then compare these two values
- We find

$$y(t + \Delta t) - \hat{y}_{t+\Delta t} = \frac{1}{2}\Delta t^2 y''(t) + \cdots$$

- This is called the LOCAL TRUNCATION ERROR
- In this case the error is of order Δt^2

In-Class Exercise

- Let

$$y'(t) = f(t, y) = \frac{1 - 3t}{4t + 4} y, \qquad y(0) = 1$$

- Fill in the missing code to use Euler's method for approximating $y(t)$
- Find $y(t)$ from $t = 0$ to $t = 1$, using Δt = 0.1, 0.01, 0.001, 0.0001
- Plot the solutions of $y(t)$ vs t for each Δt on the same graph (4 curves on the same axis)

In-Class Exercise

- The solution to this ODE is

$$y(t) = (t + 1)e^{-3t/4}$$

- Plot the error i.e. approximate solution minus the actual solution on a 2nd plot
- Examine the error at t = Δt and t = 1 for each value of Δt

slido

Join at slido.com #FSD2

How does the error at $t = 1$ compare with the error at $t = \Delta t$?

Ordinary Differential Equations

- Recall that we found

$$y(t+\Delta t) - \hat{y}_{t+\Delta t} = \frac{1}{2}\Delta t^2 y''(t) + \cdots$$

- But when we look at the actual error at t = 1 we only find first order accuracy. Why is that?
- The theoretical error was at the first grid point
- The error at the last grid point sums up all the LOCAL errors T/Δt times, so GLOBAL error goes down by 1 order

slido

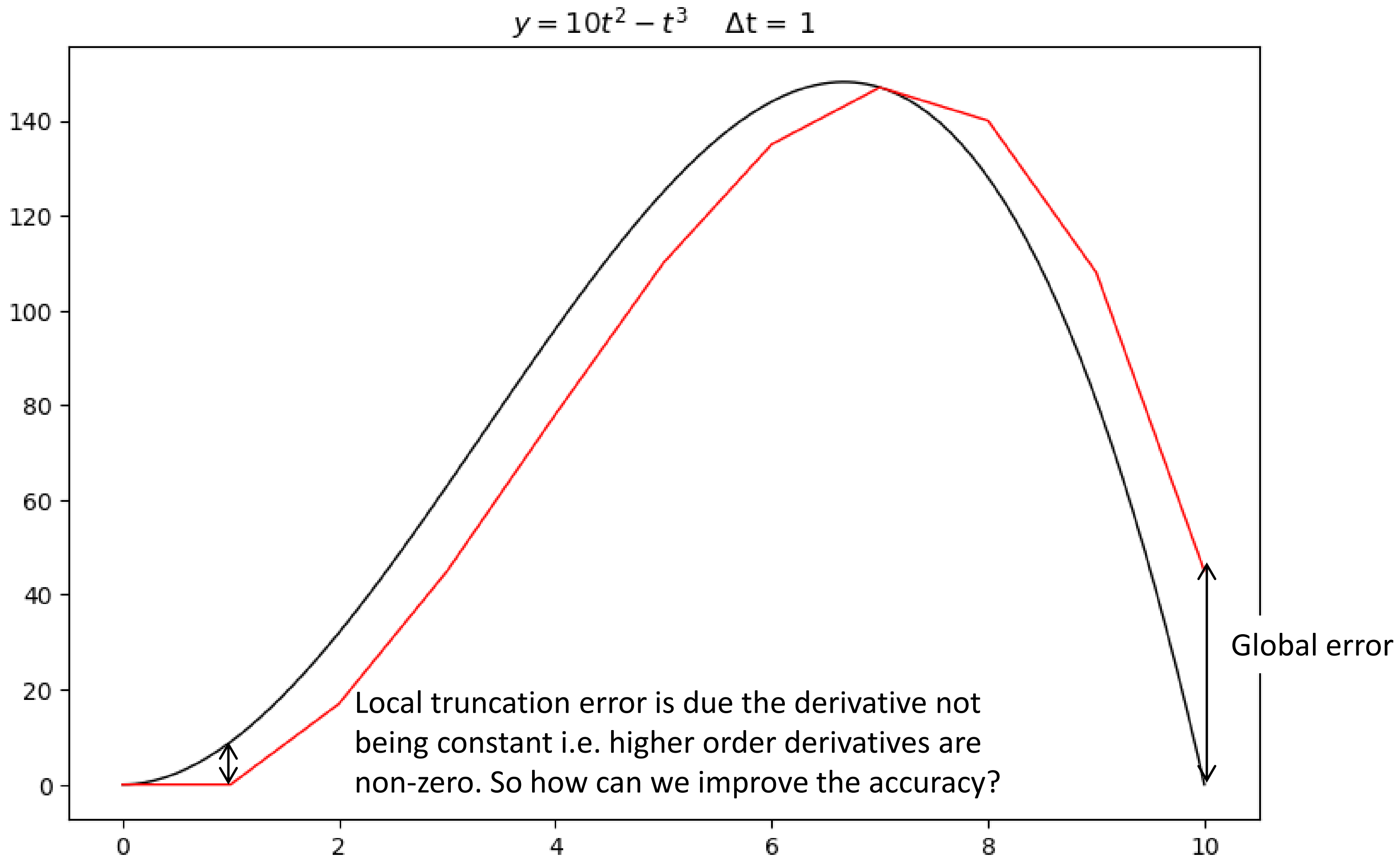
Join at
slido.com
#FSD2

Audience Q&A Session

Runge Kutta

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Local Truncation Error



Runge Kutta

- The Euler's method computes the derivative (slope) one step at a time
- This results in large errors if the slope is not stable i.e. higher order derivatives are large in value
- Runge Kutta finds better estimates of the slope by using a weighted average of the slope at different points in the interval to improve the approximation
- E.g. Calculate the slope at the 2 end points of the interval and take the average
- This is RUNGE-KUTTA 2 or HEUN'S METHOD

2^{nd} Order Runge Kutta

- If derivative function is dependent on y then the value of y at the end point is needed to compute the slope at that point
- i.e. Need $f(t + \Delta t, y(t + \Delta t))$ but we don't know $y(t + \Delta t)$
- We use Euler's method to estimate this value
- Let us temporarily define

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

$$\tilde{y}_{t+\Delta t} = y(t) + \Delta t f(t, y(t)) \text{ (Euler estimate)}$$

$$\hat{y}_{t+\Delta t} = y(t) + \frac{\Delta t}{2}[f(t, y(t)) + f(t + \Delta t, \tilde{y}_{t+\Delta t})] \text{ (RK2 estimate)}$$

2^{nd} Order Runge Kutta

- Let's examine the local error of this approximation

$$\hat{y}_{t+\Delta t} = y(t) + \tfrac{\Delta t}{2}[f(t, y(t)) + f(t + \Delta t, \tilde{y}_{t+\Delta t})]$$

- Now use a Taylor series of $f(t + \Delta t, \tilde{y}_{t+\Delta t})$ around the point $\big(t, y(t)\big)$

- $f(t + \Delta t, \tilde{y}_{t+\Delta t}) = f\big(t + \Delta t, y(t) + \Delta t f(t, y(t))\big) = f(t, y(t)) + \Delta t\big[f_t(t, y(t)) + f\big(t, y(t)\big) f_y(t, y(t))\big] + O(\Delta t^2)$

- Substituting into the RK2 estimate equation and simplifying

2^{nd} Order Runge Kutta

$$\hat{y}_{t+\Delta t} = y(t) + \Delta t f\big(t, y(t)\big) + \frac{\Delta t^2}{2}\left[f_t\big(t, y(t)\big) + f\big(t, y(t)\big) f_y(0, y_0)\right] + O(\Delta t^3)$$

- Using the definition that $y'(t) = f(t, y(t))$ and

$$y''(t) = f_t(t, y(t)) + y'(t) f_y(t, y(t))$$ (chain rule)

- Then $\hat{y}_{t+\Delta t} = y(t) + \Delta t y'(t) + \frac{\Delta t^2}{2} y''(t) + O(\Delta t^3)$
- Therefore using the RK2 we get *local* error on the order of Δt^3
- Again, when we use this over and over again we get *global* error on the order of Δt^2

2nd Order Runge Kutta

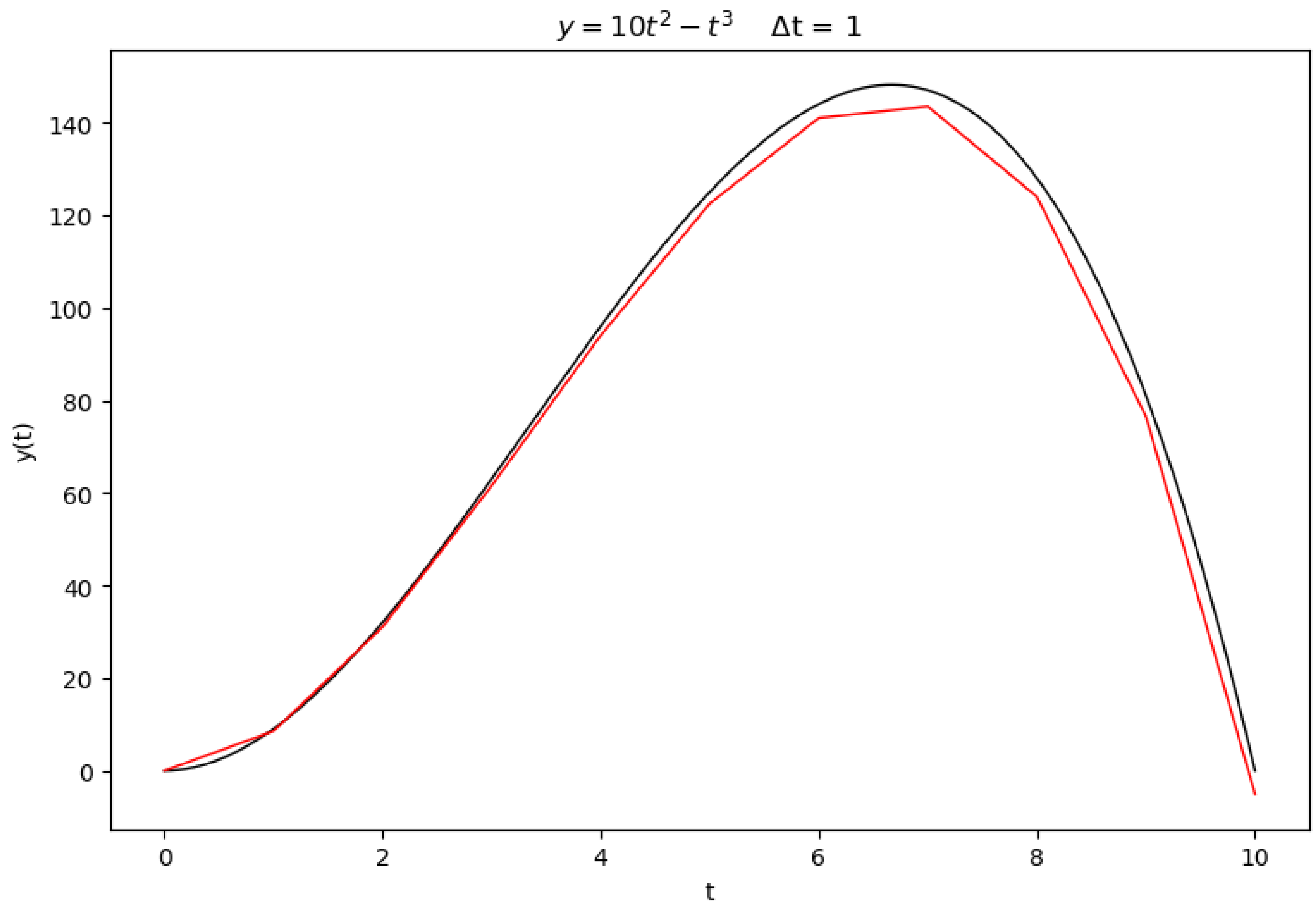
- RK2 or Heun's Method is summarized as

$$k_1 = \Delta t f(t, \hat{y}_t)$$

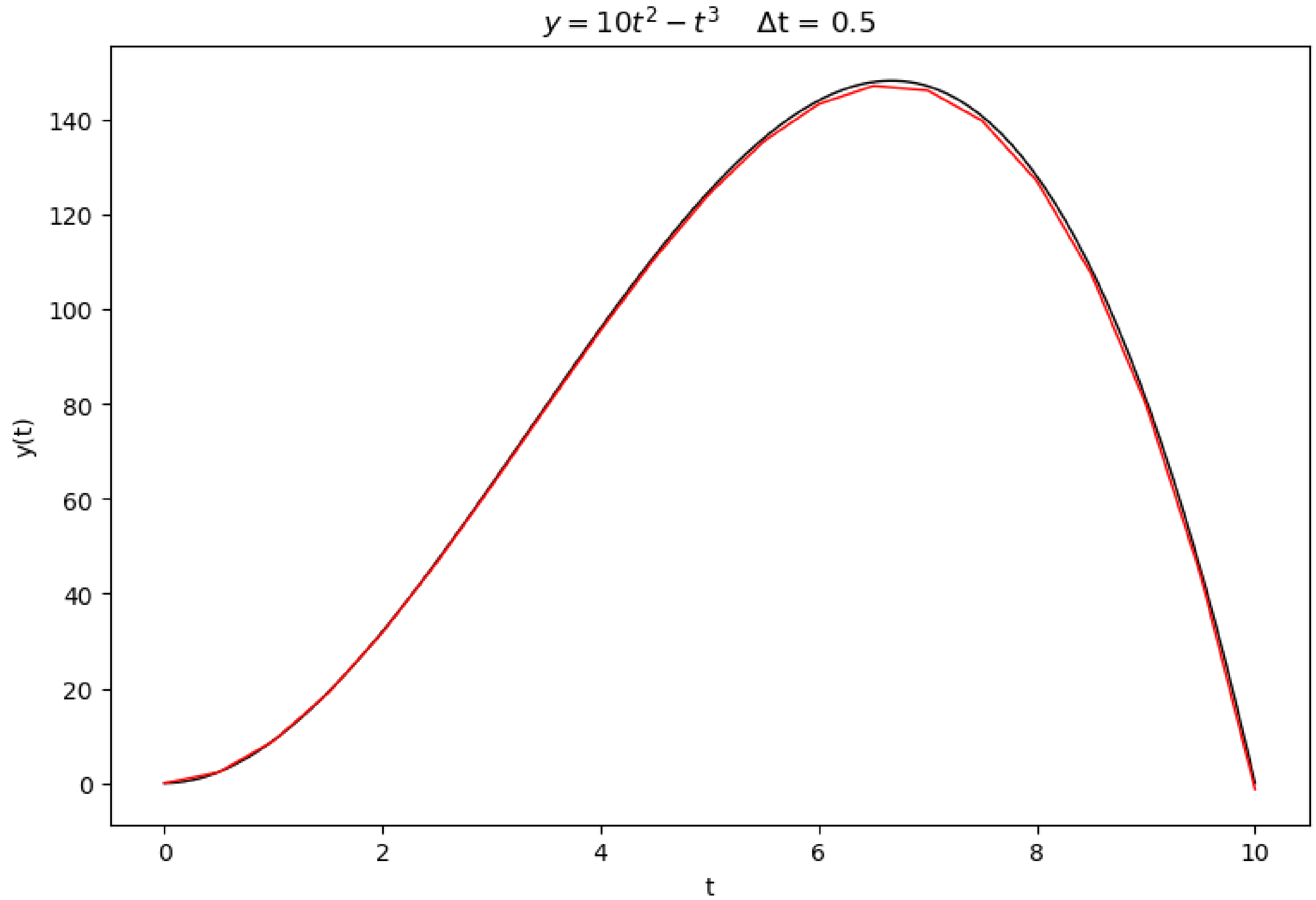
$$k_2 = \Delta t f(t + \Delta t, \hat{y}_t + k_1)$$

$$\hat{y}_{t+\Delta t} = \hat{y}_t + \tfrac{1}{2}k_1 + \tfrac{1}{2}k_2$$

2nd Order Runge Kutta



2^{nd} Order Runge Kutta



In-Class Exercise

- Let

$$y'(t) = f(t, y) = \frac{1 - 3t}{4t + 4} y, \qquad y(0) = 1$$

- Fill in the missing code to use Heun's method for approximating $y(t)$
- Find $y(t)$ from $t = 0$ to $t = 1$, using Δt = 0.1, 0.01, 0.001, 0.0001
- Plot $y(t)$ vs t

In-Class Exercise

- The solution to this ODE is

$$y(t) = (t + 1)e^{-3t/4}$$

- Plot the error i.e. approximate solution minus the actual solution on a 2^{nd} plot
- Examine the error at t = Δt and t = 1 for each value of Δt

2nd Order Runge Kutta

- Another form of the 2nd Order Runge Kutta

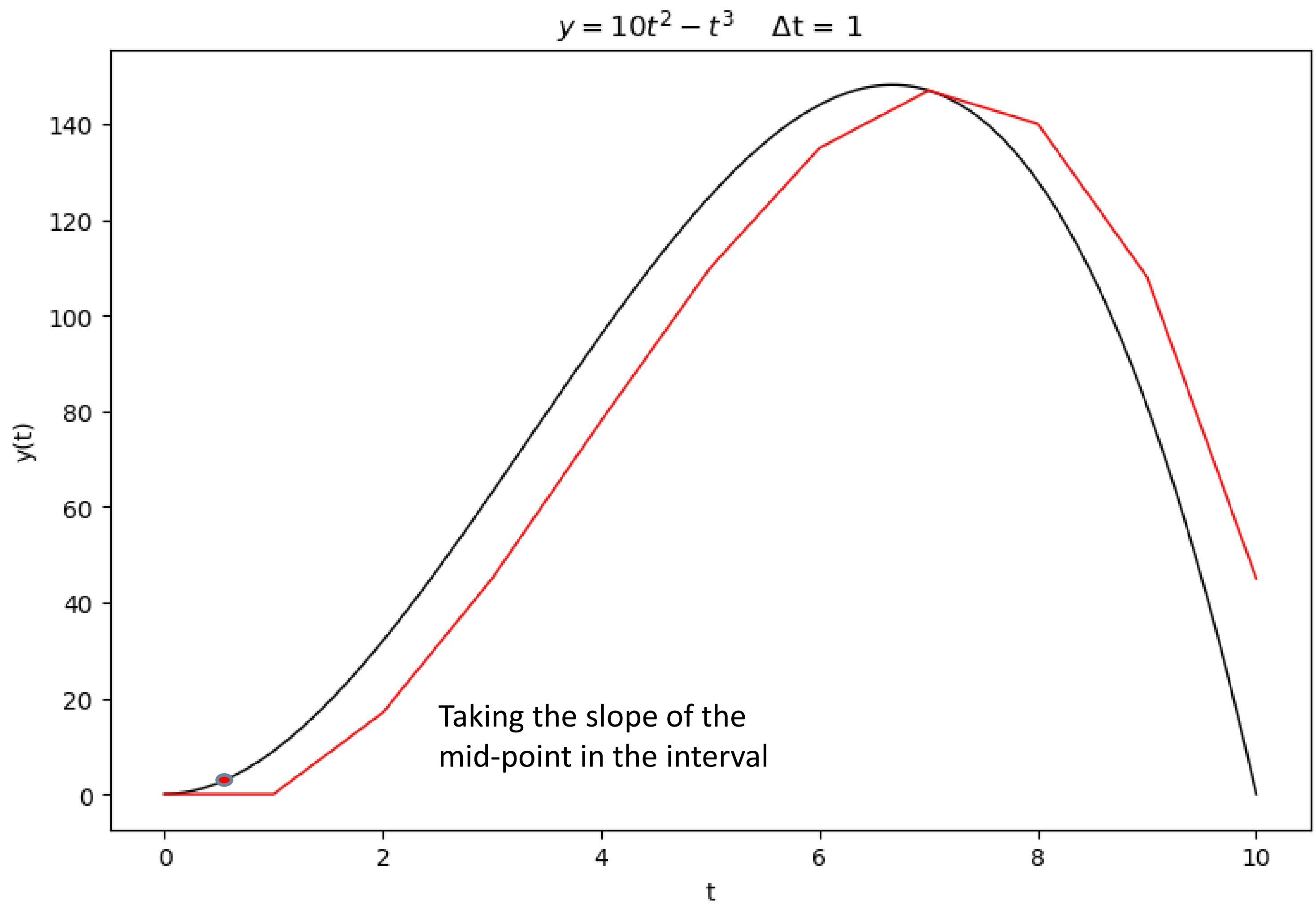
$$k_1 = f(t, \hat{y}_t)$$

$$\hat{y}_{t+\Delta t/2} = \hat{y}_t + \frac{\Delta t}{2} k_1$$

$$k_2 = f\left(t + \frac{\Delta t}{2}, \hat{y}_{t+\Delta t/2}\right)$$

$$\hat{y}_{t+\Delta t} = \hat{y}_t + k_2 \Delta t$$

2nd Order Runge Kutta



4^{th} Order Runge Kutta

- A popular method is RUNGE-KUTTA 4 (RK4)

$$k_1 = f(t, \hat{y}_t)$$

$$k_2 = f\left(t + \tfrac{1}{2}\Delta t, \hat{y}_t + \tfrac{1}{2}k_1\right)$$

$$k_3 = f\left(t + \tfrac{1}{2}\Delta t, \hat{y}_t + \tfrac{1}{2}k_2\right)$$

$$k_4 = f(t + \Delta t, \hat{y}_t + k_3)$$

$$\hat{y}_{t+\Delta t} = \hat{y}_t + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

- Local error is 5^{th} order, global error is 4^{th} order

slido

Join at
slido.com
#FSD2

Audience Q&A Session

2^{nd} Order Linear ODE

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

2^{nd} Order ODEs

- We’ve done a lot for first order ODEs
- The Black-Scholes equation has a second derivative
- Let’s talk about second order ODEs

2nd Order ODEs

- The general form for a *linear* second order ODE is

$$f_1(x)y(x) + f_2(x)y'(x) + f_3(x)y''(x) = f_0(x)$$

$$\left.\begin{aligned} g_1 y(a) + g_2 y'(a) &= g_0 \\ k_1 y(b) + k_2 y'(b) &= k_0 \end{aligned}\right\} \text{Boundary conditions}$$

- For $a \leq x \leq b$ (switched t to x)

2^{nd} Order ODEs

- Let $x_0 = a$ and $x_n = b$
- $\Delta x = \frac{b-a}{n}$ and $x_i = a + i\Delta x$ e.g. $x_3 = a + 3\Delta x$
- $y_i = y(x_i)$ e.g. $y_5 = y(x_5)$
- Then the boundary conditions can be approximated using first order approximations to $y'(a)$ and $y'(b)$

$$g_1 y_1 + g_2 \frac{y_2 - y_1}{\Delta x} = g_0$$

$$\left(g_1 - \frac{g_2}{\Delta x}\right) y_1 + \frac{g_2}{\Delta x} y_2 = g_0$$

- Approximate right boundary condition in the same way

2nd Order ODEs

- The general equation can be approximated as

$$f_1(x_i)y_i + f_2(x_i)\frac{y_{i+1} - y_{i-1}}{2\Delta x} + f_3(x_i)\frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2} = f_0(x_i)$$

- Rearranging the terms in terms points along y

$$\left(\frac{-f_2(x_i)}{2\Delta x} + \frac{f_3(x_i)}{\Delta x^2}\right){\color{red}y_{i-1}} + \left(f_1(x_i) - \frac{2f_3(x_i)}{\Delta x^2}\right){\color{red}y_i} + \left(\frac{f_2(x_i)}{2\Delta x} + \frac{f_3(x_i)}{\Delta x^2}\right){\color{red}y_{i+1}} = f_0(x_i)$$

- This is done as this expression can be translated easily to a matrix equation

2^{nd} Order ODEs

- Say we start with a known y_0 and we want to find y_1
- We find that the equations requires y_2 to find y_1

$$\left(\frac{-f_2(x_1)}{2\Delta x} + \frac{f_3(x_1)}{\Delta x^2}\right) y_0 + \left(f_1(x_i) - \frac{2f_3(x_1)}{\Delta x^2}\right) y_1 + \left(\frac{f_2(x_1)}{2\Delta x} + \frac{f_3(x_1)}{\Delta x^2}\right) y_2 = f_0(x_1)$$

- And if we move to $i = 2$ to try and find y_2 then we find that we need y_3
- Therefore instead of a point by point evolution as with Euler's method or RK2, we have a set of interconnected points that must be solved simultaneously

2^{nd} Order ODEs

- With

$$\left(\frac{-f_2(x_i)}{2\Delta x} + \frac{f_3(x_i)}{\Delta x^2}\right) y_{i-1} + \left(f_1(x_i) - \frac{2f_3(x_i)}{\Delta x^2}\right) y_i + \left(\frac{f_2(x_i)}{2\Delta x} + \frac{f_3(x_i)}{\Delta x^2}\right) y_{i+1} = f_0(x_i)$$

- There are n-2 equations from $i = 2$ to $i = n - 1$
- Plus 1 equation for each of the 2 boundary conditions
- All together there are n equations and n unknowns i.e. y_1 to y_n
- We can solve this system of equations to find y_1 to y_n
- This can be expressed as a matrix equation

2^{nd} Order ODEs

$$\begin{bmatrix} \left(g_1 - \frac{g_2}{\Delta x}\right) & \frac{g_2}{\Delta x} & 0 & \cdots & 0 & 0 \\ \left(-\frac{f_2(a+\Delta x)}{2\Delta x} + \frac{f_3(a+\Delta x)}{\Delta x^2}\right) & \left(f_1(a+\Delta x) - \frac{2f_3(a+\Delta x)}{\Delta x^2}\right) & \left(\frac{f_2(a+\Delta x)}{2\Delta x} + \frac{f_3(a+\Delta x)}{\Delta x^2}\right) & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \cdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \left(k_1 - \frac{k_2}{\Delta x}\right) & \frac{k_2}{\Delta x} \end{bmatrix} \begin{bmatrix} y_a \\ y_{a+\Delta x} \\ y_{a+2\Delta x} \\ \vdots \\ y_{b-\Delta x} \\ y_b \end{bmatrix} =$$

$$\begin{bmatrix} g_0 \\ f_0(a+\Delta x) \\ f_0(a+2\Delta x) \\ \vdots \\ f_0(b-\Delta x) \\ k_0 \end{bmatrix}$$

Linear Algebra

- How do we solve this matrix equation using code?
- We use an easy example to get a feel of what we are trying to accomplish
- Given 2 equations and 2 unknowns
  - $2x + 4y = 2$
  - $2x – 4y = -6$
- Add 2nd equation to the 1st and get $4x=-4$ i.e. $x = -1$
- From there use any of the 2 equations to get $y = 1$

Linear Algebra

- The dimension of the systems of equations that we will be dealing with are far larger
- The matrix form of the problem is a well studied problem under Numerical Linear Algebra
- There are provided algorithms in Scipy
- Let's still try to get a feel for it in matrix form

In-Class Exercise

- $\begin{pmatrix} 2 & 4 \\ 2 & -4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ -6 \end{pmatrix}$
- $Ax = b$
- $A^{-1}Ax = A^{-1}b$ so x = $A^{-1}b$
- Create matrices for this simple example with numpy
- Use the inv() function within the linalg module of numpy to generate the inverse matrix
- Solve for the unknown vector x by multiplying the inverse matrix of A to the vector b

Linear Algebra

- Recall the row operations to find the inverse of a matrix
- $\begin{pmatrix} 2 & 4 \\ 2 & -4 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
- $\begin{pmatrix} 4 & 0 \\ 2 & -4 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0 \\ 2 & -4 \end{pmatrix} \begin{pmatrix} 0.25 & 0.25 \\ 0 & 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0 \\ 0 & -4 \end{pmatrix} \begin{pmatrix} 0.25 & 0.25 \\ -0.5 & 0.5 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0.25 & 0.25 \\ 0.125 & -0.125 \end{pmatrix}$

Linear Algebra

- Finding the inverse of the matrix is a very costly operation
- There are more efficient ways of finding the solution to $Ax = b$ which do not require the inverse as an intermediate step
- The LU decomposition is one method that is easy to implement for tridiagonal matrices

LU Decomposition

- Decompose A into LU where L is lower triangular and U is upper triangular
- $Ax = b$ becomes $LUx = b$
- Further decompose into 2 equations
  - $Ly = b$
  - $Ux = y$
- As A is tri-diagonal then L will only have non-zero elements in main diagonal and the diagonal below it
- Similarly for U on the main diagonal and just above it

LU Decomposition

$$\begin{pmatrix} a_1 & c_1 & 0 & \cdots & 0 \\ b_1 & a_2 & c_2 & \ddots & \vdots \\ 0 & b_2 & a_3 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & c_{n-1} \\ 0 & 0 & \cdots & b_{n-1} & a_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_1 & 1 & 0 & \ddots & \vdots \\ 0 & l_2 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & l_{n-1} & 1 \end{pmatrix} \begin{pmatrix} d_1 & u_1 & 0 & \cdots & 0 \\ 0 & d_2 & u_2 & \ddots & \vdots \\ 0 & 0 & d_3 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & u_{n-1} \\ 0 & 0 & \cdots & 0 & d_n \end{pmatrix}$$

- $d_1 = a_1$
- $l_i d_i = b_i \Rightarrow l_i = \frac{b_i}{d_i}$
- $u_i = c_i$
- $d_{i+1} = a_{i+1} - l_i u_i$
- i goes from 1 to n-1

A) The algorithm starts by finding d_1

B) Then using d_1, you find l_1

C) Using l_1, you can find d_2

D) u_1 is directly found by reading c_1

E) Repeat the steps above starting with d_2

Forward Substitution

- Solving $Ly = b$ uses forward substitution
- Simple example

$$\begin{pmatrix} 1 & 0 & 0 \\ l_1 & 1 & 0 \\ 0 & l_2 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- $y_1 = b_1$
- $y_2 = b_2 - l_1 y_1$
- $y_3 = b_3 - l_2 y_2$
- In general, $y_{i+1} = b_{i+1} - l_i y_i$ from $i = 1$ to $n - 1$

Backward Substitution

- Solving $Ux = y$ uses backward substitution
- Simple example

$$\begin{pmatrix} d_1 & u_1 & 0 \\ 0 & d_2 & u_2 \\ 0 & 0 & d_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

- $x_3 = \frac{y_3}{d_3}$
- $x_2 = \frac{y_2 - u_2 x_3}{d_2}$
- $x_1 = \frac{y_1 - u_1 x_2}{d_1}$
- In general, $x_{i-1} = \frac{y_{i-1} - u_{i-1} x_i}{d_{i-1}}$ from $i = n$ to 2

In Class Exercise

- Verify the LU Decomposition works by solving the following system of equations
- 2x + 3y = 8
- 3x + 4y + z = 14
- y + 3z = 11
- $Ax = b \; is \begin{pmatrix} 2 & 3 & 0 \\ 3 & 4 & 1 \\ 0 & 1 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 8 \\ 14 \\ 11 \end{pmatrix}$
- Use 1 array to store each diagonal of the matrix
- Solution is x = 1, y = 2, z = 3

Linear Algebra

- LU Decomposition is efficient for cases where you solve for x with different b vectors but A remains the same
- There is another type of method that is "indirect"
- It requires iterations where you apply an operation to an initial guess
- The guess will converge to the solution as more iterations are applied to the desired accuracy
- We will discuss this if we have sufficient time for American options

Back to this

$$\begin{bmatrix} \left(g_1 - \frac{g_2}{\Delta x}\right) & \frac{g_2}{\Delta x} & 0 & \cdots & 0 & 0 \\ \left(-\frac{f_2(a+\Delta x)}{2\Delta x} + \frac{f_3(a+\Delta x)}{\Delta x^2}\right) & \left(f_1(a+\Delta x) - \frac{2f_3(a+\Delta x)}{\Delta x^2}\right) & \left(\frac{f_2(a+\Delta x)}{2\Delta x} + \frac{f_3(a+\Delta x)}{\Delta x^2}\right) & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \cdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \left(k_1 - \frac{k_2}{\Delta x}\right) & \frac{k_2}{\Delta x} \end{bmatrix} \begin{bmatrix} y_a \\ y_{a+\Delta x} \\ y_{a+2\Delta x} \\ \vdots \\ y_{b-\Delta x} \\ y_b \end{bmatrix} = \begin{bmatrix} g_0 \\ f_0(a+\Delta x) \\ f_0(a+2\Delta x) \\ \vdots \\ f_0(b-\Delta x) \\ k_0 \end{bmatrix}$$

Matrix Representation

- The matrix is a tri-diagonal so most elements are zero
- It will be a waste of memory to store the full matrix
- Calculations will be more efficient if only non-zero elements are processed
- Recall from the first class that the Scipy package has built in functions for sparse matrices
- We can use built in functions in Scipy to construct this matrix and perform the calculations more efficiently

In-Class Exercise

- Given

$$x^2 e^{x^2} y(x) + 2x e^{x^2} y'(x) + e^{x^2} y''(x) = x^3 - 4x$$

$$y(0) = 0$$

$$y'(1) = -\frac{1}{e}$$

- Construct the matrix-vector equation to solve for y in the interval (0,1) for x using a step size of 0.001

In-Class Exercise

- Use LU Decomposition to solve the system of equations and obtain the solution vector for y
- The actual solution to this ODE is

$$y(x) = xe^{-x^2}$$

- Plot the approximated curve with the actual curve
- Examine the error by plotting the actual minus the approximate values of y

In-Class Exercise

- Let $x_1 = a$ and $x_n = b$ so e.g. $x_{n-1} = b - \Delta x$
- $y_n = y(x_n)$ and $y_{n-1} = y(x_{n-1})$
- The second boundary condition

$$k_1 y_n + k_2 \frac{y_n - y_{n-1}}{\Delta x} = k_0$$

$$\left(k_1 - \frac{k_2}{\Delta x}\right) y_{n-1} + \frac{k_2}{\Delta x} y_n = k_0$$

In-Class Exercise

- Do the same as the previous exercise but use the solver spsolve in Scipy instead of LU Decomposition

Join at
slido.com
#FSD2

Audience Q&A Session

Error Estimation

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Error Estimation

- Let's think about a situation where we don't already know the exact solution to an ODE
- Our estimate, as a function of step size, is the true value plus an error
  - On the order of the first term in the Taylor series that didn't go away
  - Forget about higher order terms for now

$$\hat{y}(h) \approx y + ch^n$$

Error Estimation

- What we really want to find is the *rate* at which the error goes to zero (i.e. trying to find n)
- As we decrease h, error should go down too
  - Can we figure out the error using multiple *h*'s?
- Assume we have solved the problem for h, $2h$, and $4h$, and let's examine

$$\frac{\hat{y}(4h) - \hat{y}(2h)}{\hat{y}(2h) - \hat{y}(h)}$$

Error Estimation

- If we expand this we will find

$$\frac{\hat{y}(4h) - \hat{y}(2h)}{\hat{y}(2h) - \hat{y}(h)} \approx \frac{y + c(4h)^n - (y + c(2h)^n)}{y + c(2h)^n - (y + ch^n)}$$

$$= \frac{ch^n(4^n - 2^n)}{ch^n(2^n - 1)} = \frac{2^n(2^n - 1)}{2^n - 1} = 2^n$$

- Therefore we can find n by taking log base 2 of this ratio

In-Class Exercise

- let's try to estimate the order of accuracy using the same script for solving linear 2nd Order ODEs
- Note: we must find our approximation at the same x for each h
- Let's try to estimate the error for $x = 0.5$, $\Delta x = 0.0025$

Richardson Extrapolation

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Richardson Extrapolation

- Given an estimation scheme of order n, if we have estimates based on two different step sizes, we can combine them to obtain an estimate of order $n + 1$
- We must know what n is
- For simplicity we typically choose the second step size to be half of the first

Richardson Extrapolation

- Suppose we are using an expression $g(h)$ with error ch^n to estimate a quantity G
- We know $g(\cdot)$ at two different step sizes h_1 and h_2:

$$g(h_1) = G + ch_1^n + O(h_1^{n+1})$$
$$g(h_2) = G + ch_2^n + O(h_2^{n+1})$$

- We combine these two equations to eliminate the error term, obtaining:

$$\hat{G} = \frac{\left(\frac{h_1}{h_2}\right)^n g(h_2) - g(h_1)}{\left(\frac{h_1}{h_2}\right)^n - 1}$$

Richardson Extrapolation

- Letting $h_2 = h_1/2$, we end up with

$$\hat{G} = \frac{2^n g(h/2) - g(h)}{2^n - 1}$$

$$= \frac{2^n \left[G + c\left(\frac{h}{2}\right)^n\right] - (G + ch^n)}{2^n - 1} + O(h^{n+1})$$

$$= \frac{(2^n - 1)G}{2^n - 1} + O(h^{n+1})$$

$$= G + O(h^{n+1})$$

- This is a new estimate for G which has order n+1

Richardson Extrapolation

- From Taylor's theorem

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + ..$$
$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(x)}{2}h + ..$$

- As an example, let's apply Richardson extrapolation to the simplest forward difference equation:

$$G = f'(x), g(h) = \frac{f(x+h)-f(x)}{h} + O(h)$$

- We know $\mathrm{n} = 1$, and we again let $h_2 = h_1/2$

Richardson Extrapolation

- Then

$$\hat{G} = \frac{2\frac{f\left(x+\frac{h}{2}\right)-f(x)}{h/2} - \frac{f(x+h)-f(x)}{h}}{2-1}$$

$$= \frac{4f\left(x+\frac{h}{2}\right)-f(x+h)-3f(x)}{h}$$

- And we now have a new forward difference equation of order 2 which we have seen before if we substitute h=2H

In-Class Exercise

- Copy the code from the Euler method exercise earlier

$$y'(t) = f(t, y) = \frac{1 - 3t}{4t + 4} y, \qquad y(0) = 1$$

- Find $y(t)$ from $t = 0$ to $t = 1$, using Δt = 0.2, 0.1, 0.05
- Apply Richardson Extrapolation to the estimates with Δt = 0.2, 0.1 to obtain new estimates
- Plot the solutions of $y(t)$ vs t for each set of estimates on the same graph (4 curves on the same axis)

In-Class Exercise

- The solution to this ODE is

$$y(t) = (t+1)e^{-3t/4}$$

- Plot your approximate solution minus the actual solution on the same graph
- Examine the error at $t = \Delta t$ and $t = 1$ for each set of estimates

slido

Join at
slido.com
#FSD2

Audience Q&A Session