Summer 2001 CS 222 - Homework 3 Solutions

1. (10 points) MATLAB code:

```
% HW3 - problem 1
%
% Find the maximum of the function
% g(x)=exp(-x)sin(4 pi x^2) + (x^3)/3 - 1/4 for x
% in the interval [0,1];
%
% finding the maximum of g(x) is same as finding roots of g'(x) where
% g'(x)= 8 pi x exp(-x)cos(4 pi x^2) - exp(-x)sin(4 pi x^2) + x^2;

clear
clc

x=linspace(0,1)';
yp=gp(x);

% first, look at g'(x) and see where g'(x)=0
plot(x,yp);
title('Plot of g''(x)')

% we see from the plot, that g'(x) has several roots.
% note that the root finding method specified assumes that g' has to
% be monotone and that there is one root in the interval.  We need to
% come up with bracketing intervals such that there exists only one
% root in each.  There are several ways to do this.  One way, is to simply
% inspect the graph and find intervals that bracket only one root of the
% derivative.

% there are more sophisticated ways that look for where g' is increasing and
% decreasing, however, any way that you bracket the intervals is acceptable.

% the intervals below were found by visually inspecting the graph

lx=[0   0.2 0.55 0.75 0.9 ]';
rx=[0.2 0.4 0.65 0.82 0.97]';

% Number of intervals for the interpolation per
% bracketting interval.
n=20;

droots=[];

% Find all the roots of g'(x) in [0,1]
for i=1:length(lx)
  x=linspace(lx(i),rx(i),n);
```

```
  yp=gp(x);
  [yp i]=sort(yp);
  x=x(i);

  % get the spline
  s=spline(yp,x);
  droots=[droots; ppval(s,0)];
end

% Find the maximum for g(x) on [0,1] by checking each root.
[ymax i]=max(exp(droots).*sin(4*pi*droots.^2) + (droots.^3)/3 - 0.25);
disp(sprintf('The max of g(x) on [0,1] is: %5.4f', ymax))
disp(sprintf('for x=%5.4f',droots(i)))
disp(' ');
```

```
function yp=gp(x)
% yp=gp(x)
%
% x is a vector
% yp is the derivative of g(x)

angle=4*pi*(x.*x);
yp=8*pi*x.*cos(angle).*exp(-x)-exp(-x).*sin(angle)+ x.*x;
```

Output:

```
The max of g(x) on [0,1] is: 2.1217
for x=0.7915
```

2. (10 points)

   (a) We need to show that

$$\int_a^b C(x)dx = \frac{h}{2}(f(a)+f(b)) + \frac{h^2}{12}(f'(a)-f'(b))$$

   Note that the cubic Hermite interpolant of a function $f(x)$ is given by

$$C(x) = c_1 + c_2(x-a) + c_3(x-a)^2 + c_4(x-a)^2(x-b)$$

   where

$$\begin{aligned} c_1 &= f(a) \\ c_2 &= f'(a) \\ c_3 &= \frac{\frac{f(b)-f(a)}{b-a} - f'(a)}{b-a} \\ c_4 &= \frac{f'(b)+f'(a)-2\frac{f(b)-f(a)}{b-a}}{(b-a)^2} \end{aligned}$$

Therefore we have (using integration by parts for the c_4 term),

$$\begin{aligned}
\int_a^b C(x)dx &= c_1 x + c_2\frac{(x-a)^2}{2} + c_3\frac{(x-a)^3}{3} + c_4\left[\frac{(x-a)^3}{3}(x-b) - \frac{(x-a)^4}{12}\right]\Bigg|_a^b \\
&= c_1 h + c_2\frac{h^2}{2} + c_3\frac{h^3}{3} - c_4\frac{h^4}{12} \quad \text{where } h = b-a \\
&= \frac{h}{2}[f(a)+f(b)] + \frac{h^2}{12}[f'(a)-f'(b)] \quad \text{(substituting for } c_1, c_2, c_3, c_4 \text{ and rearranging)}
\end{aligned}$$

(b) To develop a composite rule, divide the interval $[a,b]$ into n equally spaced intervals and we apply the rule above to each piece (note that n intervals implies we have x-values $a = x_1 < x_2 < \ldots < x_n < x_{n+1} = b$).

$$\begin{aligned}
\int_a^b f(x)dx &\approx \int_a^b C(x)dx \\
&= \sum_{i=1}^{n}\left\{\frac{h}{2}[f(x_i)+f(x_{i+1})] + \frac{h^2}{12}[f'(x_i)-f'(x_{i+1})]\right\} \\
&= \frac{h}{2}[(f(a)+f(x_2)) + (f(x_2)+f(x_3)) + \ldots + (f(x_n)+f(b))] \\
&\quad + \frac{h^2}{12}[(f'(a)-f'(x_2)) + (f'(x_2)-f'(x_3)) + \ldots + (f'(x_n)-f'(b))] \\
&= \frac{h}{2}\left[f(a) + 2\sum_{i=2}^{n} f(x_i)\right] + \frac{h^2}{12}(f'(a)-f'(b)) \\
&= \frac{h}{12}\left[6(f(a)+f(b)) + h(f'(a)-f'(b)) + 12\sum_{i=2}^{n} f(x_i)\right]
\end{aligned}$$

(c) MATLAB code that implements this composite rule is below.

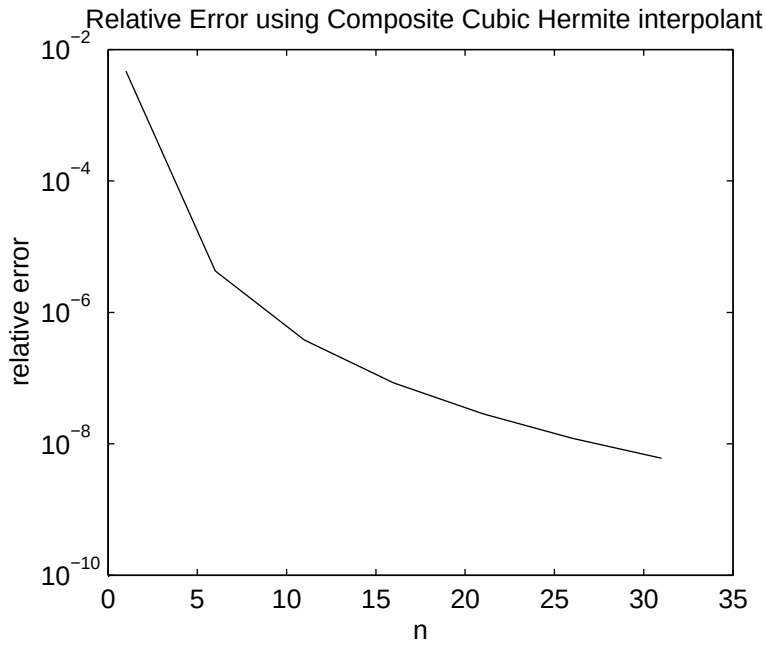
```
function q=CompQH(fname,fpname,a,b,n)
% q=CompQH(fname,fpname,a,b)
%
% Integrates a function using a composite Hermite rule
%
% fname  is a string containing the name of the function to be integrated
% fpname is a string containing the name of the function that implements
%   the derivative of fname
% a,b are the lower and upper limits of integration (a<b)
% n is the number of pieces for the composite rule (n<=1)
% q is the approximation to the integral

x=linspace(a,b,n+1)';
f=feval(fname,x);
fp=feval(fpname,[a;b]);

h=(b-a)/n;

q=6*(f(1)+f(n+1)) + h * (fp(1)-fp(2)) + 12* ones(1,n-1)*f(2:n);
q=q*h/12;
```

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

function f=prob4(x)
% f=prob4(x)
% computes f(x)=exp(x)*sin(x)

f=exp(x) .* sin(x);

-----------------

function fp=dprob4(x)
% fp=dprob4(x)
% function to compute derivative of f(x)=exp(x)*sin(x)
%
% f'(x)=exp(x)*(cos(x)+sin(x))

fp=exp(x) .* (cos(x) + sin(x));

-----------------

% HW3 - problem 2
% script to test CompQH.m
% also will generate a plot of relative error using the composite
% cubic Hermite interpolant

clear
clc

a=6;
b=7;
err=[];

% compute actual value of the integral
int=.5 * ( exp(b)*(sin(b)-cos(b)) - exp(a)*(sin(a)-cos(a)) );

disp('Value of integral for each n')
for n=1:5:31
  % Compute approximation
  q=CompQH('prob2','dprob2',a,b,n);
  disp(sprintf('n=%d ==> integral est= %8.4f',n,q));
  err=[err;abs((int-q)/q)];
end

semilogy((1:5:31)',err)
title('Relative Error using Composite Cubic Hermite interpolant')
xlabel('n')
ylabel('relative error')
```

Output:

```
Value of integral for each n
n=1 ==> integral est= 197.8252
n=6 ==> integral est= 196.9027
n=11 ==> integral est= 196.9019
n=16 ==> integral est= 196.9019
```

```
n=21 ==> integral est= 196.9018
n=26 ==> integral est= 196.9018
n=31 ==> integral est= 196.9018
```



3. (5 points) MATLAB code:

```
% HW3 - problem 3
%
% computes integral estimates of Gamma(x)=int(t^(x-1)*exp(-t))
% using both the composite Simpson's quadrature rule and adaptive
% Simpson's quadrature rule

clear
clc

% assign initial variables
u=50; %upper limit of integration
m=3;  %specifies simpson's rule
n=10;
tol=.01;


disp(' x      Simp       relerr     AdaptSimp    relerr')
disp('-------------------------------------------------')
for x=1:10

  % computes (x-1)!
  if x==1
    xm1fact=1;
  else
    xm1fact=xm1fact*(x-1);
```

```
  end

  % simpson's rule
  numI(x)=myCompQNC('f',0,u,m,n,x);
  err(x)=abs((xm1fact-numI(x))/xm1fact);

  % adaptive quad
  numIA(x)=myAdaptQNC('f',0,u,m,tol,x);
  errA(x)=abs((xm1fact-numIA(x))/xm1fact);

  % displays results
  disp(sprintf('%2.0f   %5.3e   %5.3e    %5.3e   %5.3e',x,numI(x),err(x),
       numIA(x),errA(x)))
end
```

```
function fx=f(x,t)
% fx=f(x,t)
%
% computes the value of
%  t^(x-1)*exp(-t) which is the integrand
% of the gamma function

fx=t.^(x-1).*exp(-t);
```

```
  function numI = myCompQNC(fname,a,b,m,n,x)
% numI = CompQNC(fname,a,b,m)
%
% Integrates a function of the form f(x) named by the string fname from a to b
% f must be defined on [a,b] and it must return a column vector if x is a
% column vector.
% m is an integer that satisfies 2 <= m <= 11.
% numI is the composite m-point Newton-Cotes approximation of the integral
% of f
% from a to b with n equal length subintervals.

Delta = (b-a)/n;
h = Delta/(m-1);
t = a+h*(0:(n*(m-1)))';
w = NCWeights(m);
t = linspace(a,b,n*(m-1)+1)';
f = feval(fname,x,t);
numI = 0;
first = 1;
last = m;
for i=1:n
   %Add in the inner product for the i-th subintegral.
   numI = numI + w'*f(first:last);
   first = last;
```

```
   last = last+m-1;
end
numI = Delta*numI;
```

```
  function numI = myAdaptQNC(fname,a,b,m,tol,x)
% numI = AdaptQNC(fname,a,b,m,tol)
%
% Integrates a function from a to b
% fname is a string that names an available function of the form f(x) that
% is defined on [a,b]. f should return a column vector if x is a column vector.
% a,b are real scalars, m is an integer that satisfies 2 <= m <=11, and
% tol is a positive real.
%
% numI is a composite m-point Newton-Cotes approximation of the
% integral of f(x) from a to b, with the abscissae chosen adaptively.

A1 = myCompQNC(fname,a,b,m,1,x);
A2 = myCompQNC(fname,a,b,m,2,x);
d = 2*floor((m-1)/2)+1;
error = (A2-A1)/(2^(d+1)-1);
if abs(error) <= tol
   % A2 is acceptable
   numI = A2+error;
else
   % Sum suitably accurate left and right integrals
   mid = (a+b)/2;
   numI = myAdaptQNC(fname,a,mid,m,tol/2,x) + ...
       myAdaptQNC(fname,mid,b,m,tol/2,x);
end
```

Output:

x	Simp	relerr	AdaptSimp	relerr
1	1.120e+00	1.201e-01	1.000e+00	4.528e-05
2	7.549e-01	2.451e-01	8.281e-04	9.992e-01
3	2.104e+00	5.210e-02	1.035e-02	9.948e-01
4	6.559e+00	9.315e-02	1.294e-01	9.784e-01
5	2.463e+01	2.644e-02	2.400e+01	4.758e-05
6	1.175e+02	2.116e-02	1.200e+02	2.238e-06
7	7.020e+02	2.506e-02	7.200e+02	3.413e-06
8	4.990e+03	9.983e-03	5.040e+03	3.404e-07
9	4.040e+04	2.104e-03	4.032e+04	4.677e-09
10	3.648e+05	5.365e-03	3.629e+05	2.269e-09

Note that your results for error depend on your upper limit of integration so your output may not exactly match mine. In general, adaptive quadrature is best in this case since the gamma function is skewed towards the origin. Therefore, close to the origin we need more intervals to integrate accurately and less intervals as we approach infinity. With regular Simpson's rule, we are forced to use equal spacing between the points and so the higher the upper limit is, the worse our estimate gets. This

may not be intuitive since it would seem that as we increase the upper limit of integration, we would have a better integral estimate. However, because of the shape of this function we really only need to integrate to about 20 or so where the function flattens out and converges to zero.

4. (10 points) MATLAB code:

```
function C=multAB(A,sup,d,sub)
% function C=multAB(A,sup,d,sub)
%
% A is an mxn matrix
% sup is a column (n-1)-vector
% d is a column n-vector
% sup is a column (n-1) vector
%
% C=A*B, where B is the tridiagonal matrix with d as its
% diagonal, sup as its superdiagonal, and sub as its subdiagonal

[m,n] = size(A);
C = zeros(m,n);

% calculate first and last column of C using only 2 elements of B
C(:,1) = A(:,1:2)*[d(1); sub(1)];
C(:,n) = A(:,n-1:n)*[sup(n-1); d(n)];

% calculates middle columns of C
for i = 2:n-1
  C(:,i) = A(:,i-1:i+1)*[sup(i-1); d(i); sub(i)];
end
```

Output:

```
C =

    0.5573    0.4423
    0.3968    0.2996


C =

    0.0214    0.5137    0.4550
    0.1877    0.4969    0.6774
    0.3069    1.1408    0.6594
    0.5273    1.2427    0.8715


C =

    0.2605    0.5817    1.4571    1.1417
    0.2035    0.2281    0.9351    0.7398
    0.6859    0.5144    0.9133    0.7344
```