MATLAB – ODE

Euler's method. The following M-file computes values of the dependent variable y over a range of values of the independent variable t. The name of the function $f(x, y)$ in the equation $\frac{dy}{dx} = f(x, y)$ is passed into the function as `fxy`. Also, the initial and final values of the desired range of the independent variable is passed as a vector `tspan`. The initial value and the desired step size are passed as `y0` and `h`, respectively.

```
function [t,y] = eulode(fxy,tspan,y0,h,varargin)
% eulode: Euler ODE solver
% [t,y] = eulode (fxy,tspan,y0,h,p1,p2,...):
% uses Euler's method to integrate an ODE
% input:
% fxy = name of the M-file that evaluates the ODE
% tspan = [ti, tf] where ti and tf = initial and
% final values of independent variable
% y0 = initial value of dependent variable
% h = step size
% p1,p2,... = additional parameters used by fxy
% output:
% t = vector of independent variable
% y = vector of solution for dependent variable
if nargin < 4,
    error('at least 4 input arguments required'),
end
ti = tspan(1);
tf = tspan(2);
if ~(tf > ti),
    error('upper limit must be greater than lower'),
end
t = (ti:h:tf)';
n = length(t);
% if necessary, add an additional value of t
% so that range goes from t = ti to tf
```

```
if t(n)<tf
    t(n + 1) = tf;
    n = n + 1;
end
y = y0*ones(n,1); %preallocate y to improve efficiency
for i = 1:n - 1 %implement Euler's method
    y(i + 1) = y(i) + fxy(t(i),y(i),varargin{:})*(t(i + 1) -
t(i));
end
```

Give a try to the following:

```
>> fxy = @ (t,y) 4*exp(0.8*t) - 0.5*y;

>> [t,y] = eulode(fxy,[0 4],2,1);

>> disp([t,y])
```

Popular 4^{th} order Runge-Kutta method. The following M-file computes values of the dependent variable y according to the 4^{th} order RK method over a range of values of the independent variable t. The name of the function $f(x, y)$ in the equation $\frac{dy}{dx} = f(x, y)$ is passed into the function as `fxy`. Also, the initial and final values of the desired range of the independent variable is passed as a vector `tspan`. The initial value and the desired step size are passed as `y0` and `h`, respectively

```
function [x,y] =  rk4th(fxy,tspan,yo,h,varargin)

if nargin < 4,
    error('at least 4 input arguments required'),
end
ti = tspan(1);
tf = tspan(2);
if ~(tf > ti),
    error('upper limit must be greater than lower'),
end

x = ti:h:tf ;
y = zeros(1,length(x));
y(1)= yo ;
```

```
for i = 1:(length(x)-1)
    k_1 = fxy(x(i),y(i),varargin{:});
    k_2 = fxy(x(i)+0.5*h,y(i)+0.5*h*k_1,varargin{:});
    k_3 = fxy((x(i)+0.5*h),(y(i)+0.5*h*k_2),varargin{:});
    k_4 = fxy((x(i)+h),(y(i)+k_3*h),varargin{:});
    y(i+1) = y(i) + (1/6)*(k_1+2*k_2+2*k_3+k_4)*h;
end
%fxy = @(x,y) 3.*exp(-x)-0.4*y;
%[x,y] =  rk4th(dydx,0,100,-0.5,0.5);
%plot(x,y,'o-');

end
```

Give a try to the following:

```
>> fxy = @ (t,y) 4*exp(0.8*t) - 0.5*y;

>> [t,y] = rk4th(fxy,[0 100],-0.5,0.5);

>> plot(t,y,'o-')
```

Popular 4th order Runge-Kutta method. The following M-file solves a system of ODEs using 4th order RK method. The initial and final values of the desired range of the independent variable is passed as a vector `tspan`. The initial value and the desired step size are passed as vector `y0` and `h`, respectively

```
function [tp,yp] = rk4sys(fxy,tspan,y0,h,varargin)
% rk4sys: fourth-order Runge-Kutta for a system of ODEs
% [t,y] = rk4sys(fxy,tspan,y0,h,p1,p2,...): integrates
% a system of ODEs with fourth-order RK method
% input:
% fxy = name of the M-file that evaluates the ODEs
% tspan = [ti, tf]; initial and final times with output
% generated at interval of h, or
% = [t0 t1 ... tf]; specific times where solution output
% y0 = initial values of dependent variables
% h = step size
% p1,p2,... = additional parameters used by fxy
% output:
% tp = vector of independent variable
% yp = vector of solution for dependent variables
if nargin < 4,
```

```
    error('at least 4 input arguments required'),
end
if any(diff(tspan)<= 0),
    error('tspan not ascending order'),
end
n = length(tspan);
ti = tspan(1);
tf = tspan(n);
if n == 2
    t = (ti:h:tf)';
    n = length(t);
    if t(n)<tf
        t(n + 1) = tf;
        n = n + 1;
    end
else
    t = tspan;
end
tt = ti;
y(1,:) = y0;
np = 1;
tp(np) = tt;
yp(np,:) = y(1,:);
i = 1;
while(1)
    tend = t(np + 1);
    hh = t(np + 1) - t(np);
    if hh > h,
        hh = h;
    end
    while(1)
        if tt+hh > tend,
            hh = tend-tt;
        end
        k1 = fxy(tt,y(i,:),varargin{:})';
        ymid = y(i,:) + k1.*hh./2;
        k2 = fxy(tt + hh/2,ymid,varargin{:})';
        ymid = y(i,:) + k2*hh/2;
        k3 = fxy(tt + hh/2,ymid,varargin{:})';
        yend = y(i,:) + k3*hh;
        k4 = fxy(tt + hh,yend,varargin{:})';
        phi = (k1 + 2*(k2 + k3) + k4)/6;
        y(i + 1,:) = y(i,:) + phi*hh;
```

```
        tt = tt + hh;
        i = i + 1;
        if tt >= tend,
            break,
        end
    end
    np = np + 1;
    tp(np) = tt;
    yp(np,:) = y(i,:);
    if tt >= tf,
        break,
    end
end
% function dy = dydtsys(t, y)
% dy = [y(2);9.81 - 0.25/68.1*y(2)^2];
% end
% [t y] = rk4sys(@dydtsys,[0 10],[0 0],2);
% disp([t' y(:,1) y(:,2)])
```

Give a try to the following:

```
function dy = dydtsys(t, y)
dy = [y(2);9.81 - 0.25/68.1*y(2)^2];
end
>> [t y] = rk4sys(@dydtsys,[0 10],[0 0],2);
>> disp([t' y(:,1) y(:,2)])
```