Summer 2001 CS 222 - Homework 2 Solutions

1. (10 points) Calculating the double integral of a $f(x, y)$ is the same as calculating the volume over the quarter circle region specified in the limits on the integral.

   MATLAB code:

```
function est=DoubleIntEst(n)
% est=DoubleIntEst(n)
%
% Uses Monte Carlo estimation to estimate the value of
% the integral of f(x,y)=exp(x^2*y^2) over the
% quarter circle region in the 1st quadrant.
%
% x,y,z are column n-vectors which are randomly generated
% points within a box that surrounds the volume we
% need to calculate

rand('seed',.12345)

% generate random points within the box with vertices
% (0,0),(1,0),(1,1),(0,1) and height exp(1)

% (use height=e since exp(x^2*y^2) is an increasing function
% and f(1,1)=e.  Therefore, we know that a box
% of height e will enclose the region - you could use a larger
% height and that would still work).

e=exp(1);
x=rand(n,1);
y=rand(n,1);
z=rand(n,1)*e;

% calculates the number of points within the region

NumberInside=sum( (z<=f(x,y) & (x.*x + y.*y<=1)) );
est = (NumberInside/n) * e;
```

```
function fval=f(x,y)
% function fval=f(x,y)
%
% x and y are column vectors
% fval is the value of f at x and y where
% f(x,y)=exp(x^2*y^2)

fval=exp(x.*x.*y.*y);
```

Output:

```
   n    Integral Estimate
------------------------
  100     0.67957
 1000     0.88888
 5000     0.84702
10000     0.81494
```

There was a subtlty in this problem that I did not think about before assigning it. Many people (including myself) generated their random numbers in the quarter circle by multiplying the scaling factor with a vector of random numbers. This is not exactly correct, because it no longer guarantees that the numbers are uniformly distributed (multiplying uniformly distributed numbers by a nonlinear factor destroys the uniformity). I did not take points off for this error, as I never mentioned it in class. In the case of a circle, the numbers are not that far from uniformity. However, in the case of an ellipse or other shape, the case could be worse.

The correct way to generate uniformly distributed random numbers within a circle, is to generate random numbers in the square that encloses the circle and simply remove those points that are not inside the circle. This is the method I have used in the solutions.

2. (5 points) The interpolant over f is much less accurate than the interpolant over g because of the nondifferentiability at $x = 0$. In particular, note that

$$f'''(x) = \frac{3}{8x^{\frac{5}{2}}}$$

which is $+\infty$ at $x = 0$. Observe from the discussion on page 91 of the text that the accuracy of polynomial interpolation depends on M_3, which is the maximum absolute value of the 3rd derivative of f on the interval of consideration. Since M_3 is undefined (infinite) for f on $[0, 1]$ one would expect the polynomial interpolant to be inaccurate. On the other hand, the 3rd derivative of g on $[0, 1]$ does not blow up, thus

$$|g(z) - p_2(z)| << |f(z) - p_2(z)|$$

and we have a better interpolant.

3. (15 points) MATLAB code:

```
% HW2 - problem 3
%
% Compares polynomial and trigonometric interpolation
% of the square wave function, sqwv(x), on the interval [xlower,xupper]
%
% n is the number of points used for interpolation
% m is the number of points used for evaluation
% x is a n-vector of points in [0,2]
% z is a m-vector of points in [0,2]

% assign initial variables
clc
clear
xlower=0;
xupper=2;

m=200;     % number of divisions to use for plotting/integrating
```

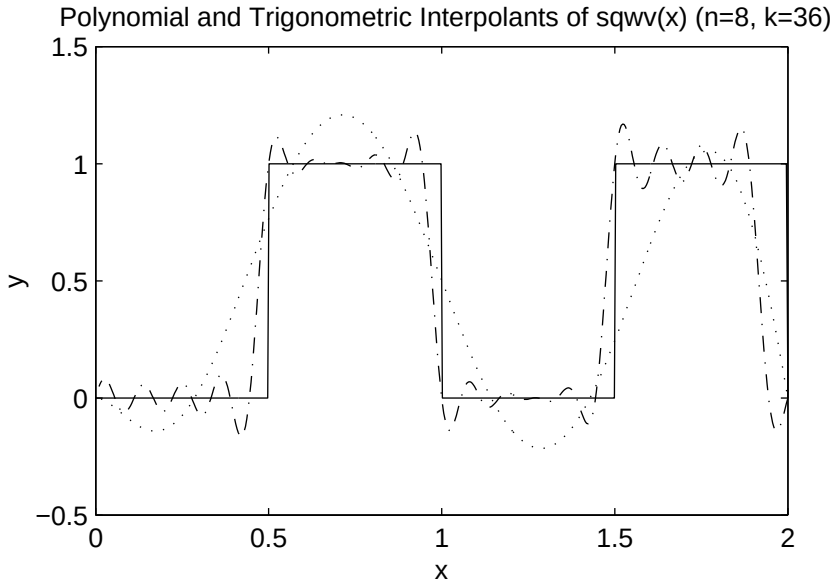
```
n=8;       % 1+degree of polynomial interpolation to use (experiment)
k=36;      % number of points to use in trig interpolation (experiment)
           %  -- MUST be an even integer!


%%%%%%%%%% POLYNOMIAL INTERPOLANT %%%%%%%%%%%%%%

x=linspace(xlower,xupper,n)';
y=sqwv(x);
z=linspace(xlower,xupper,m)';
sqz=sqwv(z);

% compute coefficients of polynomial and evaluate
c=InterpN(x,y);
pval=HornerN(c,x,z);

% computes and displays polyerror
polyerror=dist(z,pval,sqz)

% plot sqwv(x) and p(x)
plot(z,sqz,'-',z,pval,':');
hold on

%%%%%%%%%% TRIGONOMETRIC INTERPOLANT %%%%%%%%%%%%%%

x=linspace(xlower,xupper,k)';
y=sqwv(x);

% compute coefficients and evaluate
F=CSInterp(y);
tval=CSEval(F,xupper-xlower,z);

% computes and displays trigerror
trigerror=dist(z,tval,sqz)

% plot t(x)
plot(z,tval,'-.');
xlabel('x')
ylabel('y')
title('Polynomial and Trigonometric Interpolants of sqwv(x) (n=8, k=36)')
hold off
```

```
function sqwvval=sqwv(x)
% function sqwvval=sqwv(x)
%
% sqwv produces a square wave of period 1
% for each integer n, and any x in [n,n+1)
% sqwv(x) = 0 if x in [n,n+0.5) and
% sqwv(x) = 1 if x in [n+0.5,n+1).
```

```
sqwvval=round(x-floor(x));
```

Output:

```
polyerror =

    0.1523


trigerror =

    0.1439
```



4. (10 points)

   (a) MATLAB functions g and gp which computes the function value and it's derivative:

```
function y=g(x);
% y=g(x)
%
% x is a n-column vector
% y is a n-column vector that holds the value of
% g(x), where
%
%   g(x)=(exp(x)-(cos(3*pi*x))^2)/(3+x)

c=cos(3*pi*x);
y=(exp(x)-c.*c)./(3+x);
```

```
-------------------

function yp=gp(x)
% yp=gp(x)
```

```
%
% x is a n-column vector
% yp is a n-column vector that holds the value of
% g'(x), where
%
%   g(x)=(exp(x)-(cos(3*pi*x))^2)/(3+x)

angle=3*pi*x;
c=cos(angle);
ex=exp(x);
tpx=3+x;

yp=(ex + 6*pi*c.*sin(angle))./tpx - (ex-c.*c)./(tpx.*tpx);
```

(b) We start with

$$q_i(z) = a_i + b_i(z - x_i) + c_i(z - x_i)(z - x_{i+1})$$

We require that $q_i(x_i) = y_i$ and $q_i(x_{i+1}) = y_{i+1}$. Therefore,

$$y_i = q_i(x_i) = a_i$$

and

$$y_{i+1} = q_i(x_{i+1}) = a_i + b_i(x_{i+1} - x_i) = y_i + b_i(x_{i+1} - x_i)$$

$$\Rightarrow \qquad b_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Note that these are the same coefficients as in piecewise linear interpolation.

(c) MATLAB functions:

```
function [a,b,c]=pwQ(x,y,c1)
% [a,b,c]=pwQ(x,y,c1)
% Piecewise Quadratic Interpolation
%
% x,y column n-vectors with x(1) < ... < x(n)
%
% a,b,c column (n-1)-vectors that define a continuous piecewise
% quadratic polynomial q(z) with the property that for i=1:n
%
%      q(x(i))=y(i)
%
% On the interval [x(i),x(i+1)],
%
%   q(z)=a(i) + b(i)(z-x(i)) + c(i)(z-x(i))(z-x(i+1))



% use piecewise linear interpolation to find a,b since the formulas
% are the same

[a,b]=pwL(x,y);
```

```
% below, we find c

% assigning and initializing

n=length(x);
nm1=n-1;
c=zeros(nm1,1);
c(1)=c1;

% we have to do the following calculation in a loop since the value
% of c(i) depends on c(i-1)

h=x(2)-x(1); %spacing is equal between all the data points
for i=2:nm1
  c(i)=-1/h * (c(i-1)*h + ((y(i)-y(i-1))/h) - ((y(i+1)-y(i))/h));
end
```

```
  function Cvals = pwQEval(a,b,c,x,zVals)
% Cvals = pwQEval(a,b,c,x,zVals)
%
% Evaluates the piecewise quadratic polynomial defined by the column
% (n-1)-vectors a,b,c,
% and the column n-vector x. It is assumed that x(1) < ... < x(n).
% zVals  is a column m-vector with each component in [x(1),x(n)].
%
% CVals is a column m-vector with the property that CVals(j) = C(zVals(j))
% for j=1:m where on the interval [x(i),x(i+1)]
%
%   C(z)= a(i) + b(i)(z-x(i)) + c(i)(z-x(i)(z-x(i+1))

m = length(zVals);
Cvals = zeros(m,1);
g=1;
for j=1:m
   i = Locate(x,zVals(j),g);
   Cvals(j) = c(i)*(zVals(j)-x(i+1)) + b(i);
   Cvals(j) = Cvals(j)*(zVals(j)-x(i)) + a(i);
   g = i;
end
```

(d) MATLAB script for (f) and (g):

```
% HW2 - problem 4
%
% for g(x)=(exp(x)-cos(3*pi*x).^2)./(3+x)
% computes and plots:  cubic Hermite interpolant with given data
%                      cubic Hermite interpolant with perturbed data
%                      quadratic interpolant with given data
%                      quadratic interpolant with perturbed data
%
% x,y are column n-vectors where y(i)=g(x(i))
```

```
% s is a column n-vector where s(i)=g'(x(i))
% zvals is a m-vector that is used to evaluate the interpolant
% a,b,c,d are column (n-1) vectors with the value of the cubic interpolant
% anew,bnew,cnew,dnew are column (n-1) vectors with the value of the
%    perturbed cubic interpolant
% aq,bq,cq are column (n-1) vectors with the value of the quadratic interp.
% aqnew,bqnew,cqnew are column (n-1) vectors with the value of the
%    perturbed quadratic interpolant

clear
clc
close all

n=8;
m=50;
x=linspace(-.5,.5,n)';
y=g(x);
s=gp(x);
zvals=linspace(-.5,.5,m);
gzvals=g(zvals);

% set up perturbed data
ynew=y;
ynew(1)=y(1)+1;


%%%%%%% COMPUTE CUBIC HERMITE INTERPOLANT %%%%%%%%%

% work with given data
[a,b,c,d]=pwC(x,y,s);
Cvals=pwCEval(a,b,c,d,x,zvals);
plot(zvals,gzvals,'-',zvals,Cvals,':');
legend('g(x)','Interpolant')
title('g(x) and its cubic Hermite interpolant (given data)')
xlabel('x')
ylabel('y'),

% work with perturbed data
figure
[anew,bnew,cnew,dnew]=pwC(x,ynew,s);
Cvals=pwCEval(anew,bnew,cnew,dnew,x,zvals);
plot(zvals,gzvals,'-',zvals,Cvals,':');
legend('g(x)','Interpolant')
title('g(x) and its cubic Hermite interpolant (perturbed data)')
xlabel('x')
ylabel('y'),

%%%%%%% COMPUTE QUADRATIC INTERPOLANT %%%%%%%%%%

% work with given data
figure
```

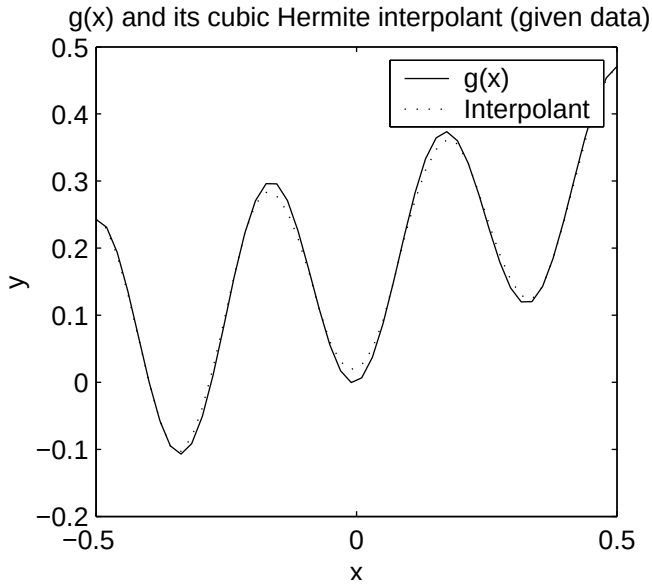
```
c1=5;
[aq,bq,cq]=pwQ(x,y,c1);
Cvalsq=pwQEval(aq,bq,cq,x,zvals);
plot(zvals,gzvals,'-',zvals,Cvalsq,':');
legend('g(x)','Interpolant')
title('g(x) and its quadratic interpolant (given data)')
xlabel('x')
ylabel('y'),

% work with perturbed data
figure
[aqnew,bqnew,cqnew]=pwQ(x,ynew,c1);
Cvalsq=pwQEval(aqnew,bqnew,cqnew,x,zvals);
plot(zvals,gzvals,'-',zvals,Cvalsq,':');
legend('g(x)','Interpolant')
title('g(x) and its quadratic interpolant (perturbed data)')
xlabel('x')
ylabel('y')

% displays all results

disp(sprintf('                    Quadratic Interpolant         '))
disp(sprintf('-----------------------------------------------------------'))
disp(sprintf('         given data          |       perturbed data  '))
disp(sprintf('     a         b         c   |     a        b         c'))
disp(sprintf('-----------------------------------------------------------'))

for i=1:n-1
  disp(sprintf('%8.4f  %8.4f  %8.4f  %8.4f  %8.4f  %8.4f',aq(i),bq(i),cq(i),
     aqnew(i),bqnew(i),cqnew(i)))
end

disp(' ')
disp(' ')

disp(sprintf('               Cubic Hermite Interpolant         '))
disp(sprintf('-------------------------------------------------------------------------------',
disp(sprintf('               given data               |           perturbed data  '))
disp(sprintf('     a         b         c        d     |     a         b        c        d '))
disp(sprintf('-------------------------------------------------------------------------------',

for i=1:n-1
  disp(sprintf('%9.4f %9.4f %9.4f %9.4f %9.4f %9.4f %9.4f %9.4f',a(i),b(i),
  c(i),d(i),anew(i),bnew(i),cnew(i),dnew(i)))

end
```

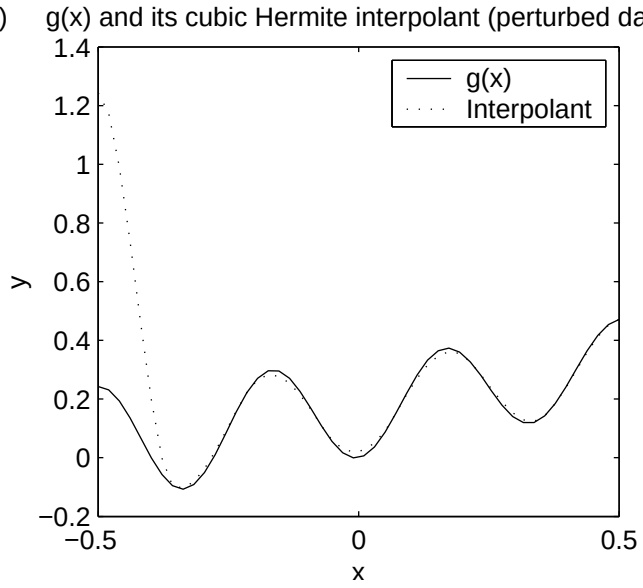
Output:

```
                    Quadratic Interpolant
-----------------------------------------------------------
         given data          |       perturbed data
     a         b         c   |     a        b         c
```

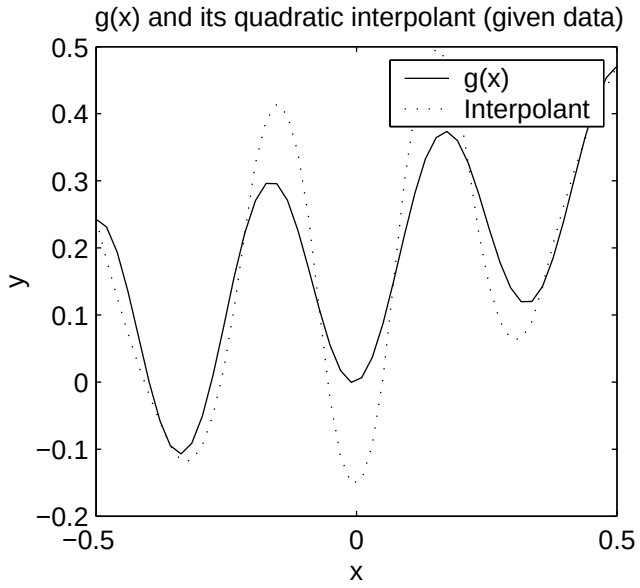
```
------------------------------------------------------
  0.2426   -2.3626    5.0000    1.2426   -9.3626    5.0000
 -0.0949    2.2194   27.0740   -0.0949    2.2194   76.0740
  0.2222   -0.7907  -48.1446    0.2222   -0.7907  -97.1446
  0.1092    0.2903   55.7116    0.1092    0.2903  104.7116
  0.1507    1.2335  -49.1090    0.1507    1.2335  -98.1090
  0.3269   -1.2900   31.4444    0.3269   -1.2900   80.4444
  0.1426    2.2992   -6.3203    0.1426    2.2992  -55.3203
```

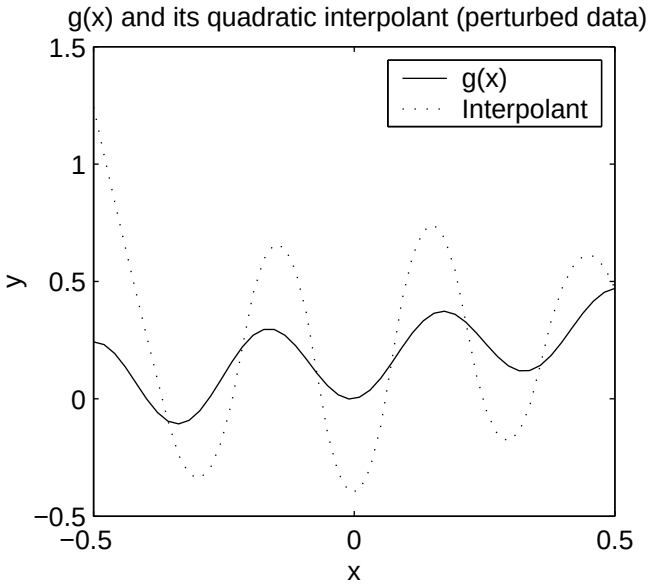
```
                Cubic Hermite Interpolant
-------------------------------------------------------------------------------
                given data                |           perturbed data
    a         b         c         d       |     a         b         c         d
-------------------------------------------------------------------------------
   0.2426    0.1456  -17.5572  177.5822     1.2426    0.1456  -66.5572  863.5822
  -0.0949   -1.2466   24.2623 -138.6858    -0.0949   -1.2466   24.2623 -138.6858
   0.2222    2.8551  -25.5206   77.4003     0.2222    2.8551  -25.5206   77.4003
   0.1092   -2.8569   22.0304   -7.1184     0.1092   -2.8569   22.0304   -7.1184
   0.1507    3.2922  -14.4109  -57.9923     0.1507    3.2922  -14.4109  -57.9923
   0.3269   -2.0087    5.0310  106.4575     0.3269   -2.0087    5.0310  106.4575
   0.1426    1.6013    4.8850 -130.3679     0.1426    1.6013    4.8850 -130.3679
```





(e) Output:





First, notice that for the same number of points that are interpolated, the cubic Hermite interpolant is a better approximation to g. With only one more degree polynomial, we get a much better approximation. Second, notice what happens when we perturb only one $y-$value; in the case of the piecewise cubic Hermite interpolant, the approximation is still good (except at the point we perturbed). However, in the quadratic case, it changes the entire interpolant. Therefore quadratic interpolation is very sensitive to small changes in the data – in fact changing the interpolant over the entire domain.