Math 2400 - Numerical Analysis

Homework #3 Solutions

1. Write a program for polynomial interpolation of function values. Your program should accept two vectors as input. A vector of x coordinates for interpolation and a vector of y values (the function values at the given x coordinates). Your program should then: Compute the divided differences table for an arbitrary set of data pairs.
   Test your code by constructing a polynomial of degree at most 2 that interpolates $f(x) = e^x$ at $x_1 = 0$, $x_2 = 1$, $x_3 = 2$. Provide the divided difference table, the polynomial and the value of $P_2(0.5)$ in your solution.
   You may use this code for the remainder of the assignment.
   There is already a discussion about the coding for this subroutine in lab 2 notes. So here I will provide the source.

```
function F=divdif(x,y)

%  x is a column vector containing the x-cordinates
%  y is a column vector containing the function values
%  F will be a matrix of divided differences

     len=length(y);
     F=zeros(len);

     F(:,1)=y;

     for i=2:(len)
          for j=i:(len)
               F(j,i)=(F(j,i-1)-F(j-1,i-1))/(x(j)-x(j-i+1));
          end
     end
return
```

   Here is the session testing the code.

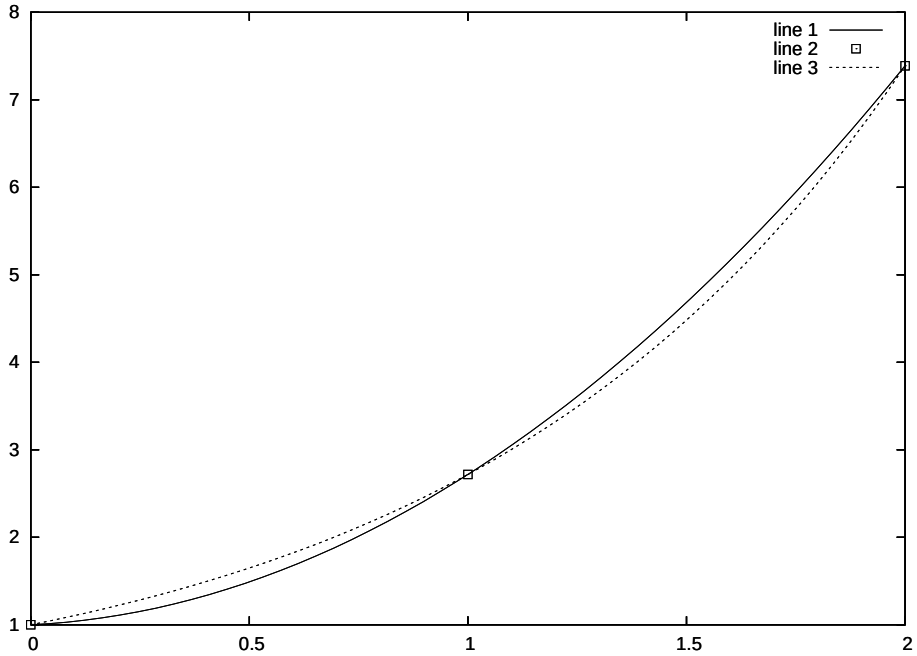
```
octave:2> x=linspace(0,2,3)'
x =

  0
  1
  2

octave:3> y=exp(x)
y =

  1.0000
  2.7183
  7.3891
```

```
octave:4> F=divdif(x,y)
F =

  1.00000  0.00000  0.00000
  2.71828  1.71828  0.00000
  7.38906  4.67077  1.47625

octave:5> [xi,yi]=divdifplot(x,F);
octave:6> plot(xi,yi)
octave:7> hold on
octave:8> plot(x,y,'x')
octave:9> plot(xi,exp(xi))
octave:10> gset output "hw3-07q1.eps"
octave:11> gset terminal postscript eps
octave:12> replot
octave:13> quit
```

The interpolating polynomial is then given by,

$$P_2(x) = 1 + 1.71828x + 1.47625x(x-1)$$

and the value of $P_2(.2) = 1.4901$ which compares to $e^{1/2} = 1.6487$, but not too well. Here is the graph produced by the session.



2. Write a program that inputs a vector of interpolation points, $x_1, \ldots, x_n$ and the interpolation values $y_1, \ldots . y_n$. The program should then output the coefficients for the interpolating cubic spline with free boundary conditions. Use the code to find the interpolating cubic spline for the function $y = \frac{1}{1+x^2}$ with $x_1 = -2$, $x_2 = -1$, $x_3 = 1$ and $x_4 = 2$. Graph and compare the spline with the original function.
   Here is a listing of my code.

```
function [ a b c d ]= cubspline(x,y)

% input x the interpolation points
% input y the interpolation values
n=length(x);

%  n the number of interpolation points

% output a b c d - vectors of lenght n-1
% the ith spline s(i)=a(i)+b(i)(x-x(i))+c(i)(x-x(i))^2+d(i)(x-x(i))^3


% A will be the matrix used to solve for the c's
% f is the right hand side of the matrix

A=zeros(n);
f=zeros(n,1);

a=zeros(n-1,1);
b=zeros(n-1,1);
c=zeros(n,1);  % recal we have an extra c which we don't need

d=zeros(n-1,1);

% set up the A matrix and f vector
A(1,1)=1;
f(1)=0;
for i=2:n-1
   A(i,i-1)=x(i)-x(i-1);
   A(i,i+1) =x(i+1)-x(i);
   A(i,i)=2*(x(i+1)-x(i-1));
   f(i)=3*((y(i+1)-y(i))/(x(i+1)-x(i))-(y(i)-y(i-1))/(x(i)-x(i-1)));
end
A(n,n)=1;
f(n)=0;

% solve for c's and then the other coef.
c=inverse(A)*f;
for i=1:n-1
   d(i)=(c(i+1)-c(i))/(3*(x(i+1)-x(i)));
   b(i)=(y(i+1)-y(i))/(x(i+1)-x(i)) - (x(i+1)-x(i))/3*(c(i+1)+2*c(i));
   a(i)=y(i);
end

return
```

I also wrote a program to plot the splines. Here it is.

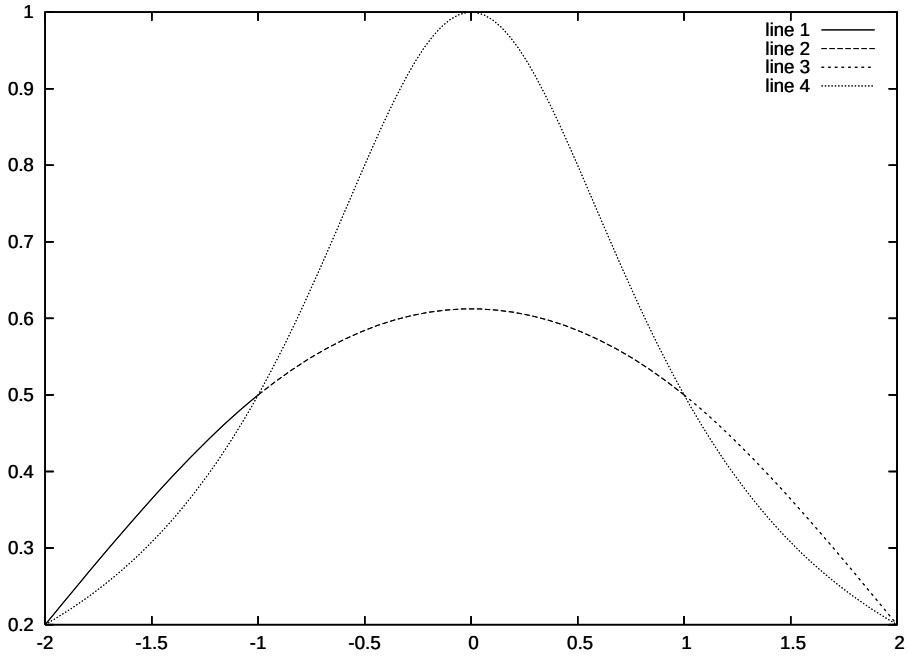
```
function plotspline(a,b,c,d,xi)

n=length(xi)-1;
x=zeros(n,26);
for i=1:n
   x(i,:)=linspace(xi(i),xi(i+1),26);
end

s=zeros(n,26);
for i=1:n
   s(i,:)=a(i)+b(i).*(x(i,:)-xi(i))+c(i).*(x(i,:)-xi(i)).^2+d(i).*(x(i,:)-xi(i)).^3;
end
hold on
for i=1:n
   plot(x(i,:),s(i,:))
end
```

Here is the session I used to test the program.

```
octave:2> x=[-2 -1 1 2]'
x =

  -2
  -1
   1
   2

octave:3> y=1./(1+x.^2)
y =

  0.20000
  0.50000
  0.50000
  0.20000

octave:4> [a b c d]=cubspline(x,y)
a =

  0.20000
  0.50000
  0.50000

b =

   0.33750
   0.22500
  -0.22500
```

```
c =

   0.00000
  -0.11250
  -0.11250
   0.00000

d =

  -0.037500
   0.000000
   0.037500

octave:5> plotspline(a,b,c,d,x)
octave:6> hold on
octave:7> xi=linspace(-2,2);
octave:8> plot(xi,1./(1+xi.^2)
> )
octave:9> gset output "hw3-07q3.eps"
octave:10> gset term postscript eps
octave:11> replot
octave:12> diary off
```

Finally here is the plot.



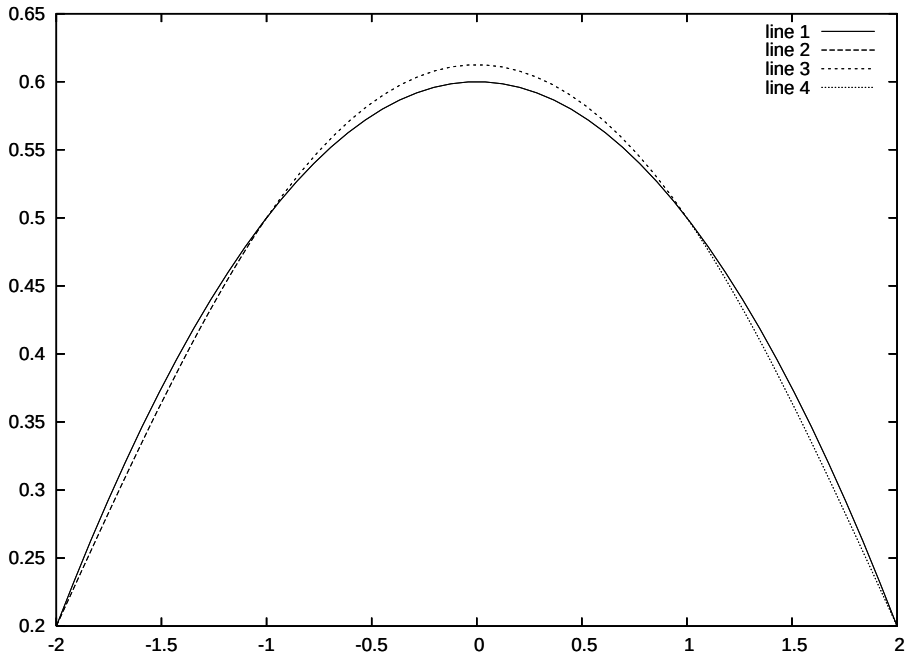
As you can see, the spline provides a very poor approximation of the function. The interval in between -1 and 1 is too large. If we had added one more point at zero, we would get a much better apprxoimation.

3. Use the function written for (1) to construct a cubic polynomial to approximate the function

$f = \frac{1}{1+x^2}$. The constructed cubic should interpolate the function values at the points $x_1 = -2$, $x_2 = -1$, $x_3 = 1$, $x_4 = 2$. Graph and Compare the approximation to that of the question (2). The approximating polynomials in this question and in question (2) are both cubics interpolating the same data. Why are the different?
Here is the session used to create the graph.

```
octave:14> F=divdif(x,y)
F =

   0.20000   0.00000   0.00000   0.00000
   0.50000   0.30000   0.00000   0.00000
   0.50000   0.00000  -0.10000   0.00000
   0.20000  -0.30000  -0.10000   0.00000

octave:15> [xi yi]=divdifplot(x,F);
octave:16> hold off
octave:17> plot(xi,yi)
octave:18> hold on
octave:19> plotspline(a,b,c,d,x)
octave:20> gset output "hw3-07q4.eps"
octave:21> gset term postscript eps
octave:22> replot
octave:23> diary off
```

The graph produced by the session.



The graph has similar problems to that of the previous question. Since we have no data between -1 and 1 and the function changes quite fast in this region, we can't expect a good approximation.
The spline and the interpolating polynomial both provide similar approximations, but as

you can see from the graph the curves are different. Although, both the spline and the interpolating polynomial are cubics interpolating the same data, they are constructed using different criteria. The spline must have zero second derivative at the endpoints for example.

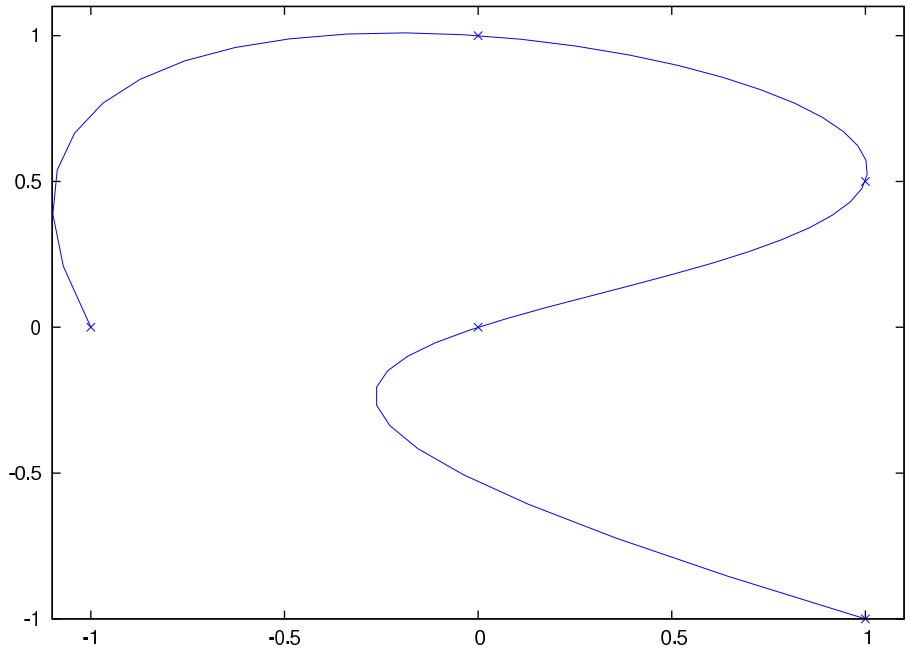
4. Write a program which will accept two vectors as input, a vector of x values (not necessarily unique) and a vector of y values. The program will then plot the parametrically defined polynomial interpolating these points.
   Test your code on the data set

i	1	2	3	4	5
x_i	-1	0	1	0	1
y_i	0	1	0.5	0	-1

   Here is the code. It follows the method used in the lab notes.

```
octave:2> x=[-1 0 1 0 1]'
x =

  -1
   0
   1
   0
   1

octave:3> y=[0 1 .5 0 -1]'
y =

   0.00000
   1.00000
   0.50000
   0.00000
  -1.00000

octave:4> para(x,y)
octave:5> hold on
octave:6> plot(x,y,'x')
octave:7> print("hw3-07q4.eps",'-d eps')
octave:8> diary off
```

   Here is the graph produced.



Again the graph goes through all the points, but it is not really the shape I had in mind when just picking points in the plane.

5. Write a program which accepts four vectors as input. Two of the vectors will contain the values for the interpolation points x_i's and y_i's. The other two vectors will contain the control points $\hat{x}_i$ and $\hat{y}_i$. The program will then draw the Bezier spline defined by the points.
Test your code on the data

i	1	2	3	4	5
x_i	-1	0	1	0	1
y_i	0	1	0.5	0	-1
$\hat{x}_i$	-1	0	1	0	2
$\hat{y}_i$	1	1	0	-1	-1

For this question, there seems to be a difference between the way I have been taught to construct Bezier splines and the way the book does it. If you follow the code in the book, the spline you get will go through the same points, but the slope will be reversed. No points will be deducted for this. Sorry for the confusion.
Here is the code for the Bezier splines:

```
function bez(x,y,xh,yh)

n=length(x);
alp=xh-x;
bet=yh-y;

t=linspace(0,1)';

bezx=zeros(100,n-1);
bezy=zeros(100,n-1);

for i=1:(n-1)
bezx(:,i)=(2*(x(i)-x(i+1))+3*(alp(i)+alp(i+1))).*t.^3+(3*(x(i+1)-x(i))
```

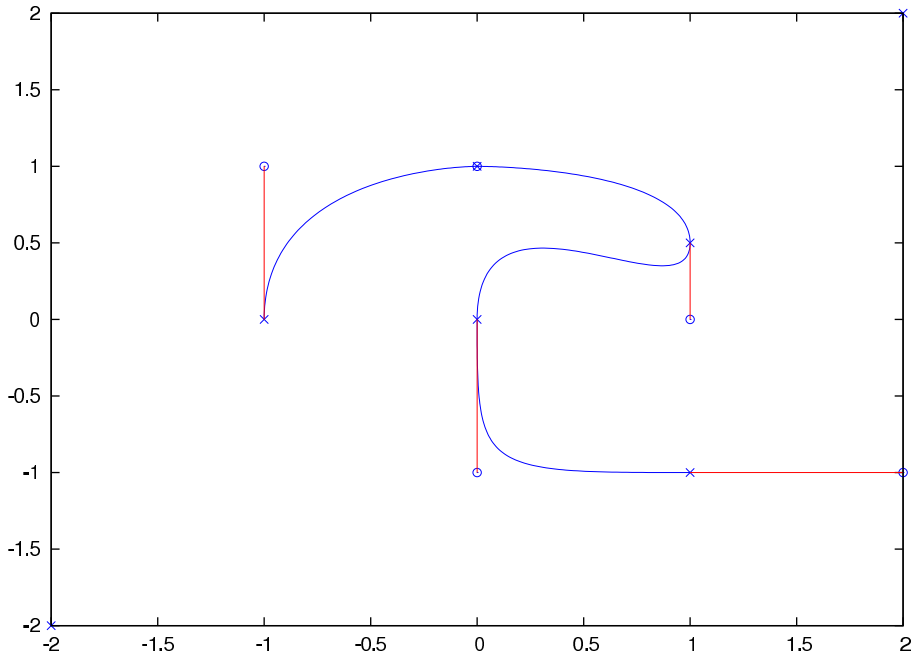
```
-3*(alp(i+1)+2*alp(i))).*t.^2+3*alp(i).*t+x(i);
bezy(:,i)=(2*(y(i)-y(i+1))+3*(bet(i)+bet(i+1))).*t.^3+(3*(y(i+1)-y(i))
-3*(bet(i+1)+2*bet(i))).*t.^2+3*bet(i).*t+y(i);
plot(bezx(:,i),bezy(:,i))
hold on
end

end
```

Here is the session which uses the bez subroutine. At the end, I plot the control points to show how the code works.

```
octave:2> x=[-1 0 1 0 1]'
x =

  -1
   0
   1
   0
   1

octave:3> y=[0 1 .5 0 -1]'
y =

   0.00000
   1.00000
   0.50000
   0.00000
  -1.00000

octave:4> xh=[-1 0 1 0 2]'
xh =

  -1
   0
   1
   0
   2

octave:5> yh=[1 1 0 -1 -1]'
yh =

   1
   1
   0
  -1
  -1
```

```
octave:6> bez(x,y,xh,yh)
octave:7> for i=1:6
> plot([x(i);xh(i)],[y(i);yh(i)],'r')
> end
octave:8> diary off
```

The graph produced by the session.



This curve is a little more like what I had in mind. You can see how the control points influence the shape of the curve. If we were to try moving the control points around a little, we could make the curve approximate a shape we like a little more.

6. Compare the results from question (5) and (4). What are any advantages one method could have over another.
   For question (4), we will get a curve which passes through all the point we choose, but it may be a little more curvy then we would like. If we were to add more points, the curves could become even more uncontrollable. The Bezier spline does not suffer from this, we can add as many points as we like and still have a nice controllable curve. We can also add slope information at each of the nodes. I think that if we wish to draw a cartoon mouse, it would be a lot easier using Bezier splines. However, we need to input twice as much information for an equivalent curve.
   I think the method of inputing points from the command line is not really appropriate for splines. This algorithm is meant to be used with a mouse or tablet for input.