```
function [wc] = rkf(ti,wi,h,f,argf,argPhi)
%routine that implements the Runge-Kutta-Fehlberg method
%part of Algorithm 5.3 in Burden & Faires

%k 's need to take x and y

  k1=h*f(ti,wi,argf);  %set the K1 to K6 as described in algorithm 5.3
  k2=h*f(ti+argPhi(2,1)*h,wi+k1*argPhi(2,2),argf);  %use argPhi butcher tableau matrix
  k3=h*f(ti+argPhi(3,1)*h,wi+argPhi(3,2)*k1+argPhi(3,3)*k2, argf);
  k4=h*f(ti+argPhi(4,1)*h,wi+argPhi(4,2)*k1+argPhi(4,3)*k2+argPhi(4,4)*k3, argf);
  k5=h*f(ti+h,wi+argPhi(5,2)*k1+argPhi(5,3)*k2+argPhi(5,4)*k3+argPhi(5,5)*k4, argf);

k6=h*f(ti+h*argPhi(6,1),wi+argPhi(6,2)*k1+argPhi(6,3)*k2+argPhi(6,4)*k3+argPhi(6,5)*k4+argP
hi(6,6)*k5, argf);

  %Runge-Kutta method with LTE of order five, w5
  wtilde_iplus1 =
wi+argPhi(8,2)*k1+argPhi(8,4)*k3+argPhi(8,5)*k4+argPhi(8,6)*k5+argPhi(8,7)*k6;
  %use above to estimate the local error in a Runge-Kutta method of order
  %four givn by:
  w_iplus1 = wi+argPhi(7,2)*k1+argPhi(7,4)*k3+argPhi(7,5)*k4+argPhi(7,6)*k5;

  %w4 first row, w5 second row
  wc=[w_iplus1; %the van der pol has 2 variables so we have x and y
      wtilde_iplus1 ]; %and wc is 2x2 matrix


end






function [fty] = fvdp(t, y, argf)
% this routine defines the f(t,y) in the Van der Pol problem
% the matrix size of the output argument fty is 1X2



dx = y(2);
dy = argf*(1 - y(1)^2)*y(2) - y(1);  %we plug in 1 and 2 because

fty = [dx dy];
end
function [ yt ] = odesolver( a, b, y0, hmin, hmax, tol, f, argf, option)
```

```
%routine implements an adaptive step-size method ex. rkf45
%Runge-Kutta-Fehlberg algorithm approximates soln of IVP with local
%truncation error within a given tolerance
%part of Algorithm 5.3 in Burden & Faires

%yt is a Nx3 matrix

switch option
   case 'rkf';

   %argPhi is the 8*7 Butcher Tableau matrix for rkf45
argPhi=[0 0 0 0 0 0 0
   1/4 1/4 0 0 0 0 0
   3/8 3/32 9/32 0 0 0 0
   12/13 1932/2197 -7200/2197 7296/2197 0 0 0
   1 439/216 -8 3680/513 -845/4104 0 0
   1/2 -8/27 2 -3544/2565 1859/4104 -11/40 0
   0 25/216 0 1408/2565 2197/4104 -1/5 0
   0 16/135 0 6656/12825 28561/56430 -9/50 2/55];

ti(1)=a;  %set t=a
wi(1,:)=y0;  %y0 is initial condition alpha, vectorized
h=hmax;
j=2;
i=1;  %set up a flag index starting at 1

while i==1  %while i=1, implement the rkf method
   %use (t,w) in next while loop step

[wc] = rkf(ti(j-1),wi(j-1),h,f,argf,argPhi);  %wc is a vector where wc(1) is w4
%and wc(2) is w5

  %R is the difference between w tilde_i+1 and w_i+1 over h
  R=1/h*abs(wc(:,1)-wc(:,2));

  % get L norm (norm infinity) of R, since R is a matrix and we need to
  % compare it to a scalar tolerance
  R = norm(R, inf)

  if R<=tol
     ti(j)=ti(j-1)+h;  %approximation accepted
     wi(j,:)=wc(:,1); %w approximates y(t)
     j=j+1;
  end
```

```
  delta=0.84*(tol/R)^(1/4);
  if delta<=0.1
    h=0.1*h;
  elseif delta>=4
    h=4*h;
  else
    h=delta*h;  %calculate new h
  end
  if h>hmax, h=hmax; end
  if ti(j-1)>=b
    break;  %exit while loop set i=0, flag=0
  elseif ti(j-1)+h>b
    h=b-ti(j-1);
  elseif h<hmin  %procedure completed unsuccessfully
    error('Minimum h exceeded');
  end
end

yt = [ti' wi'];

end
```

```
%main script solves IVPs specified in Solutions

%plug in start and end position,
m = odesolver(0,20,[2,0],0.01,0.25,10^-4,@fvdp, 2,'rkf');
size = length(m);

for i = 1:size


ti(i) = m{i}(1);

x(i) = m{i}(2);

y(i) = m{i}(3);
```

```
end

figure(1);

plot(ti,x,'-o')

figure(2);

plot(ti,y,'-o')

figure(3);

plot(x,y,'-o')
```