L18: November 5, 2014 ME 564, Fall 2014

Overview of Topics

① Runge Kutta Integration

(a) 2^{nd} order

(b) 4^{th} order

② Chaotic Example: Lorenz Equation

(a) Slow: "for" loops

(b) Fast: vectorized

Second-Order Runge-Kutta ('ode 23')

$$\boxed{y_{k+1} = y_k + \Delta t\, f\left(t_k + \frac{\Delta t}{2},\, y_k + \frac{\Delta t}{2} f(t_k, y_k)\right)}$$

OR

$$y_{k+1} = y_k + \Delta t\, f_2$$

where
$$f_1 = f(t_k, y_k)$$
$$f_2 = f\left(t_k + \frac{\Delta t}{2},\, y_k + \frac{\Delta t}{2} f_1\right)$$

Fourth-Order Runge-Kutta ('ode 45')

$$y_{k+1} = y_k + \frac{\Delta t}{6}\left[f_1 + 2f_2 + 2f_3 + f_4\right]$$

where
$$f_1 = f(t_k, y_k)$$
$$f_2 = f\left(t_k + \frac{\Delta t}{2},\, y_k + \frac{\Delta t}{2} f_1\right)$$
$$f_3 = f\left(t_k + \frac{\Delta t}{2},\, y_k + \frac{\Delta t}{2} f_2\right)$$
$$f_4 = f\left(t_k + \Delta t,\, y_k + \Delta t\, f_3\right)$$

local error : $\mathcal{O}(\Delta t^5)$

global error : $\mathcal{O}(\Delta t^4)$.

$\dot{y} = f(t, y)$

Last time we saw the 4th order Runge-Kutta integrator (RK4):

RK4
('ode45' based on this)

$$y_{k+1} = y_k + \frac{\Delta t}{6}\left(f_1 + 2f_2 + 2f_3 + f_4\right)$$

$$f_1 = f(t_k, y_k)$$

$$f_2 = f\left(t_k + \Delta t/2,\ y_k + (\Delta t/2) f_1\right)$$ [evaluate vector field after taking half Euler step using f_1]

$$f_3 = f\left(t_k + \Delta t/2,\ y_k + (\Delta t/2) f_2\right)$$ [evaluate VF using half Euler step w/ f_2]

$$f_4 = f\left(t_k + \Delta t,\ y_k + \Delta t\, f_3\right)$$ [take full Euler step w/ f_3].

- Very accurate $(\mathcal{O}(\Delta t^5))$ local accuracy per time step.
- Uses four evaluations of $f(t,y)$, which is typically expensive.

[More evaluations of $f(t,y)$ per time step than Forward Euler, <u>but</u>, many fewer Δt's required for same accuracy!]

Example: Lorentz Equation (1963, atmospheric convection model)

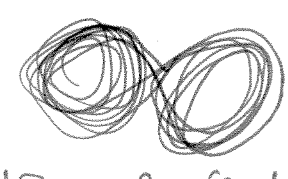
$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned}$$

parameters may lead to 'chaos'

$\sigma = 10$

$\beta = 8/3$ $\Longrightarrow$

$\rho = 28$



'Butterfly effect'

Last time: we wrote codes: lorenz.m
rk4singlestep.m

Now, we will investigate how to integrate <u>many</u> initial conditions (trajectories) efficiently

<u>Idea 1</u>: use 'for' loop and integrate each particle one-at-a-time.

* very slow in Matlab! Why?

Matlab scripts are not compiled, so every iteration of the for loop, it re-translates your commands into machine-code instructions.

<u>Idea 2</u>: * Alternative is to <u>vectorize</u> computation, so all particles are passed through vector field at the same time.

* <u>much faster</u> (100-1000x for our example) because matrix operations in Matlab are built on LAPACK, a highly optimized, compiled Fortran package.

```
function dy = lorenz(t,y,sigma,beta,rho)
% y is a three dimensional state-vector
dy = [
sigma*(y(2)-y(1));
y(1)*(rho-y(3))-y(2);
y(1)*y(2)-beta*y(3);
];
```

```
function yout = rk4singlestep(fun,dt,t0,y0)

f1 = fun(t0,y0);
f2 = fun(t0+dt/2,y0+(dt/2)*f1);
f3 = fun(t0+dt/2,y0+(dt/2)*f2);
f4 = fun(t0+dt,y0+dt*f3);

yout = y0 + (dt/6)*(f1+2*f2+2*f3+f4);
```

```
clear all

% Lorenz's parameters (chaotic)
sigma = 10;
beta = 8/3;
rho = 28;

% Initial condition
y0=[-8; 8; 27];

% Compute trajectory
dt =0.01;
tspan=[0:dt:4];

Y(:,1)=y0;
yin = y0;
for i=1:tspan(2)/dt
    time = i*dt;
    yout = rk4singlestep(@(t,y)lorenz(t,y,sigma,beta,rho),dt,time,yin);
    Y = [Y yout];
    yin = yout;
end
plot3(Y(1,:),Y(2,:),Y(3,:),'b')
hold on
[t,y] = ode45(@(t,y)lorenz(t,y,sigma,beta,rho),tspan,y0);
plot3(y(:,1),y(:,2),y(:,3),'r')
```

```
clear all

% Lorenz's parameters (chaotic)
sigma = 10;
beta = 8/3;
rho = 28;

% Initial condition - large cube of points
xvec = -20:2:20;
yvec = -20:2:20;
zvec = -20:2:20;
[x0,y0,z0] = meshgrid(xvec,yvec,zvec);
yIC(1,:,:,:) = x0;
yIC(2,:,:,:) = y0;
yIC(3,:,:,:) = z0;

plot3(yIC(1,:),yIC(2,:),yIC(3,:),'r.','LineWidth',2,'MarkerSize',4)
axis([-40 40 -40 40 -40 40])
view(20,40);
drawnow

%% Compute trajectory
dt =0.01;
duration = 4
tspan=[0,duration];
L = duration/dt;
yparticles = yIC;

% this code is slow because MATLAB is not compiled
% we use nested for loops to step through every single IC in the cube
% one at a time...
for step=1:L
    time = step*dt
    for i=1:length(xvec)
        for j=1:length(yvec)
            for k=1:length(zvec)
                yin = yparticles(:,i,j,k);
                yout = rk4singlestep(@(t,y)lorenz(t,y,sigma,beta,rho),dt,time,yin);
                yparticles(:,i,j,k) = yout;
            end
        end
    end
    plot3(yparticles(1,:),yparticles(2,:),yparticles(3,:),'r.','LineWidth',2,'MarkerSize',4)
    view(20,40);
    axis([-40 40 -40 40 -10 40])
    drawnow
end
```

```
function dy = lorenz3D(t,y,sigma,beta,rho)
% y is a three dimensional state-vector
dy = [
sigma*(y(2,:,:,:)-y(1,:,:,:));
y(1,:,:,:).*(rho-y(3,:,:,:))-y(2,:,:,:);
y(1,:,:,:).*y(2,:,:,:)-beta*y(3,:,:,:);
];
```

```
clear all

% Lorenz's parameters (chaotic)
sigma = 10;
beta = 8/3;
rho = 28;

% Initial condition 1 - Large cube of data
y0=[0 0 0];
xvec = -20:2:20;
yvec = -20:2:20;
zvec = -20:2:20;
% % Initial condition 2 - small cube around initial condition from last class
% y0=[-8; 8; 27];
% xvec = -1:.1:1;
% yvec = -1:.1:1;
% zvec = -1:.1:1;
% % Initial condition 3 - even smaller cube around initial condition
% y0=[-8; 8; 27];
% xvec = -.1:.01:.1;
% yvec = -.1:.01:.1;
% zvec = -.1:.01:.1;
[x0,y0,z0] = meshgrid(xvec+y0(1),yvec+y0(2),zvec+y0(3));
yIC(1,:,:,:) = x0;
yIC(2,:,:,:) = y0;
yIC(3,:,:,:) = z0;

plot3(yIC(1,:),yIC(2,:),yIC(3,:),'r.','LineWidth',2,'MarkerSize',10)
axis([-40 40 -40 40 -40 40])
view(20,40);
drawnow

%% Compute trajectory
dt =0.01;
duration = 4
tspan=[0,duration];
L = duration/dt;
yin = yIC;

for step = 1:L
    time = step*dt
    yout = rk4singlestep(@(t,y)lorenz3D(t,y,sigma,beta,rho),dt,time,yin);
    yin = yout;
    plot3(yout(1,:),yout(2,:),yout(3,:),'r.','LineWidth',2,'MarkerSize',10)
    view(20+360*step/L,40);
    axis([-40 40 -40 40 -10 40])
    drawnow
end
```