Numerical Solutions of Differential Equations:

Runge- Kutta Methods

Runge-Kutta Methods

Runge-Kutta methods are very popular because of their good efficiency; and are used in most computer programs for differential equations.

Runge-Kutta Methods

To convey some idea of how the Runge-Kutta is developed, let's look at the derivation of the **2nd order**. Two estimates

$$y_{n+1} = y_n + ak_1 + bk_2$$

$$k_1 = hf\left(x_n, y_n\right)$$

$$k_2 = hf\left(x_n + \alpha h, y_n + \beta k_1\right)$$

We will see what *a* , *b* , *α* and *β* mean….

Runge-Kutta Methods

The initial conditions are:

$$\frac{dy}{dx} = f(x, y) \qquad\qquad y(x_0) = y_0$$

Using the Taylor series expansion

$$g(x+h) \;=\; g(x) + hg'(x) + \frac{h^2}{2!}g''(x) + \cancel{\frac{h^3}{3!}g'''(x)} + \mathrm{L}$$

We can write:

$$y(x_{n+1}) = y(x_n) + h\frac{dy(x_n, y_n)}{dx} + \frac{h^2}{2!}\frac{d^2y(x_n, y_n)}{dx^2}$$

Runge-Kutta Methods

Expand the derivatives:

$$\frac{d^2y}{dx^2} = \frac{d}{dx}\left[\quad \frac{dy}{dx} \quad\right]$$

The Taylor series expansion becomes

$$y_{n+1} = y_n + hf + h^2\left[\frac{1}{2}\left(f_x + f_y f\right)\right]$$

Runge-Kutta Methods

According to Runge-Kutta methods

$$y_{n+1} = y_n + hf + h^2\left[\frac{1}{2}\left(f_x + f_y f\right)\right]$$ Is written as:

The definition of the function '*f*'

$$f\left(x_n + \alpha h, y_n + \beta hf\right) = f + \alpha h f_x + \beta hf \; f_y$$

Expand in the next step to get

$$y_{n+1} = y_n + ahf + bh\left(f + \alpha h f_x + \beta hf \; f_y\right)$$

$$= y_n + [a+b]hf + b\alpha h^2 f_x + b\beta h^2 f \; f_y$$

Runge-Kutta Methods

From the Runge-Kutta

$$y_{n+1} = y_n + [a+b]hf + b\alpha h^2 f_x + b\beta h^2 f \; f_y$$

Compare with the Taylor series

$$[a+b] = 1$$

$$\alpha b = \frac{1}{2}$$

4 unknowns

$$\beta b = \frac{1}{2}$$

Runge-Kutta Methods

The Taylor series coefficients (3 equations/4 unknowns)

$$[a+b]=1,\ \alpha b=\frac{1}{2},\quad \beta b=\frac{1}{2}$$

If you select “a” as

$$a=\frac{2}{3},\ b=\frac{1}{3}\rightarrow\ \alpha=\frac{3}{2},\quad \beta=\frac{3}{2}$$

If you select “a” as

$$a=\frac{1}{2}\ b=\frac{1}{2}\rightarrow\ \alpha=\beta=1$$

Runge-Kutta Methods

We started with:

$$y_{n+1} = y_n + ak_1 + bk_2$$

$$k_1 = hf\left(x_n, y_n\right)$$

$$k_2 = hf\left(x_n + \alpha h, y_n + \beta k_1\right)$$

a ,*b* , *α* and *β* are appropriate weights to be found

Using:

$$\boxed{a = \frac{1}{2} \quad b = \frac{1}{2}, \quad \alpha = \beta = 1}$$

$$k_1 = hf\left(x_i, y_i\right)$$

$$k_2 = hf\left(x_i + h, y_i + k_1\right)$$

$$y_{i+1} = y_i + \frac{1}{2}\left[k_1 + k_2\right]$$

2nd Order Runge-Kutta Method or **Modified Euler's Method**

Runge-Kutta Methods

What if we choose:

the values as $a = \frac{2}{3},\ b = \frac{1}{3},\ \alpha = \frac{3}{2},\ \beta = \frac{3}{2}$

$$y_{i+1} = y_i + a k_1 + b k_2$$
$$k_1 = hf\left(x_i, y_i\right)$$
$$k_2 = hf\left(x_i + \alpha h, y_i + \beta k_1\right)$$

2nd Order Runge-Kutta Method or Heun's Method

Runge-Kutta Methods

The Runge-Kutta methods are higher order approximation of the basic forward integration. These methods provide solutions which are comparable in accuracy to Taylor series solution in which higher order derivatives are retained.

It should be noted that the equations are not need to be linear.

Runge-Kutta Methods

Method	**Equations**
Euler (Error of the order h^2)	$\Delta y = k_1$ $k_1 = h[f(x, y)]$
Modified Euler (Error of the order h^3)	$\Delta y = \frac{1}{2}[k_1 + k_2]$ $k_1 = h[f(x, y)]$ $k_2 = h[f(x+h, y+k_1)]$
Heun (Error of the order h^4)	$\Delta y = \frac{1}{4}[k_1 + 3k_3]$ $k_1 = \Delta h[f(x, y)]$ $k_2 = h\left[f\left(x+\frac{1}{3}h, y+\frac{1}{3}k_1\right)\right]$ $k_3 = h\left[f\left(x+\frac{2}{3}h, y+\frac{2}{3}k_2\right)\right]$
4th order Runge Kutta (Error of the order h^5)	$\Delta y = \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$ $k_1 = h[f(x, y)]$ $k_2 = h\left[f\left(x+\frac{1}{2}h, y+\frac{1}{2}k_1\right)\right]$ $k_3 = h\left[f\left(x+\frac{1}{2}h, y+\frac{1}{2}k_2\right)\right]$ $k_4 = h[f(x+h, y+k_3)]$

Runge-Kutta Methods

This is a fourth order function that solves an initial value problems using a four step program to get an estimate of the Taylor series through the fourth order.

This will result in a local error of $O(h^5)$ and a global error of $O(h^4)$

Runge-Kutta Methods

The general form of the equations for the 4th Order method are:

$$\Delta y = \frac{1}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right]$$

$$k_1 = h\left[f(x, y)\right]$$

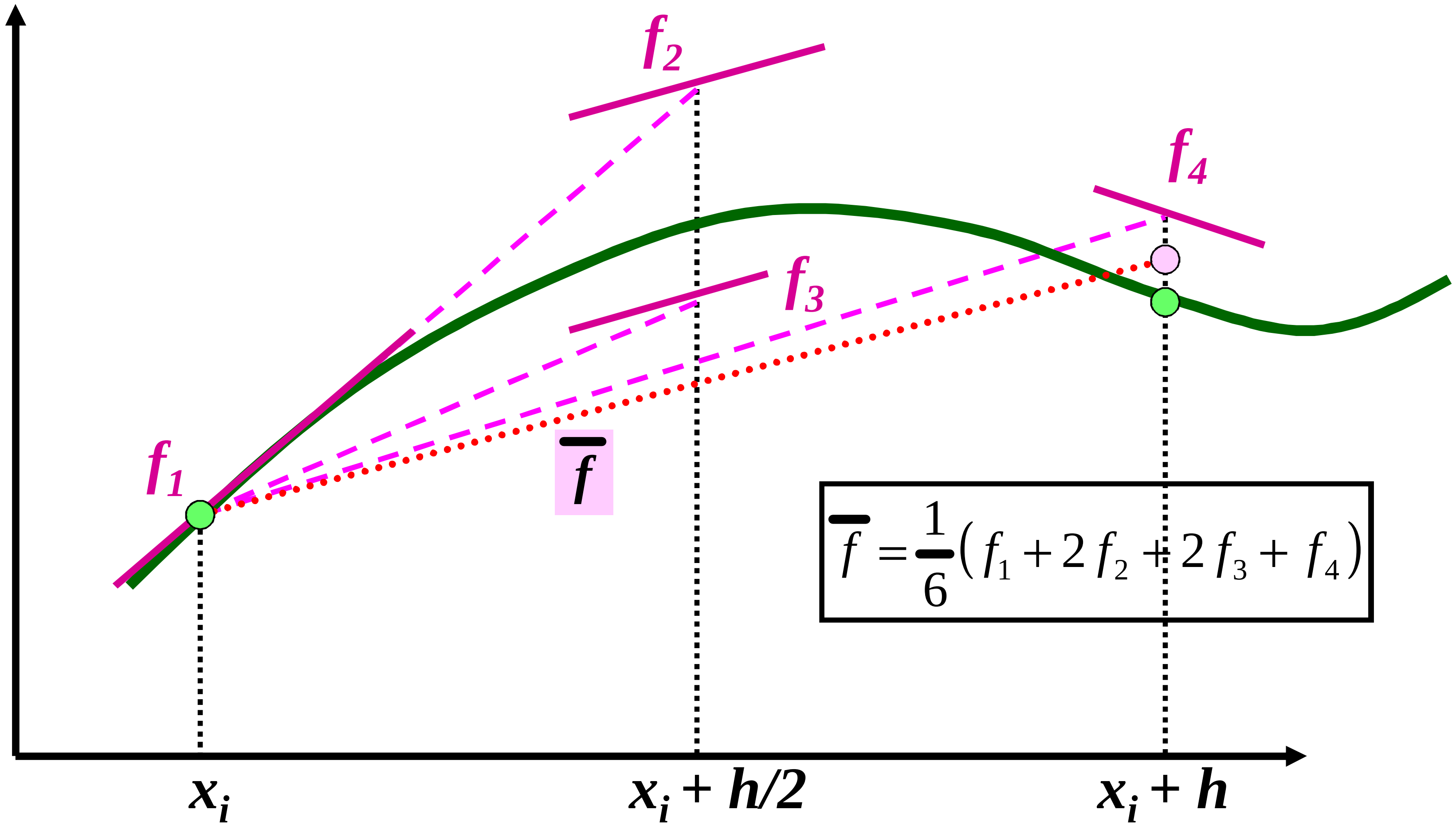
$$k_2 = h\left[f\left(x + \frac{1}{2}h, y + \frac{1}{2}k_1\right)\right]$$

$$k_3 = h\left[f\left(x + \frac{1}{2}h, y + \frac{1}{2}k_2\right)\right]$$

$$k_4 = h\left[f(x + h, y + k_3)\right]$$

Runge-Kutta Methods

Graphical Representation of the 4rth order method:



$$\bar{f} = \frac{1}{6}\left(f_1 + 2f_2 + 2f_3 + f_4\right)$$

Runge-Kutta Methods

Higher order differential equations can be treated as if they were a set of first-order equations. Runge-Kutta type forward integration solutions can be obtain. A more direct solution can be obtained by repeating the whole process used in first-order cases.

Runge-Kutta Methods

The general form of the equations for higher order differential equations are:

$$y'' = f(x, y, y')$$

$$k_1 = h^2\left[f(x, y, y')\right]$$

$$k_2 = \left(\frac{h^2}{2}\right)\left[f\left(x + \frac{1}{2}h, y + \frac{h}{2}y' + \frac{1}{4}k_1, y' + \frac{1}{h}k_1\right)\right]$$

$$k_3 = \left(\frac{h^2}{2}\right)\left[f\left(x + \frac{1}{2}h, y + \frac{h}{2}y' + \frac{1}{4}k_2, y' + \frac{1}{h}k_2\right)\right]$$

$$k_4 = \left(\frac{h^2}{2}\right)\left[f\left(x + h, y + hy' + k_3, y' + \frac{2}{h}k_3\right)\right]$$

Runge-Kutta Methods

The step sizes are:

$$\Delta y = \frac{1}{3}\left[k_1 + k_2 + k_3\right]$$

$$\Delta y' = \frac{1}{3h}\left[k_1 + 2k_2 + 2k_3 + k_4\right]$$

The next step would be:

$$y(x+h) = y(x) + h\, y'(x) + \Delta y$$

$$y'(x+h) = y'(x) + \Delta y'$$

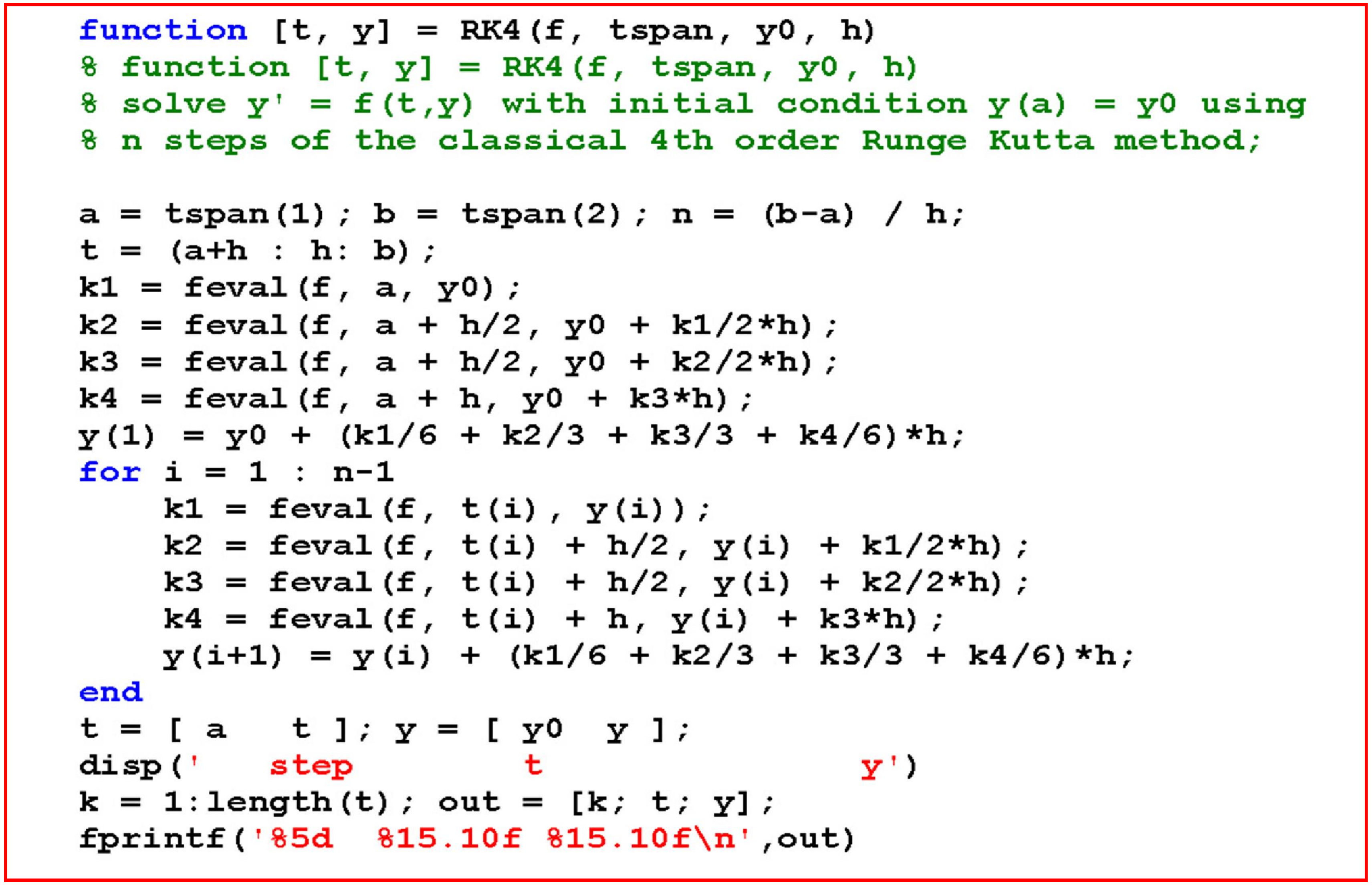
Runge-Kutta Methods

Up until this point we have dealt with:

- Euler Method
- Modified Euler and Heun's Method
- Runge-Kutta Methods

These methods are called single step methods, because they use only the information from the previous step.

Runge-Kutta Methods



```
function [t, y] = RK4(f, tspan, y0, h)
% function [t, y] = RK4(f, tspan, y0, h)
% solve y' = f(t,y) with initial condition y(a) = y0 using
% n steps of the classical 4th order Runge Kutta method;

a = tspan(1); b = tspan(2); n = (b-a) / h;
t = (a+h : h: b);
k1 = feval(f, a, y0);
k2 = feval(f, a + h/2, y0 + k1/2*h);
k3 = feval(f, a + h/2, y0 + k2/2*h);
k4 = feval(f, a + h, y0 + k3*h);
y(1) = y0 + (k1/6 + k2/3 + k3/3 + k4/6)*h;
for i = 1 : n-1
    k1 = feval(f, t(i), y(i));
    k2 = feval(f, t(i) + h/2, y(i) + k1/2*h);
    k3 = feval(f, t(i) + h/2, y(i) + k2/2*h);
    k4 = feval(f, t(i) + h, y(i) + k3*h);
    y(i+1) = y(i) + (k1/6 + k2/3 + k3/3 + k4/6)*h;
end
t = [ a   t ]; y = [ y0  y ];
disp('   step          t                   y')
k = 1:length(t); out = [k; t; y];
fprintf('%5d  %15.10f %15.10f\n',out)
```

Single Step Method

- These methods allow us to vary the step size.
- Use only one initial value.
- After each step is completed the past step is “forgotten: We do not use this information.

MATLAB uses 2nd and 4th order methods – “*ode23*” and “*ode45*” solvers.

Matlab’s ode45

- ode45 is a variable step solver and is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair.

- ode45 needs only the solution at the immediately preceding point to compute the next value.

- Dormand, J. R. and P. J. Prince, "A family of embedded Runge-Kutta formulae," *J. Comp. Appl. Math.*, Vol. 6, 1980, pp 19-26.

Matlab's ode45

$$y'(t) = \alpha y(t) - \gamma y(t)^2, \quad \text{i.c. } y(0) = 10$$

```
clear;
tspan=[0,8]; % set time interval
y0=10;       % set initial condition
% fyt evaluates r.h.s. of the ode
[t,y]=ode45('fyt',tspan,y0);
plot(t,y)
[t,y]        % print out t and y(t)
```

```
function yprime = fyt(t,y)
a=2; g=0.0001;
yprime = a*y-g*y^2;
```

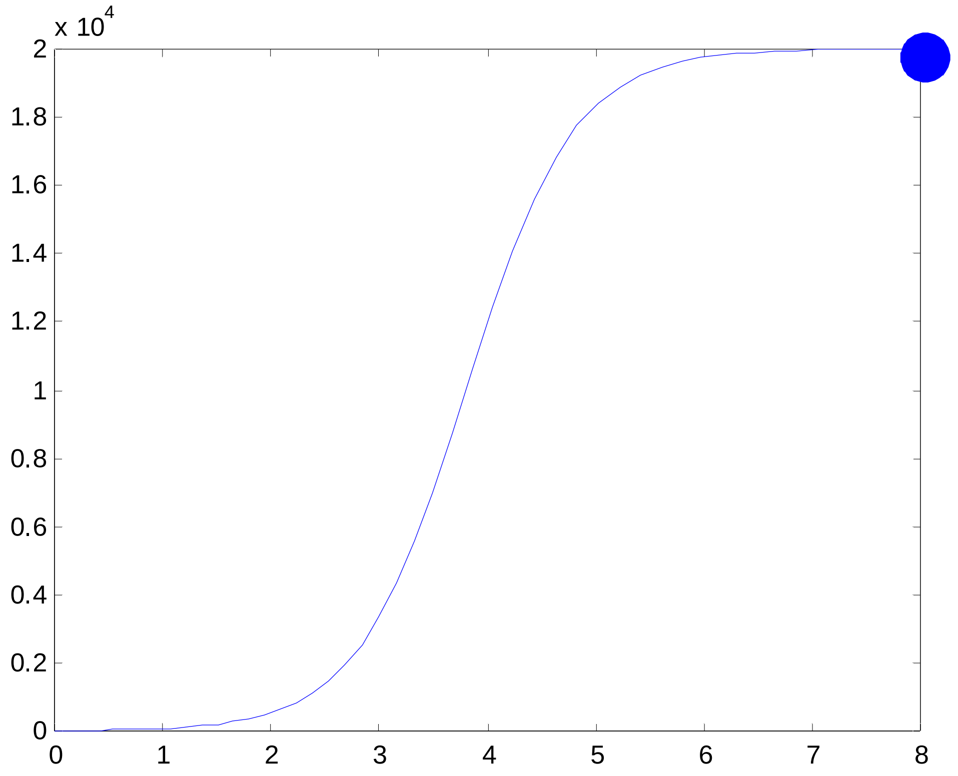
Matlab's ode45

$$y'(t) = \alpha y(t) - \gamma y(t)^2, \quad \text{i.c. } y(0) = 10$$

```
clear;
tspan=[0,8]; % set time interval
y0=10;       % set initial condition
% fyt evaluates r.h.s. of the ode
[t,y]=ode45('fyt',tspan,y0);
plot(t,y)
[t,y]        % print out t and y(t)
```

```
function yprime = fyt(t,y)
a=2; g=0.0001;
yprime = a*y-g*y^2;
```

Matlab's Plot



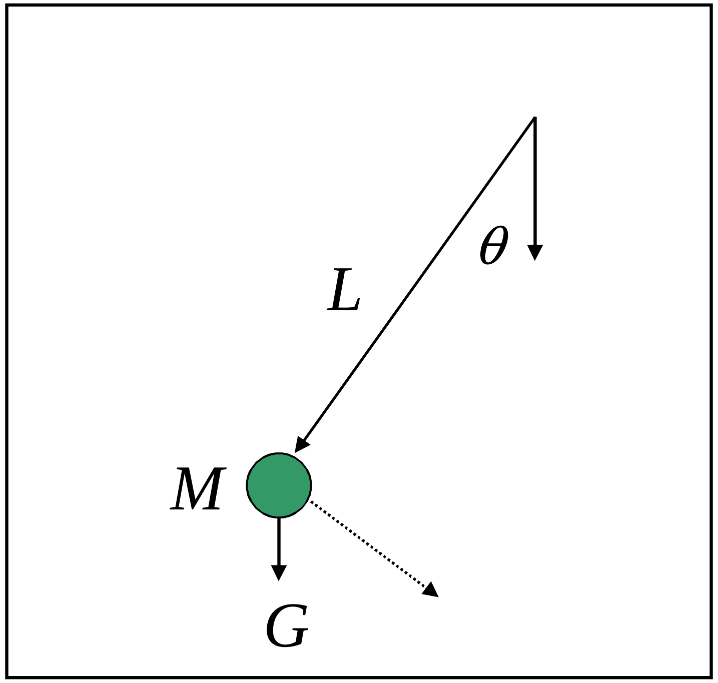
$y(8)=19,950$

Steady state solution
as t à infinity
is $\alpha/\gamma=20,000$.

Simple Pendulum

$$ML\frac{d^2\theta}{dt^2} = -MG\sin\theta$$

$$\frac{d^2\theta}{dt^2} = -\frac{G}{L}\sin\theta$$



Second order non-linear ODE.
Non-linear because of $\sin\theta$.

To **solve analytically**, make the approximation $\sin\theta \sim \theta$. This makes the ODE linear for "small amplitude oscillations".

Solve Numerically

$$\theta'' = -G/L \sin\theta$$

$$\theta(0) = \pi/3$$

$$\theta'(0) = 0$$

$$\theta'(t) = \theta_1(t)$$

$$\theta_1'(t) = -G/L \sin\theta(t)$$

$$\theta(0) = \pi/3$$

$$\theta_1(0) = 0$$

Convert 2nd order ODE to standard form

Matlab Script: Non-linear Pendulum

```
clear;
tspan=[0,2*pi]; % set time interval
th_0=[pi/3,0];  % set initial conditions
% pend evaluates r.h.s. of the ode
[t,th]=ode45('pend',tspan,th_0);
plot(t,th(:,1))
```

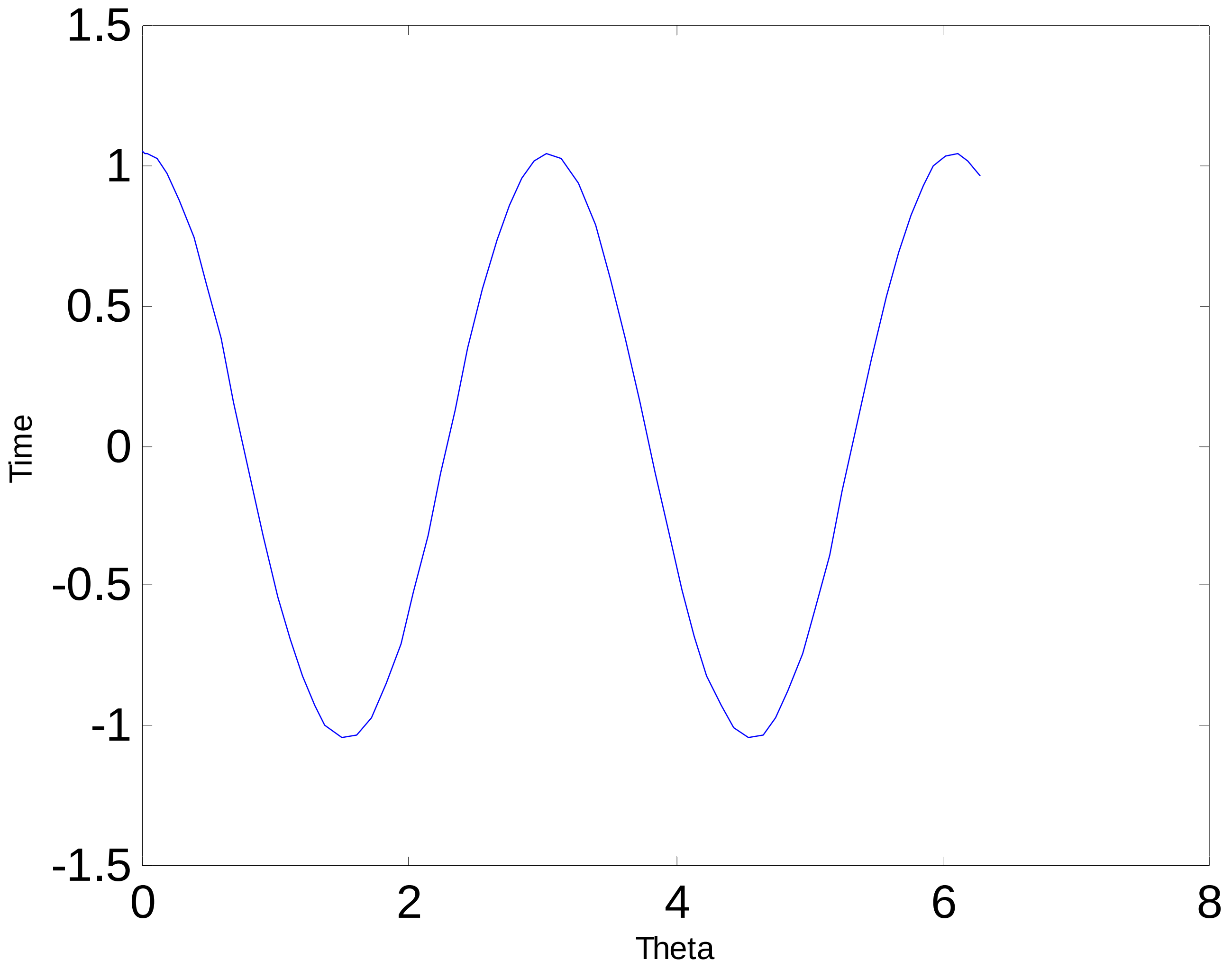
```
function th_prime = pend(t,th)
G=9.8; L=2;                  % set constants
z=th(1);                     % get theta
z1=th(2);                    % get theta1
zprime=z1;                   % compute theta'
z1prime=-G/L*sin(z) %compute theta1'
th_prime = [zprime ; z1prime];
```

Matlab Script-Cont.

```
clear;
tspan=[0,2*pi]; % set time interval
th_0=[pi/3,0];  % set initial conditions
% pend evaluates r.h.s. of the ode
[t,th]=ode45('pend',tspan,th_0);
plot(t,th(:,1))
```

```
function th_prime = pend(t,th)
G=9.8; L=2;                 % set constants
z=th(1);                    % get theta
z1=th(2);                   % get theta1
zprime=z1;                  % compute theta’
z1prime=-G/L*sin(z) %compute theta1’
th_prime = [zprime ; z1prime];
```

Matlab’s Plot



A predator-prey model

$$r(t) = \text{rabbit population, } f(t) = \text{fox population}$$

$$\frac{dr(t)}{dt} = \alpha r(t) - \beta r(t) f(t), \quad r(0) = 400$$

Rate of change of rabbits

Birth-natural death rate term

Foxes eat rabbits-death rate due to foxes

$$\frac{df(t)}{dt} = -\gamma f(t) + \delta r(t) f(t), \quad f(0) = 16$$

Rate of change of foxes

Compete for food-no rabbits

More rabbits, more food so more foxes

Seek Equilibrium Solutions for Rabbit and Fox Populations

$$0 = \frac{dr}{dt} = \alpha r - \beta rf = r(\alpha - \beta f)$$

$$0 = \frac{df}{dt} = -\gamma f + \delta rf = f(-\gamma + \delta r)$$

$$\alpha = 1.6,\ \beta = 0.11,\ \delta = 0.01,\ \gamma = 3.7$$

$$\boxed{\begin{aligned} r(t) &= r^* = {}^{\gamma}\!/_{\delta} = {}^{3.7}\!/_{0.01} = 370 \\ f(t) &= f^* = {}^{\alpha}\!/_{\beta} = {}^{1.6}\!/_{0.11} = 14.5 \end{aligned}}$$

Matlab Script

```
clear;
tspan = [0, 1500];  % solution time span
f0 = 16;            % number of foxes at t=0
r0 = 400;           % number of rabbits at t=0
yaxes = [-20,480];  % for plotting
z0 = [r0, f0];      % i.c.'s for r(t) and f(t)
[t,z] = ode45('nonlin_rf', tspan, z0);
r = z(:,1);         % extract r(t)
f = z(:,2);         % extract f(t)
figure        % plot over the entire time span
plot(t,r,'b', t,10*f,'k')
axis([tspan, yaxes]);
title('entire time span');
```

Matlab Script-cont

```
figure % plot first 10% of tspan
plot(t,r,'b', t,10*f,'k')
axis([tspan(2)*0.00,  tspan(2)*0.10, yaxes]);
title('first 10% of time span');
figure % plot last 10% of tspan
plot(t,r, 'b', t,10*f,'k')
axis([tspan(2)*0.90,  tspan(2)*1.00, yaxes]);
title('last 10% of time span');
```

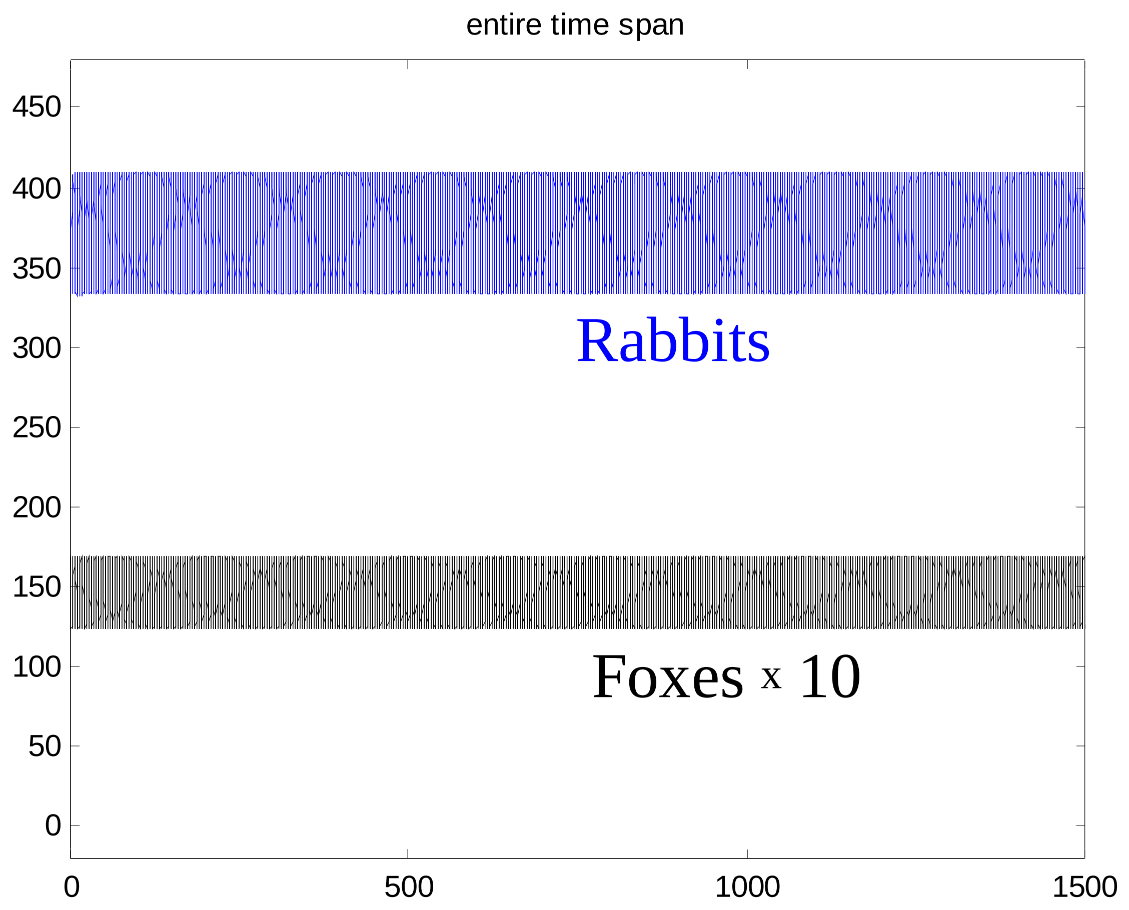
Matlab Script-cont

```matlab
figure % plot last 1% of tspan
plot(t,r, 'b', t,10*f,'k')
axis([tspan(2)*0.99,  tspan(2)*1.00, yaxes]);
title('last 1% of time span');
figure % plot first 1% of tspan
plot(t,r, 'b', t,10*f,'k')
axis([tspan(2)*0.00,  tspan(2)*0.01, yaxes]);
title('first 1% of time span');
```

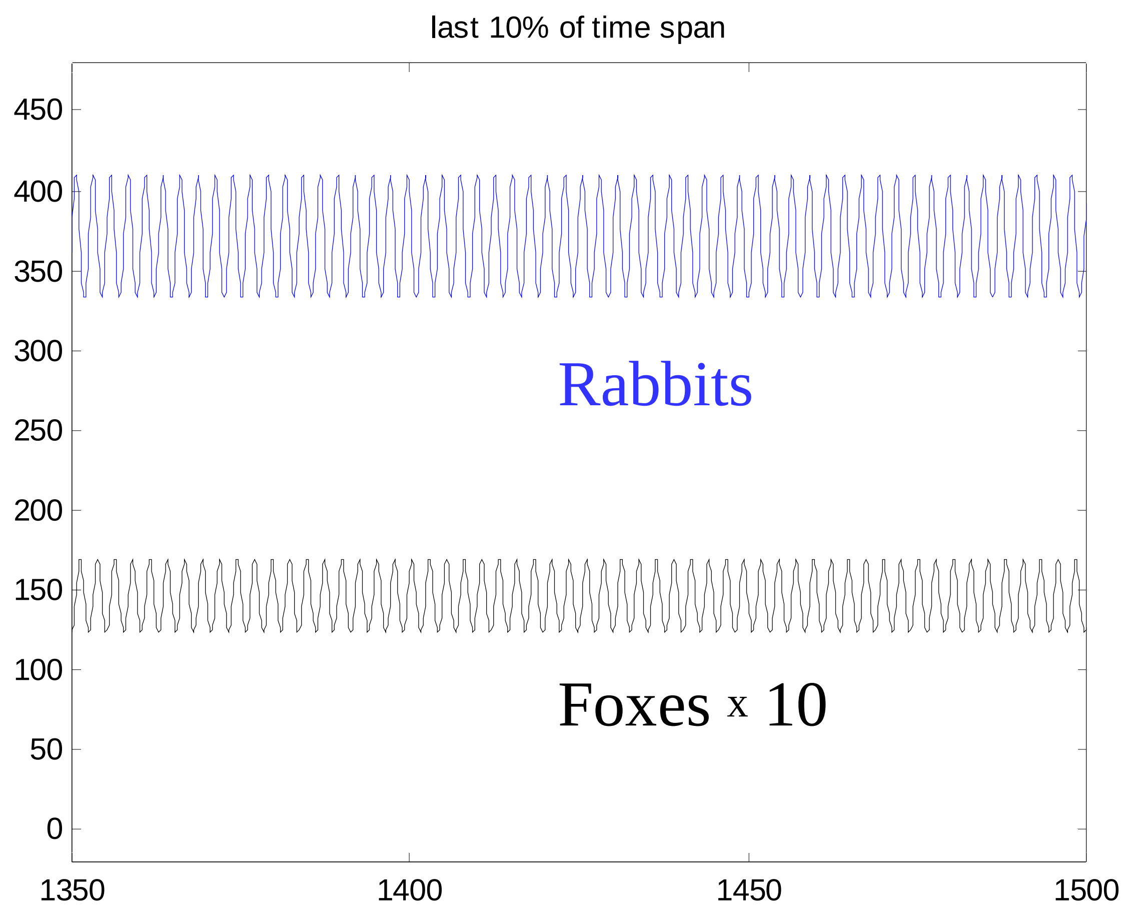
Script to evaluate the RHS of the PDE

```
function zprime = nonlin_rf(t,z)
% evaluate the derivatives of r and f
r = z(1);    % extract r(t)
f = z(2);    % extract f(t)
alpha = 1.6;
beta = 0.11;
gamma = 3.7;
delta = 0.01;
rprime = alpha*r - beta*r*f;
fprime = -gamma*f + delta*r*f;
zprime = [rprime; fprime];
```

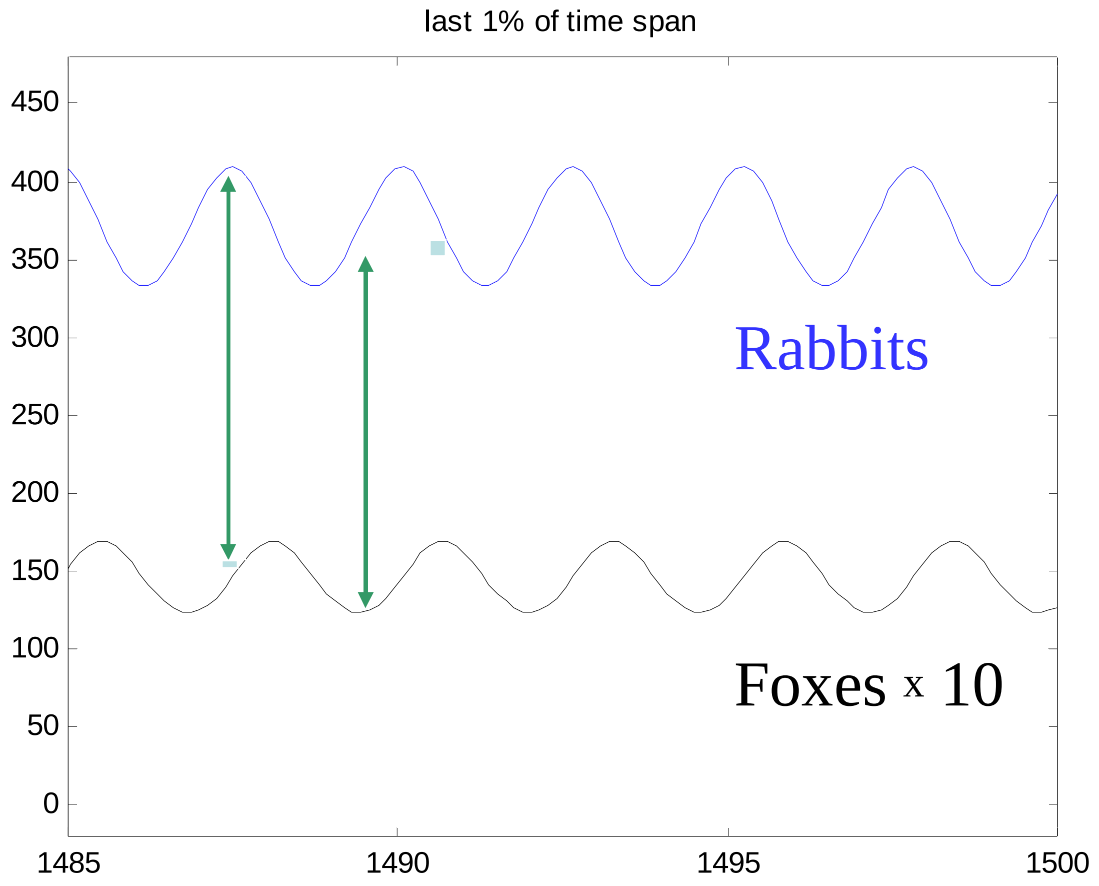
Rabbit and Fox Populations vs. Time



Solution of Rabbit and Fox Populations vs last 10% of time



Solution of rabbit and fox populations vs. last 1% of time



$r^*=370$

$f^*=14.5$

Summary

- Matlab has powerful built-in functions to numerically solve difficult ODE’s
- The challenge is constructing the Matlab script to provide the initial conditions and call ode45 and to write the user-supplied function that evaluates the r.h.s. of the ODE’s.