```
function [ xi, wi ] = XisWis( )
%XisWis Store weight and sample points for the quadrature rule.
%   Use 16-point Gauss quadrature rule.

xi = [-0.989400934991649932596154, ...
-0.944575023073232576077988, ...
-0.865631202387831743880467, ...
-0.755404408355003033895101, ...
-0.617876244402643748446671, ...
-0.458016777657227386342419, ...
-0.281603550779258913230460, ...
-0.095012509837637440185319, ...
0.095012509837637440185319, ...
0.281603550779258913230460, ...
0.458016777657227386342419, ...
0.617876244402643748446671, ...
0.755404408355003033895101, ...
0.865631202387831743880467, ...
0.944575023073232576077988, ...
0.989400934991649932596154 ];

wi = [0.027152459411754094851780, ...
0.062253523938647892862843, ...
0.095158511682492784809925, ...
0.124628971255533872052476, ...
0.149595988816576732081501, ...
0.169156519395002538189312, ...
0.182603415044923588866763, ...
0.189450610455068496285396, ...
0.189450610455068496285396, ...
0.182603415044923588866763, ...
0.169156519395002538189312, ...
0.149595988816576732081501, ...
0.124628971255533872052476, ...
0.095158511682492784809925, ...
0.062253523938647892862843, ...
0.027152459411754094851780 ];

xi=xi';
wi=wi';

end
```

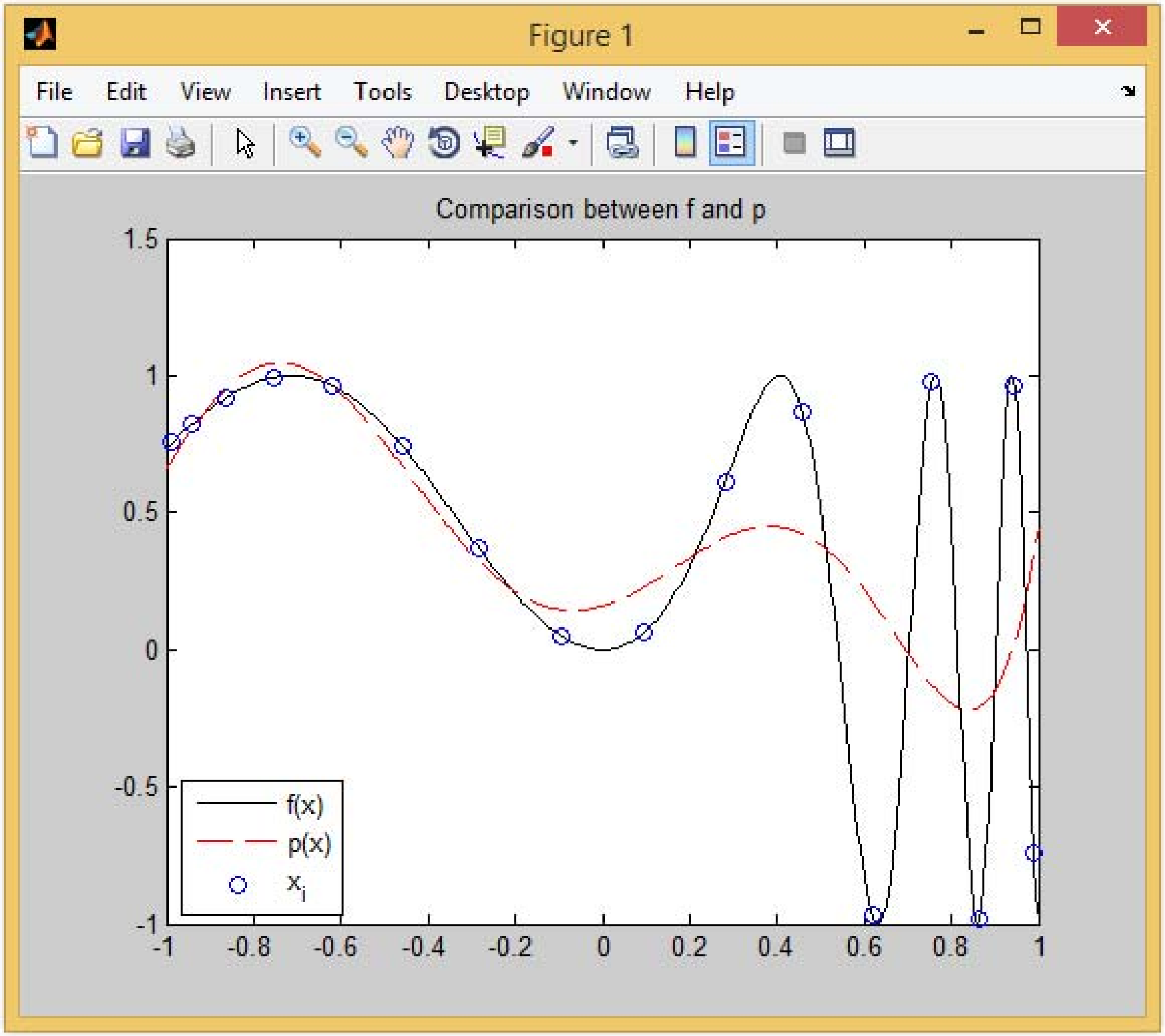
```
function [ c ] = LeastSquareP6( f )
%LeastSquareP6 Compute the coefficients for the least square of f from P_6
%   Input: f is the function to be approximated. We assume -1 <= x <=1
%   Output: c = vector of coefficients of least square approx of f from P_6
%   which is in the form a_0L_0 + ... + a_6L_6. Each L_i is Legendre poly.

[ x, w ] = XisWis( );

N = length(x);
L = zeros(N,7);
L(:,1) = ones(size(x));
L(:,2) = x;
L(:,3) = (3*x.^2-1)/2;
L(:,4) = (5*x.^3-3*x)/2;
L(:,5) = (35*x.^4-30*x.^2+3)/8;
L(:,6) = (63*x.^5-70*x.^3+15*x)/8;
L(:,7) = (231*x.^6-315*x.^4+105*x.^2-5)/(16);

y = f(x);

c = zeros(1,7);
for r=1:7
    c(r) = sum(w.*L(:,r).*y)*(2*(r-1)+1)/2;
end

end
```

```
function [ xplot, yplot, pplot ] = PlotLeastSquareP6( h )
%PlotLeastSquareP6 Plot the function and its least sqaure approx from P_6
%   Input: h = mesh size for the plot
%   Output = all info we need for the plot

xplot = -1:h:1;
xplot = xplot';
f = inline( 'sin(2*pi*x.^2.*exp(x))' ); %Can change function here.
yplot = f(xplot);

[ c ] = LeastSquareP6( f );
pplot=0;

N = length(xplot);
L = zeros(N,7);
L(:,1) = ones(size(xplot));
L(:,2) = xplot;
L(:,3) = (3*xplot.^2-1)/2;
L(:,4) = (5*xplot.^3-3*xplot)/2;
L(:,5) = (35*xplot.^4-30*xplot.^2+3)/8;
L(:,6) = (63*xplot.^5-70*xplot.^3+15*xplot)/8;
L(:,7) = (231*xplot.^6-315*xplot.^4+105*xplot.^2-5)/(16);

for r=1:7
    pplot = pplot + c(r)*L(:,r);
end

xi = XisWis( );
yi = f(xi);
plot(xplot,yplot,'-k',xplot,pplot,'--r',xi,yi,'ob');
legend('f(x)','p(x)','x_i');
legend('Location', 'Southwest');
title('Comparison between f and p');
% hold on;
% xx = -1:0.1:1;
% yy = f(xx);
% plot(xx,yy,'ob');
% hold off;

end
```

Command Window

New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> [ xplot, yplot, pplot ] = PlotLeastSquareP6( 1/200 );
>>
```



```
function [ c ] = LeastSquareP2( f, a, b )
%LeastSquareP2 Compute the coefficients for the least square of f from P_2
%   Input: f is the function to be approximated. We assume a <= x <= b
%   Output: c = vector of coefficients of least square approx of f from P_2
%   which is in the form a0L0 + a1L1 + a2L2. Each Li is Legendre poly.

[ zi, wi ] = XisWis( );

xi = (b-a)*zi/2 + (a+b)/2;

N = length(xi);
L = zeros(N,3);
L(:,1) = ones(size(zi));
L(:,2) = zi;
L(:,3) = (3*zi.^2-1)/2;

y = f(xi);

c = zeros(1,3);
for r=1:3
    c(r) = sum(wi.*L(:,r).*y)*(2*(r-1)+1)/2;
end

end
```

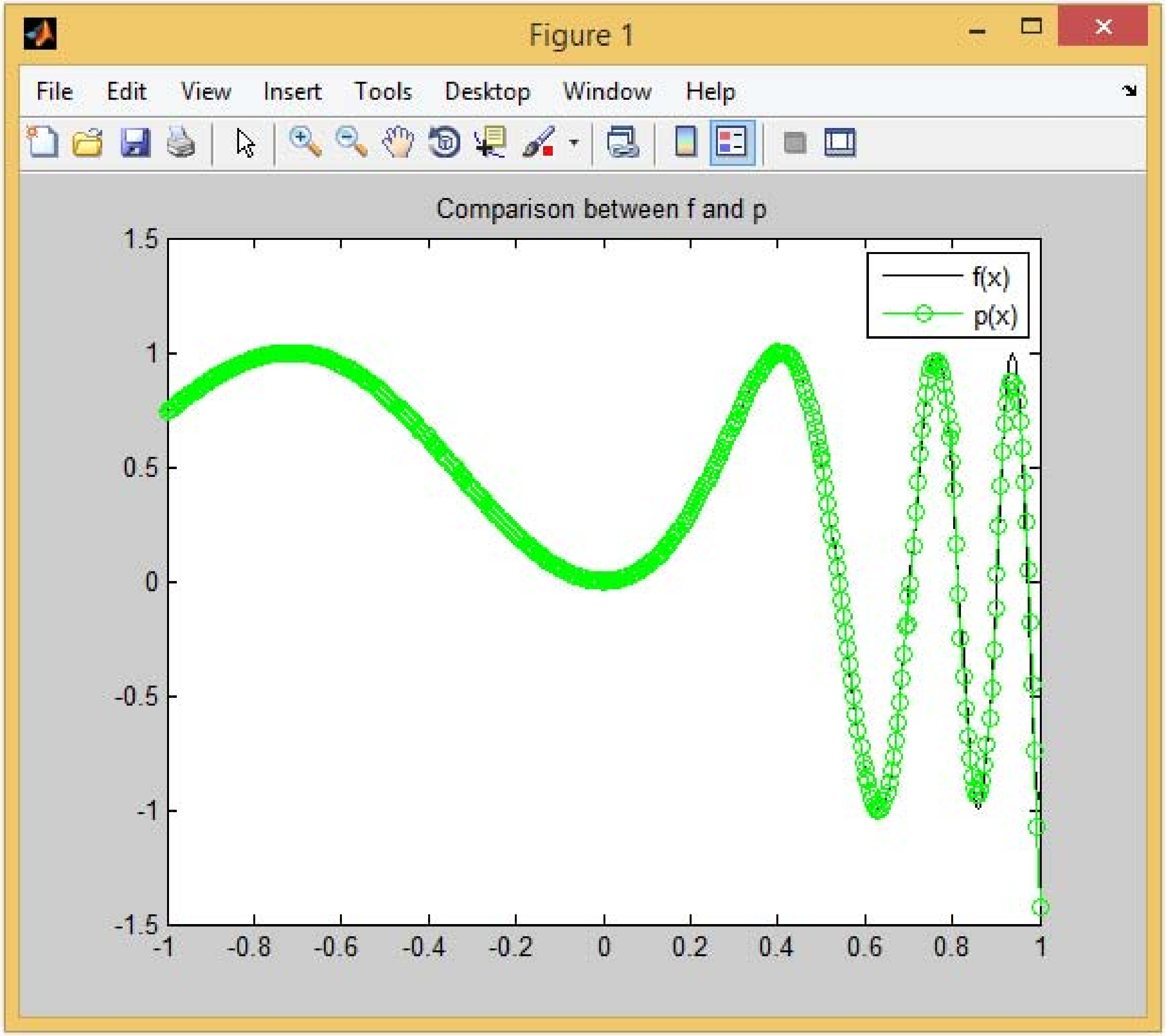
```
function [ xplot, yplot ] = PlotLeastSquareP2( a, b, h, c )
%PlotLeastSquareP2 Compute data points for the least square plot
%   Input: [a,b] = domain, h = mesh size for the plot,
%           c = coefficients of the least square approx.
%   Output = all info we need for the plot

xplot = a:h:b;
zi = 2*xplot/(b-a) + (a+b)/(a-b);

yplot = c(1)*ones(size(zi)) + c(2)*zi + c(3)*(3*zi.^2-1)/2;

xplot = xplot';
yplot = yplot';

%p = plot(xplot,yplot,'.r');

end
```

```
function [ x, y, xplot, yplot, p ] = PlotLeastSquareP2h( h, hplot )
%PlotLeastSquareP2h Plot the function and its least sqaure approx from P_2
%   Input: h = mesh size for the plot, hplot = 0.1 for this homework
%   Output = all info we need for the plot

x = -1:h:1;
x = x';
f = inline( 'sin(2*pi*x.^2.*exp(x))' ); %Can change function here.
y = f(x);

xp = -1:hplot:1;
xp1 = xp(1):h:xp(2);
N = length(xp)-1;
n1 = length(xp1);
xplot = zeros(n1,N);
yplot = zeros(n1,N);
p = zeros(1,N+1);
p(1) = plot(x,y,'-k');
hold on;
for r=1:N
    ar = xp(r);
    br = xp(r+1);
    cr = LeastSquareP2( f, ar, br );
    [ xplot(:,r), yplot(:,r) ] = PlotLeastSquareP2( ar, br, h, cr );
    p(r+1) = plot(xplot(:,r),yplot(:,r),'o-g');
end

legend([p(1) p(2)],'f(x)','p(x)');
title('Comparison between f and p');

hold off;

end
```

Command Window

New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> [ x, y, xplot, yplot, p ] = PlotLeastSquareP2h( 1/200, 1/10 );
>>
```



③ Heun's method

$$\begin{cases} y^* = y_n + hf(x_n, y_n) \\ y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_n + h, y^*)\right] \end{cases}$$

1. $Y_{n+1} = Y_n + hY_n' + \frac{h^2}{2} Y_n'' + \frac{h^3}{6} Y_n''' + O(h^4)$ by Taylor expansion.

Since we assume $Y_n = y_n$, we have that

$$Y_{n+1} = y_n + hy_n' + \frac{h^2}{2} y_n'' + \frac{h^3}{6} y_n''' + O(h^4) \quad \text{— (I)}$$

2. $y_{n+1} = y_n + \frac{h}{2}\left[f_n + f(x_n+h, y^*)\right]$

$= y_n + \frac{h}{2}\left[f_n + f(x_n, y^*) + hf_x(x_n, y^*) + \frac{h^2}{2} f_{xx}(x_n, y^*) + O(h^3)\right]$

$= y_n + \frac{h}{2}\Big[f_n + \left(f_n + (hf_n)\frac{\partial}{\partial y} f_n + \frac{(hf_n)^2}{2}\frac{\partial^2}{\partial y^2} f_n + O(h^3)\right)$

$+ h\left(\frac{\partial}{\partial x} f_n + (hf_n)\frac{\partial^2}{\partial y \partial x} f_n + O(h^2)\right)$

$+ \frac{h^2}{2}\left(\frac{\partial^2}{\partial x^2} f_n + O(h)\right) + O(h^3)\Big]$

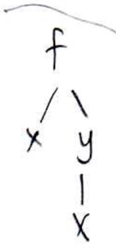
Notation

$\frac{\partial}{\partial x} f_n = f_x(x_n, y_n)$

$\frac{\partial}{\partial y} f_n = f_y(x_n, y_n)$

$y_n' = f_n$

$= y_n + hf_n + \frac{h^2}{2}\left(y_n' \frac{\partial}{\partial y} f_n + \frac{\partial}{\partial x} f_n\right) + h^3\left(\frac{(y_n')^2}{4}\frac{\partial^2}{\partial y^2} f_n + \frac{y_n'}{2}\frac{\partial^2}{\partial y \partial x} f_n + \frac{1}{4}\frac{\partial^2}{\partial x^2} f_n\right) + O(h^4)$



$= y_n + hy_n' + \frac{h^2}{2} y_n'' + h^3\left(\frac{(y_n')^2}{4} f_{yy}(x_n, y_n) + \frac{y_n'}{2} f_{xy}(x_n, y_n) + \frac{1}{4} f_{xx}(x_n, y_n)\right)$

$+ O(h^4) \quad \text{— (II)}$

$\frac{df}{dx} = f_x + f_y \frac{dy}{dx}$

$f' = f_x + f_y y'$

From (I) and (II), we get

$$\tau_{n+1} = \frac{Y_{n+1} - y_{n+1}}{h} = h^2\left[\frac{y_n'''}{6} - \left(\frac{(y_n')^2}{4} f_{yy} + \frac{y_n'}{2} f_{xy} + \frac{f_{xx}}{4}\right)\Bigg|_{(x,y)=(x_n,y_n)}\right] + O(h^3)$$

So, the Heun's method is of order 2.

```
function [ ] = ShootingMethod( z0 )
%ShootingMethod Implement the shooting method (for HW 9 problem 4 only).
%   Input = z0, initial guess. No output because everything will be
%   printed out as the code is executed.

h = 1;
y0 = -12;
x0 = 0;

display(sprintf('Step %d, x = %f, y = %f, z = %f', 0,x0,y0,z0));

for r=1:3
    y = h*z0 + y0;
    z = z0 + h*(2*z0 + y0 + x0*(x0-1));
    x = x0 + h;
    x0 = x;
    y0 = y;
    z0 = z;
    display(sprintf('Step %d, x = %f, y = %f, z = %f', r,x,y,z));
end

end
```

```
>> ShootingMethod( 0 )
Step 0, x = 0.000000, y = -12.000000, z = 0.000000
Step 1, x = 1.000000, y = -12.000000, z = -12.000000
Step 2, x = 2.000000, y = -24.000000, z = -48.000000
Step 3, x = 3.000000, y = -72.000000, z = -166.000000
>> ShootingMethod( 10 )
Step 0, x = 0.000000, y = -12.000000, z = 10.000000
Step 1, x = 1.000000, y = -2.000000, z = 18.000000
Step 2, x = 2.000000, y = 16.000000, z = 52.000000
Step 3, x = 3.000000, y = 68.000000, z = 174.000000
>> ShootingMethod( 4.714 )
Step 0, x = 0.000000, y = -12.000000, z = 4.714000
Step 1, x = 1.000000, y = -7.286000, z = 2.142000
Step 2, x = 2.000000, y = -5.144000, z = -0.860000
Step 3, x = 3.000000, y = -6.004000, z = -5.724000
>> f = inline('0 + (10-0)/(68+72)*(y+72)')

f =

     Inline function:
     f(y) = 0 + (10-0)/(68+72)*(y+72)

>> f(-6)

ans =

   4.714285714285714

>>
```