

# Lecture #8: Computational Methods for Plasma Physics Hines (1)

## I. Simulation of Single Particle Motion

### A. Particle Motion in Specified Fields $\underline{E}(x,t)$ and $\underline{B}(x,t)$

1. Lorentz Force Law 
$$\frac{d\underline{v}(t)}{dt} = \frac{q}{m} [\underline{E}(x,t) + \underline{v}(t) \times \underline{B}(x,t)]$$

2. Velocity 
$$\frac{d\underline{x}(t)}{dt} = \underline{v}(t)$$

3. This is a set of Ordinary differential equations (ODEs) which can be solved, if  $\underline{E}(x,t)$  &  $\underline{B}(x,t)$  are known, to advance  $\underline{x}(t)$  and  $\underline{v}(t)$  in time.

4. This requires initial conditions at  $t=0$ ,  $\underline{x}(t=0)$  and  $\underline{v}(t=0)$ .

5. In general, any ODE problem can be reduced to a set of first-order equations:

a. Ex: 
$$\frac{d^2y}{dx^2} + q(x) \frac{dy}{dx} = r(x)$$

$$\Rightarrow \begin{cases} \frac{dy}{dx} = z(x) \\ \frac{dz}{dx} = r(x) - q(x)z(x) \end{cases} \quad \leftarrow \text{Define new variable } z(x)$$

b. The general form, then, is for a set of  $N$  ~~coupled~~ <sup>coupled</sup> first-order differential equations for functions

$$y_i, \quad i=1, 2, \dots, N$$

where

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, y_2, \dots, y_N)$$

and the functions  $f_i$  are known.

c. Boundary values on functions  $y_i$  are required to specify solution fully.

d. Plasma simulation usually requires  $y_i(t=0)$  as initial conditions.

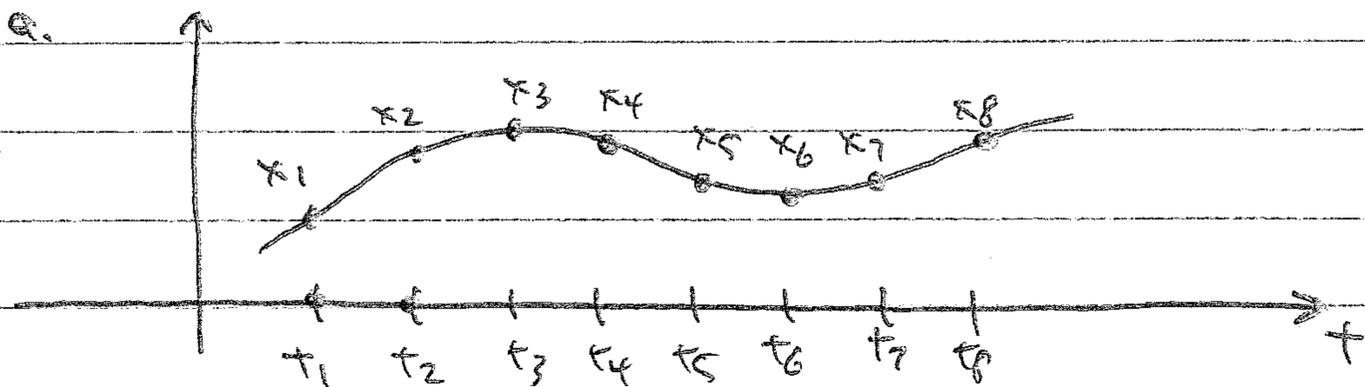
## Lecture #8 (Continued)

Answers ②

L. (Continued)

### B. Euler's Method:

1. To solve an equation numerically, we must represent it discretely. Take  $\frac{dx}{dt} = v$



b. Take  $t_j \equiv j \Delta t$  and  $x(t_j) \equiv x_j$  [and  $v_j \equiv v(t_j)$ ]

2. Discretize

$$\frac{dx}{dt} = \frac{x_{j+1} - x_j}{\Delta t} = v_j$$

a. Thus

$$x_{j+1} = x_j + \Delta t v_j$$

### 3. Order of Convergence:

a. Take  $\frac{dy}{dt} \Big|_{t_j} = \frac{y_{j+1} - y_j}{\Delta t} \Rightarrow y_{j+1} = y_j + \Delta t \frac{dy}{dt} \Big|_{t_j}$

b. The true solution  $y(t_{j+1}) = y(t_j) + \frac{\Delta t}{1!} \frac{dy}{dt} \Big|_{t_j} + \frac{\Delta t^2}{2!} \frac{d^2y}{dt^2} \Big|_{t_j} + \frac{\Delta t^3}{3!} \frac{d^3y}{dt^3} \Big|_{t_j} + \dots$

c. Error =  $y(t_{j+1}) - y_{j+1} = + \frac{\Delta t^2}{2!} \frac{d^2y}{dt^2} \Big|_{t_j} + \dots = \mathcal{O}(\Delta t^2)$

d. DEFINE: A method is order  $n$  if error scales as  $\mathcal{O}(\Delta t)^{n+1}$ .

Thus, Euler's Method is first order.

### C. Higher Order Methods

1. Can we conserve a higher order method for  $\frac{dy}{dt}$ ?

## Lecture #8 (Continued)

Times ③

I.C. (Continued)

2. Take  $\left. \frac{dy}{dt} \right|_{t_j} = \frac{y_{j+1} - y_{j-1}}{2\Delta t}$

3. In this case,  $y_{j+1} = y_{j-1} + 2\Delta t \left. \frac{dy}{dt} \right|_{t_j}$

a. ~~Since~~ Take  $y_{j+1} = y(t_j) + \Delta t \left. \frac{dy}{dt} \right|_{t_j} + \frac{\Delta t^2}{2!} \left. \frac{d^2y}{dt^2} \right|_{t_j} - \frac{\Delta t^3}{3!} \left. \frac{d^3y}{dt^3} \right|_{t_j} + \dots$

b. Thus  $y(t_{j+1}) - y_{j+1} =$

$$y_j + \Delta t \left. \frac{dy}{dt} \right|_{t_j} + \frac{\Delta t^2}{2} \left. \frac{d^2y}{dt^2} \right|_{t_j} + \frac{\Delta t^3}{3!} \left. \frac{d^3y}{dt^3} \right|_{t_j} - \left[ y_j - \Delta t \left. \frac{dy}{dt} \right|_{t_j} + \frac{\Delta t^2}{2!} \left. \frac{d^2y}{dt^2} \right|_{t_j} - \frac{\Delta t^3}{3!} \left. \frac{d^3y}{dt^3} \right|_{t_j} + \dots \right]$$

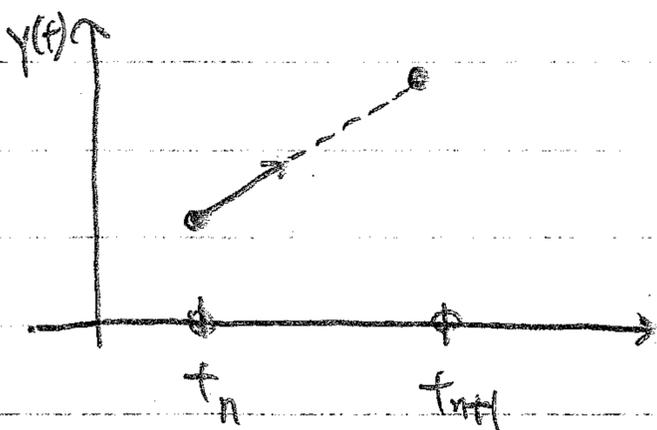
$$- 2\Delta t \left. \frac{dy}{dt} \right|_{t_j} = \frac{2\Delta t^3}{2!} \left. \frac{d^2y}{dt^2} \right|_{t_j} + \dots = O(\Delta t^3)$$

c. Thus, this centered time difference is ~~is~~ Second order

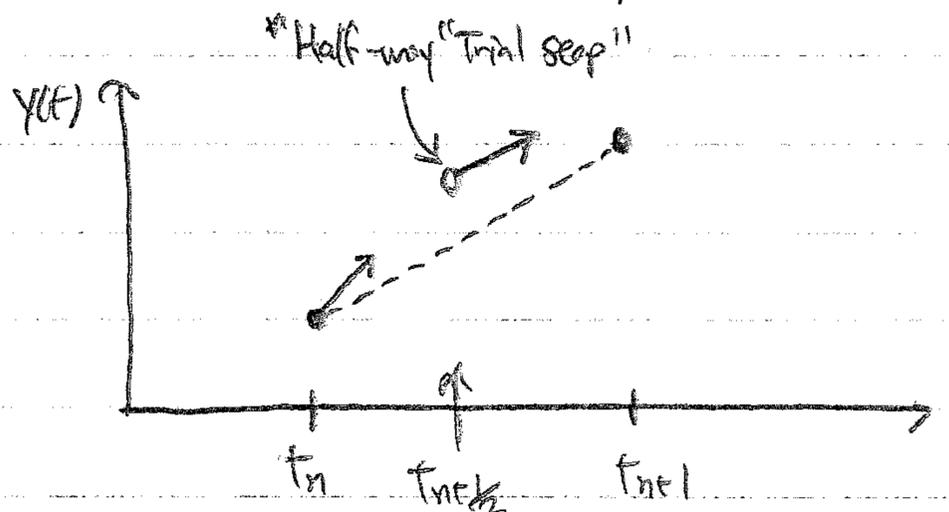
## D. Runge-Kutta Methods

1. Runge-Kutta Methods are a widely used workhorse of numerical methods.

2. In this ~~multi~~ multi step method, a "trial step" is taken to determine the derivative at the midpoint of the interval.



Euler's Method



2nd-order Runge-Kutta (or Midpoint) Method

$$\frac{dy}{dt} = f \Rightarrow$$

be written:

$$k_1 = \Delta t f(t_n, y_n)$$

$$k_2 = \Delta t f(t_{n+1/2}, y_n + \frac{1}{2}k_1)$$

$$y_{n+1} = y_n + k_2 + O(h^3)$$

4. Higher-order Methods of Runge-Kutta are widely used.

a. 4th order Runge-Kutta (RK4)

5. These methods can be used with a scheme that estimates error and adjust stepsize  $\Delta t$  to maintain a specified error.

a. RK45 is an Adaptive Stepsize Algorithm:

i) Takes one step of  $2\Delta t$  using RK4  
and two steps of  $\Delta t$  " RK4

ii) Estimates error from the difference

iii) Can use <sup>both</sup> solutions to eliminate  $O(\Delta t^5)$  terms,  
yielding a  $O(\Delta t^6)$  Fifth order method.

b. These schemes can be very efficient for smooth solutions,  
yet maintain small errors for stiff (difficult) problems.

## II. Introduction to Matlab

A. 1. Matlab is installed on all computers in 201 VAN.

2. Matlab has excellent online help

3. See handout by Kristian Sandberg "Introduction to Matlab"

for some basics, such as variables, math operations, and plotting.

4. The <sup>course</sup> homepage has Matlab m-files needed for the homeworks

## B. Variables (Matrices)

1. The basic variables of Matlab are matrices

a.  $A = [1 \ 2 \ 3 ; 4 \ 5 \ 6]$

$$\Rightarrow A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

b. Row Column format, so  $A(4,12)$  has 4 rows and 12 columns.

2. Vector Math can be used

$$y(:, i) = y(:, i-1) + \Delta t * dy/dt(:, i-1)$$

3. A semi-colon after a line suppress output  $\Rightarrow y = \frac{b}{a};$

4. Matrix transpose is an apostrophe:

$$x = \text{zeros}(3, 1)$$

$$\Rightarrow x = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$y = x'$$

$$\Rightarrow y = (0 \ 0 \ 0)$$

5. Matrix size can be queried ~~Q~~  $Q = \text{zeros}(14, 64)$

a.  $\text{size}(Q) = [14 \ 64]$

b.  $\text{size}(Q, 2) = 64$

## II, Continued

C. Plotting: (see handout)

D. Example: Euler back fishing program

1. Incorporate function call, where ~~function~~ m-file  
lon.m contains function.
2. Modify to produce Leapfrog Method.

E. Built-in Matlab Functions:

1. Matlab help describes all functions in detail.
2. ode45: Explicit Runge-Kutta (4,5) Adaptive Method.

$[T, Y] = \text{ode45}(\text{funhandle}, \text{tspan}, y_0)$

a. INPUT: 1) funhandle: is a function handle for passing in a specified function  $y' = f(t, y)$ .

funhandle = @f

2) tspan: specifies interval of integration  $[t_0, t_f]$

3) y0: vector of initial conditions

b. OUTPUT: 1) T column vector of time points  
2) Y solution array: Each row is vector of corresponding time row.

3. This built-in function make numerical work easy & fast