CHBE 553 – Assignment 4 – 02/12/2019

Objective: To find $\bar{y}$ at $\bar{x}$ = 0.75 using 6, 9 and 12 data points

(a) Neville's Algorithm:

Neville's method can be applied in the situation that we want to interpolate f(x) at a given point x = p with increasingly higher order Lagrange interpolation polynomials.
Given m data points, we construct a m-1 polynomial which can give the approximate value of $\bar{y}$ [1].

Step 1: Construct a zero order polynomial

$$Pi(xi) = yi$$

Step 2: Since zero order polynomial isn't a very good approximation, so we construct higher order polynomial using Lagrange's polynomial method [2]

$$P_{i\,(i+1)\cdots(i+m)} = \frac{(x - x_{i+m})\,P_{i\,(i+1)\cdots(i+m-1)}}{x_i - x_{i+m}} + \frac{(x_i - x)\,P_{(i+1)\,(i+2)\cdots(i+m)}}{x_i - x_{i+m}}.$$

Result and observation:

$\bar{y}$ at $\bar{x}$ = 0.75 was calculated using 6, 9 and 12 data points using Neville's algorithm and the result are as follows:

- **6 datapoints: $\bar{y} = 0.1942$**
- **9 data points: $\bar{y} = 0.2019$**
- **12 data points: $\bar{y} = 0.1638$**

It can be seen that as the number of data point increases, the order of polynomial will increase and higher order polynomial can cause significant deviation due to round off error that is encountered while recursively evaluating the (m-1)th oder polynomial.

MATLAB CODE:

```
%interpolation using Neville's algorithm
%import data
data = xlsread('datafile.xlsx');
x = data(:,1);
y = data(:,2);

x_bar = 0.75;

%neville method with 6 data points
x1 = x([1:6],1);
```

```
y1 = y([1:6],1);
%function call to evaluate y_bar using nevielle's algorithm
y_bar1 = nev(x1,y1,x_bar)

%neville method with 9 data points
x2 = x([1:9],1);
y2 = y([1:9],1);
%function call to evaluate y_bar using nevielle's algorithm
y_bar2 = nev(x2,y2,x_bar)

%neville method with 12 data points
x3 = x([1:12],1);
y3 = y([1:12],1);
%function call to evaluate y_bar using nevielle's algorithm
y_bar3 = nev(x3,y3,x_bar)

%function to evaluate Nevielle's algorithm
function p = nev(x,y,x_bar)
[r,c] = size(x);
n = r - 1;
p = zeros(n+1,n+1);
%zero order polynomial
for i = 1:n+1
    p(i,i) = y(i);
end
%evaluating polynomial
for j = 1:n+1
    for i = 1:n+1
        if i+j <= n+1
            p(i,i+j) = ((x(i+j)-x_bar)*p(i,i+j-1) + (x_bar-
x(i))*p(i+1,i+j))/(x(i+j)-x(i));
        end
    end
end
%y_bar is equal to the last vaue from the polynomial
p = p(1,n+1);
end
```

(b) Cubic spline interpolation

Spline is a piece wise curve put together from the m-1 cubics, all of which have different co-efficients.

For continuity,

$f''_{i-1,i}(x_i) = f''_{i,i+1}(x_i)$

For natural cubic spline, $f''(x_1) = f''(x_m) = 0$

For a smooth curve, $f'_{i-1,i}(x_i) = f'_{i,i+1}(x_i)$

For i = 1,2….m

To find the unknown second derivative, a system of m-2 linear equations should be solved simultaneously using any linear solver.

Finally the cubic spline equation is

.

$$f_{i,i+1}(x) = \frac{k_i}{6}\left[\frac{(x-x_{i+1})^3}{x_i-x_{i+1}} - (x-x_{i+1})(x_i-x_{i+1})\right] - \frac{k_{i+1}}{6}\left[\frac{(x-x_i)^3}{x_i-x_{i+1}} - (x-x_i)(x_i-x_{i+1})\right] + \frac{y_i(x-x_{i+1}) - y_{i+1}(x-x_i)}{x_i-x_{i+1}}$$

Where ki and k(i+1) are f''_i and f''_{i+1} respectively.

Result and observation

$\bar{y}$ at $\bar{x}$ = 0.75 was calculated using 6, 9 and 12 data points using Neville's algorithm and the result are as follows:

- **6 datapoints: $\bar{y} = 0.2122$**
- **9 data points: $\bar{y} = 0.2034$**
- **12 data points: $\bar{y} = 0.2033$**

The following plot was obtained:

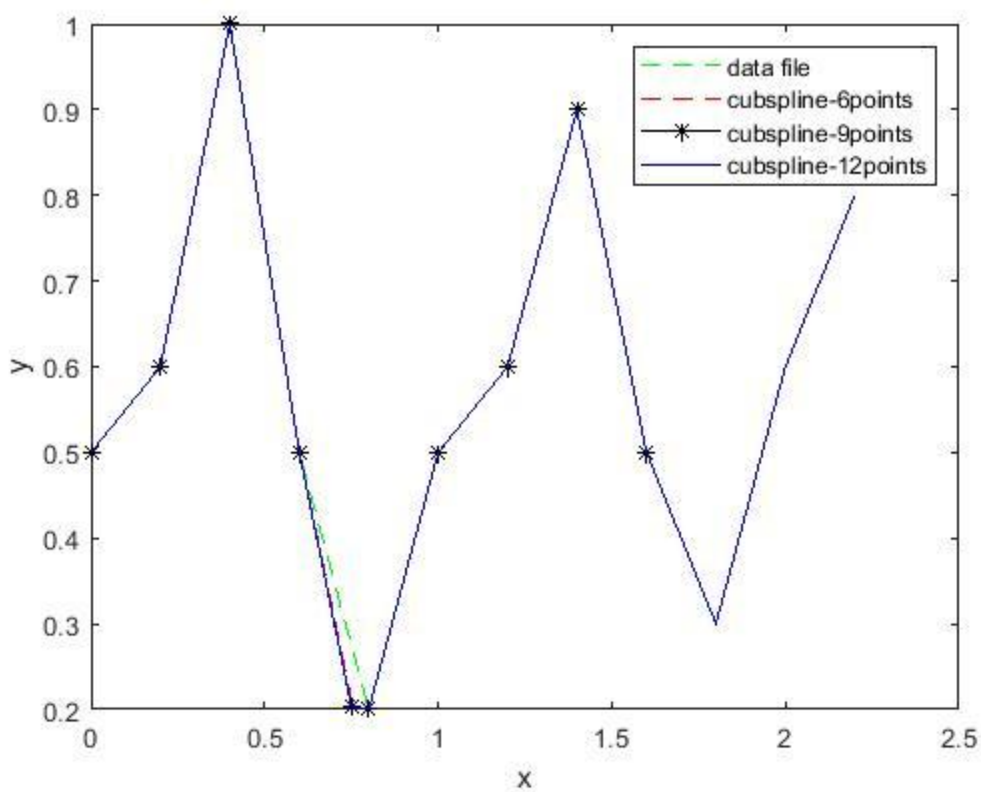

Fig 1: Cubic spline interpolation

The result obtained from cubic-spline was fairly consistent.

MATLAB CODE:

```
%cubic spline interpolation
%import data
data = xlsread('datafile.xlsx');
x = data(:,1);
y = data(:,2);
%x_bar = 0.75
x_bar = 0.75;
```

```
%using 6 points
x1 = data([1:6],1);
y1 = data([1:6],2);
%evaluating the tridiagonal matrix
k1 = matrix_eval(x1,y1);
%evaluating y_bar using cubic spline interpolation
y_bar = cub_spl(x1,y1,x_bar,k1)
%finding x_bar with in the range of x_data
i = search_x(x1,x_bar);
%inserting x_bar and the corresponding y-bar in the x_data and y_data
x_new = [x1(1:i);x_bar;x1((i+1):end)];
y_new = [y1(1:i);y_bar;y1((i+1):end)];


%using 9 points
x2 = data([1:9],1);
y2 = data([1:9],2);
%evaluating the tridiagonal matrix
k2 = matrix_eval(x2,y2);
%evaluating y_bar using cubic spline interpolation
y_bar2 = cub_spl(x2,y2,x_bar,k2)
%findind x_bar in range of x_data
i2 = search_x(x2,x_bar);
%inserting x_bar and the corresponding y-bar in the x_data and y_data
x_new2 = [x2(1:i2);x_bar;x2((i2+1):end)];
y_new2 = [y2(1:i2);y_bar2;y2((i2+1):end)];

%using 12 points
x3 = data([1:12],1);
y3 = data([1:12],2);
%evaluating the tridiagonal matrix
k3 = matrix_eval(x3,y3);
%evaluating y_bar using cubic spline interpolation
y_bar3 = cub_spl(x3,y3,x_bar,k3)
%findind x_bar in range of x_data
i3 = search_x(x3,x_bar);
%inserting x_bar and the corresponding y-bar in the x_data and y_data
x_new3 = [x3(1:i3);x_bar;x3((i3+1):end)];
y_new3 = [y3(1:i3);y_bar3;y3((i3+1):end)];

%plotting
plot(x,y,'g--')
hold on
%plotting x and y using 6+1 points
plot(x_new, y_new,'r--')
hold on
%plotting x and y using 9+1 points
plot(x_new2, y_new2,'k-*')
hold on
%plotting x and y using 12+1 points
plot(x_new3, y_new3,'b')
xlabel('x');
ylabel('y');
legend({'data file','cubspline-6points','cubspline-9points','cubspline-
12points'},'location','northeast')
hold off
```

```
%function to return the position of x_bar in the x_data
function x_position = search_x(x_data,x_bar)
for i = 1:size(x_data)
    if (x_bar > x_data(i)) && (x_bar < x_data(i+1))
        x = [x_data(i);x_data(i+1)];
        x_position = i;
    end
end

%function to evaluate y_bar using cubic spline interpolation
function y_cub = cub_spl(x,y,x_bar,k)
%function call to return the position of x_bar within x_data
x_position = search_x(x,x_bar);
i = x_position;
%y_bar evaluated using cubic spline formula
y_cub = ((((x_bar - x(i+1))^3)/(x(i)-x(i+1)))-(x_bar-x(i+1))*(x(i)-
x(i+1)))*k(i)/6 - (((x_bar- x(i))^3)/(x(i)-x(i+1))-(x_bar-x(i))*(x(i)-
x(i+1)))*k(i+1)/6 + (y(i)*(x_bar-x(i+1))-y(i+1)*(x_bar-x(i)))/(x(i)-x(i+1));
end

%function to return the tridiagonal matrix
function k = matrix_eval(x,y)
[n,m] = size(x);
d = [0;0;0;0];
for i = 2 : (n-1)
    %a is the co-effcicient of y''(i-1)
    a(i)= x(i-1)-x(i);
    %b is the co-efficient of y''(i)
    b(i-1) = 2*(x(i-1)-x(i+1));
    %c is the co-efficient of y''(i-1)
    c(i-1) = x(i)-x(i+1);
    %d is the value of the sum of coefficients and the derivatives
    d(i-1) = 6*(((y(i-1)-y(i))/(x(i-1)-x(i)))-((y(i)-y(i+1))/(x(i)-x(i+1))));
end
%constructing the tridiagonal matrix
A = zeros(n-2);
%diahonal elements = b
for i = 1:n-2
    A(i,i) = b(i);
end
for i = 1:n-3
    %c ranges from row 1 to row n-3
    A(i,i+1)=c(i);
    %a ranges from row 2 to n-2
    A(i+1,i)=a(i+1);
end
%evaluating the derivatives
k = zeros(n,1);
%for continuity, the derivatives at first and last points are zero
k(1) = 0;
k(n)= 0;
%evaluating derivatives
k([2:n-1],1)= inv(A)*d;
end
```

REFERENCES:

1. http://people.math.sfu.ca/~kevmitch/teaching/316-09.05/neville.pdf
2. Numerical methods in engineering using MATLAB, Jaan Kiusalaas