function [x,y] = RungeKutta4(fxy,x0,h,xmax,y0)

% YU-HAO CHIANG

%This is a function which using Runge-Kutta method to find the approximate

%of the ODE

%[x,y] = RungeKutta(fxy,x0,h,xmax,y0)

%First of all define the function fxy that represent the ODE

%[diff(y);diff(y,2)] = [0 1;-x 0] \* [y;diff(y)]

%

%Next define the interval with x0 as the inital value of x and xmax as the

%terminate value of the function

%

%Define h as the stepwidth

%

%Then define y0 as the initial value of y, and y0 can be scalar or column

%vector

flag1 = iscolumn(y0);

flag2 = isscalar(y0);

if flag1 == 1 | flag2 == 1

if isscalar(x0)

if isscalar(h)

if isscalar(xmax)

Y = y0;

y0 = Y(1);

y0p = Y(2);

x = x0:h:xmax;

y = y0;

yp = y0p;

xv = zeros(length(x),1);

yv = zeros(length(x),1);

ypv = zeros(length(x),1);

for i = 1:length(x);

xv(i) = x(i);

yv(i) = y;

ypv(i) = yp;

X=[0 1;-x(i) 0];

Y=[y;yp]

k1 = f(X,Y);

k2 = f(X+[0 0;-0.5\*h 0],Y+0.5\*h\*k1);

k3 = f(X+[0 0;-0.5\*h 0],Y+0.5\*h\*k2);

k4 = f(X+[0 0;-h 0],Y+h\*k3);

Y= Y + h/6 \* (k1+2\*k2+2\*k3+k4);

y = Y(1);

yp =Y(2);

end

X = xv;

Y = yv;

plot(xv,yv);

else

error('xmax should be scalar')

end

else

error('stepwidth h should be scalar')

end

else

error('x0 should be scalar')

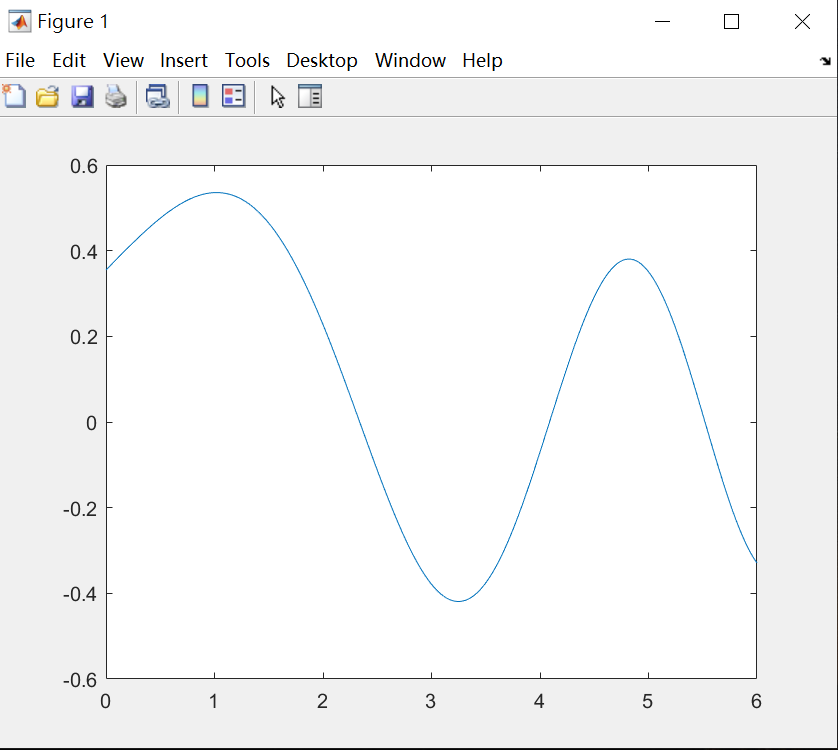
end

else

error('y0 should be scalar or calumn vector')

end

end



function [A] = jacobipq(A,itermax,tol)

%[A] = jacobipq returns the converge diagonal matrix A

%elements are the eigen value of the orginal input matrix A

%

%set 'itermax' for the max iterations of the function

%

%set 'tol' to set any value smaller than 'tol' during process to 0

A0 = A - diag(diag(A));

[r,c] = find(A0 == max(A0(:)));

p = r(1);

q = c(1);

for n = 1:itermax

U = diag(ones(1,size(A,2)));

%zeros(size(A,1),size(A,2));

cot\_2phi = (A(q,q) - A(p,p))/(2\*A(p,q));

tan\_2phi = 1/cot\_2phi;

sec\_2phi = sqrt(tan\_2phi^2 + 1);

cos\_2phi = 1/sec\_2phi;

cos\_phi = sqrt((cos\_2phi + 1)/2);

sin\_phi = sqrt((1 - cos\_2phi)/2);

U(p,p) = cos\_phi;

U(q,q) = cos\_phi;

U(p,q) = sin\_phi;

U(q,p) = -sin\_phi;

%U = U - diag(diag(U)) + diag(ones(1,size(U,1)));

A = U'\*A\*U;

sym = A-A';

if all(sym,'all') > tol

error('A is not symmetric')

end

A(abs(A) < tol) = 0;

A0 = A - diag(diag(A));

[r,c] = find(abs(A0) == max(abs(A0(:))));

p = r(1);

q = c(1);

end

end