I. Simulation of Single Particle Motion

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

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

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

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

4. This requires <u>initial conditions</u> at $t=0$, $\underset{\sim}{x}(t=0)$ and $\underset{\sim}{v}(t=0)$.

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

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

$$\Rightarrow \begin{cases} \frac{dy}{dx} = z(x) & \longleftarrow \text{ define new variable } z(x) \\ \frac{dz}{dx} = r(x) - q(x)z(x) \end{cases}$$

b. The general form, then, is for a set of N coupled first-order differential equations for functions

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

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

and the functions f_i are known.

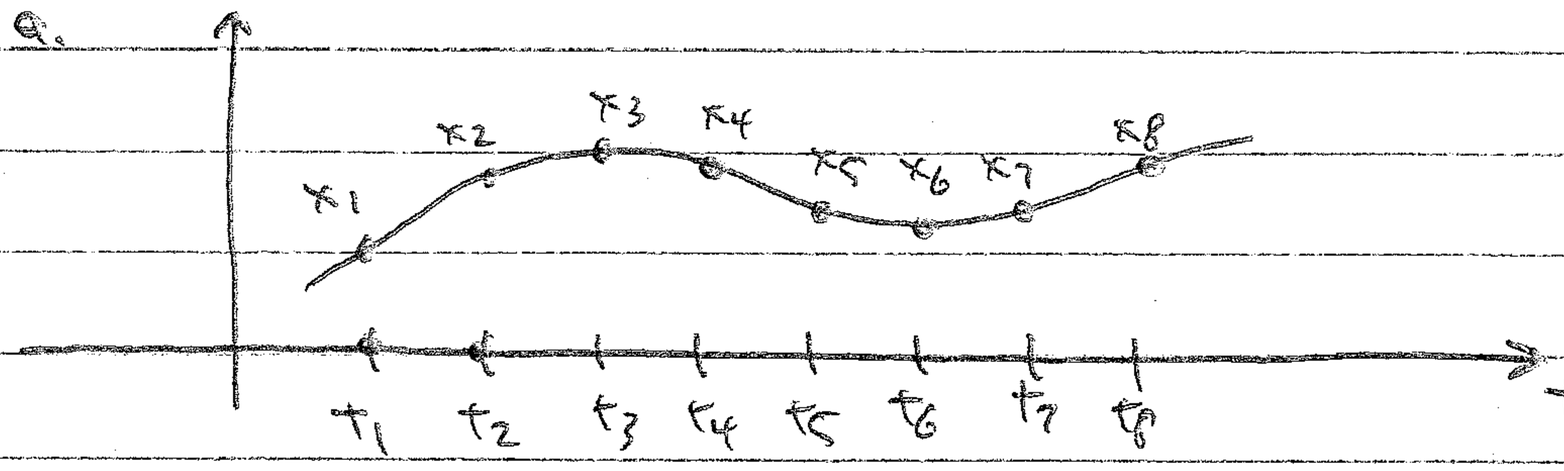
c. <u>Boundary Values</u> on functions y_i are required to specify solution fully.

d. Plasma Simulation usually requires $y_i(t=0)$ as <u>initial conditions</u>.

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

a. Thus $\boxed{x_{j+1} = x_j + \Delta t\, v_j}$

3. Order of Convergence:

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

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

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

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 construct a higher order method for $\frac{dy}{dt}$?

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

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

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

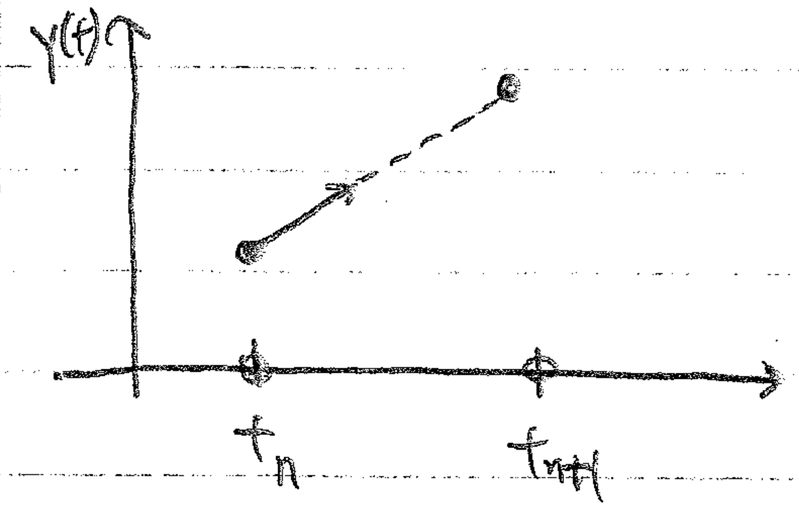
$2\Delta t \left.\frac{dy}{dt}\right|_j = \frac{2\Delta t^3}{3!}\left.\frac{d^3y}{dt^3}\right|_j + \ldots = O(\Delta t^3)$

c. Thus, this Centered Time difference is Second order

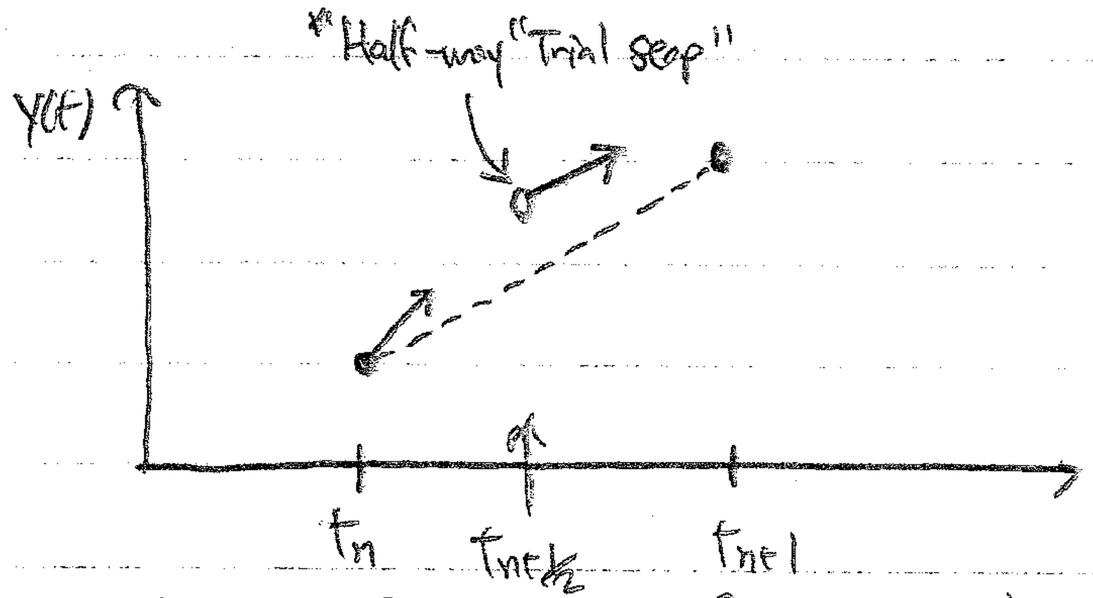
D. Runge-Kutta Methods

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

2. In this multistep 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

3. The Method can be written:

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

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

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

$y_{n+1} = y_n + k_2 + O(\Delta t)^3$

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 Δt to maintain a specified error.
   a. RK45 is an Adaptive Stepsize Algorithm:
      i) Takes one step at $2\Delta t$ using RK4
         and two steps of Δt " RK4
      ii) Estimates error from the difference
      iii) Can use both solutions to eliminate $\mathcal{O}(\Delta t^5)$ terms,
         yielding a $\mathcal{O}(\Delta t^6)$ 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 computer 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 course homepage has Matlab m-files needed for the homework.

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

$$Y(:,i) = Y(:,i-1) + \Delta t * dydt(:,i-1)$$

3. A semi-colon after a line suppress output => Y = b/a;
4. Matrix transpose is an apostrophe:

X = zeros(3,1)

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

Y = X'

$$\Rightarrow Y = (0 \quad 0 \quad 0)$$

5. Matrix size can be queried Q = zeros(14,64)
a. size(Q) = [14 64]
b. size(Q,2) = 64

Lecture #8 (Continued) Hwes6

II. Continued)

C. Phasing: (See handout)

D. Example: Euler Parabola Rolling Program

1. Incorporate function call, where lor.m contains function. m-file

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] = ode45 (funchandle, tspan, y0)

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

funchandle = @f

2) tspan: specifies interval of integration [t0, tf]

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

```
% Simple Euler particle pushing program
tspan= [0. 2.*pi*1.]';
nn=40;

dt=(tspan(end)-tspan(1))/nn;
t=tspan(1):dt:tspan(end); %Calculate row vector of times

%Initial Conditions [ x  y vx vy ]'
y0= [0. 0. 1. 0.]'

%Parameters
B= 1;
qoverm=1.;

%Set solution as matrix of zeros
y=zeros(size(y0,1),size(t,2));

%Copy initial conditions y0  to first timestep of solution y
y(:,1)=y0(:);

%Loop to advance each timestep
for i = 2:nn+1
  dt=t(i)-t(i-1);
  y(1,i)=y(1,i-1) + dt*y(3,i-1);
  y(2,i)=y(2,i-1) + dt*y(4,i-1);
  y(3,i)=y(3,i-1) + dt*(qoverm*B*y(4,i-1));
  y(4,i)=y(4,i-1) + dt*(-qoverm*B*y(3,i-1));
end

%Calculate error due to numerical errors
terr=( (y(1,1)-y(1,end))^2. + (y(1,2)-y(2,end))^2.)^0.5

%Plot trajectory of particle (x,y)
plot(y(1,:),y(2,:))
xlabel('x')
ylabel('y')
title('Trajectory of Larmor Motion')
```