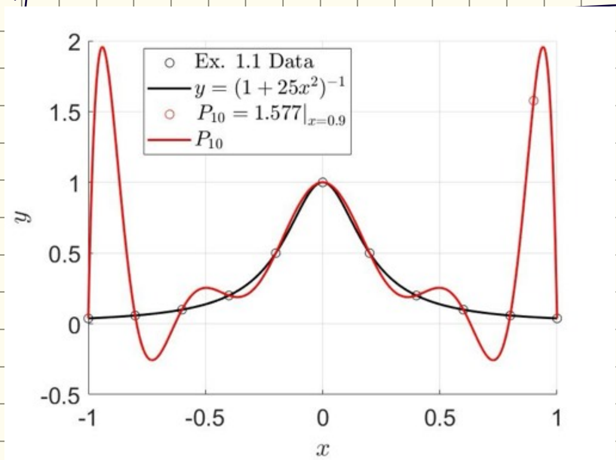
Benjamin Bemis Numerical Method's HW1 AME 60614

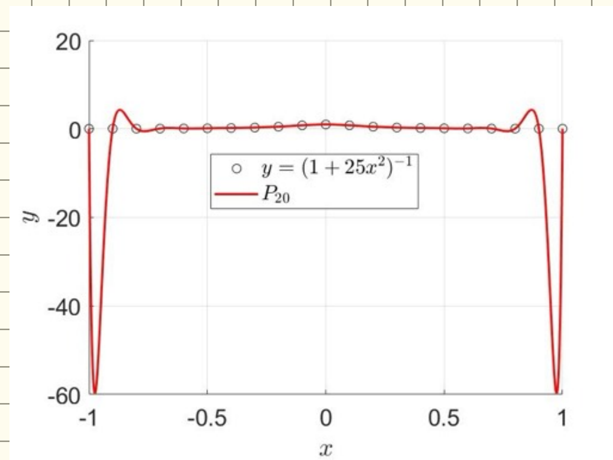
1.) Build a Lagrange Interpolation Computer program.

Part a.) See Attached Matlab Script and detailed explanation of the Solution to the following eqn.

$$P_n(x) = \sum_{j=0}^{n} y_j L_j(x) \, , \quad L_j = \prod_{\substack{i=0 \\ i\neq j}}^{n} \frac{x-x_i}{x_j-x_i} \Bigg|_{\substack{x=0.9 \\ n=10}} = \boxed{1.557}$$

Part b.) $\boxed{\text{Comments on Results are in the Script.}}$




2.) Build a Newtonian Interpolation Computer program.

Part a.) See Attached Matlab Script and detailed explanation of the Solution to the following eqn.

$$P_n(x) = f(x_0) + \sum_{j=0}^{n} C_j \prod_{i=0}^{j-1} (x-x_i)$$

$$C_j = \sum_{i=0}^{j} \frac{f(x_i)}{\prod_{\substack{k=0 \\ k\neq i}}^{j} x_i - x_k} \, , \quad j = 1,2,3\ldots,n$$

| 1st | 2nd | 3rd | 4th | 5th | 6th |
|---|---|---|---|---|---|
| 0.048 | 0.037 | 0.0587 | 0.0144 | -0.0245 | 0.2359 |

3.) Quadratic Spline

a.) There are 3 unknowns so there needs to be 3 Joint conditions.

Function values: $\boxed{\begin{aligned} S_i(x_i) &= y_i \\ S_i(x_{i+1}) &= y_{i+1} \end{aligned}}$

Continuity of $S'$ must be enforced

$$\boxed{S'_{i+1}(x_i) = S'_i(x_i)}$$

Since $S$ is piecewise quadratic, $S'$ is piecewise linear

b.)

From Lagrange Polynomial

$$S_i'(x) = S'(x_{i+1}) \frac{x - x_i}{x_{i+1} - x_i} + S'(x_i) \frac{x_{i+1} - x}{x_{i+1} - x_i} = \frac{S'(x_{i+1}) - S'(x_i)}{x_{i+1} - x_i}(x - x_i) + S'(x_i)$$

Integrate once

$$S_i(x) = S'(x_i) \frac{(x - x_{i+1})^2}{2(x_i - x_{i+1})} + S'(x_{i+1}) \frac{(x - x_i)^2}{2(x_{i+1} - x_i)} + y_i$$

If we let $x = x_{i+1}$

$$y_{i+1} = \frac{S'(x_{i+1}) - S'(x_i)}{2(x_{i+1} - x_i)}(x_{i+1} - x_i)^2 + S'(x_i)(x_{i+1} - x_i) + y_i$$

$$S'(x_{i+1}) = 2\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - S'(x_i)$$

Conditions for $i = 0$ & $n$

(a) $S'(x_0) = S'(x_n) = 0$ Free run out condition

(b) $S'(x_0) = S'(x_1)$ parabolic runout

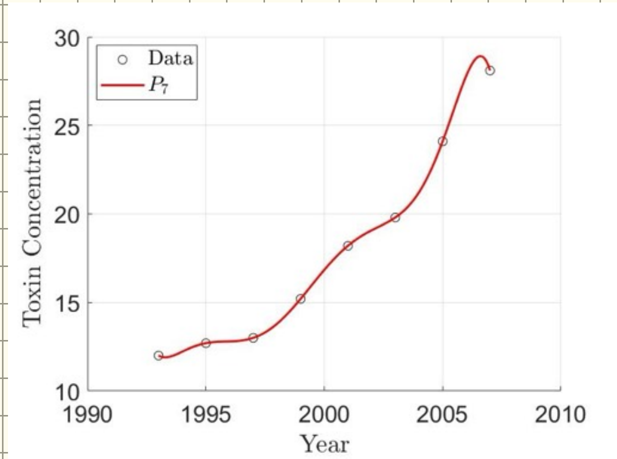
(c) $S'(x_0) = S'(x_n)$ Periodic runout

(c.) Due to the lack of the need to calculate the Second derivative, the quadratic spline is simpler and numerically easier to Solve than the cubic spline.

$\therefore$ Due to Simpler equations and fewer parameters the quadratic Spline is Numerically Cheaper.

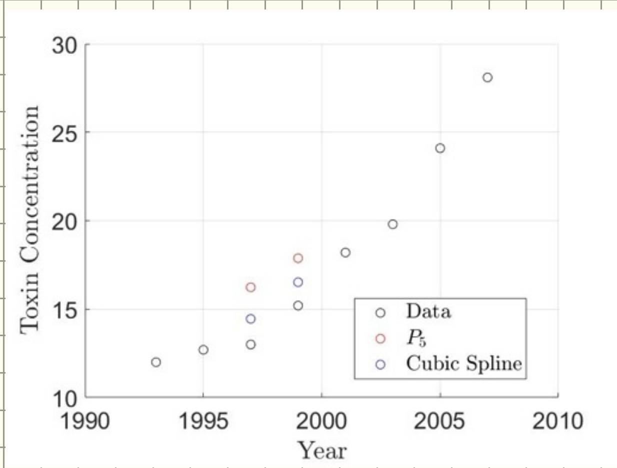
4.) Using data provided: A $7^{th}$ order Lagrange polynomial was used to Approximate
a.) the toxin concentration in 2009.
This resulted in a concentration of $\boxed{-38.4}$

Comments on Results are in the Script




b & c

Comments on Results are in the Script




| Year | (n=5) Lagrange | Spline | Original |
|---|---|---|---|
| 1997 | 16.233 | 14.45 | 13 |
| 1999 | 17.88 | 16.52 | 15.2 |

5.) $P_n(x) = \sum_{j=0}^{n} L_j(x) f(x_j)$ with 3 points $n=2$

$P_n(x)$ $L_0 y_0 + L_1 y_1 + L_2 y_2$

$$L_2 = f(x_{i-1}) \frac{(x-x_i)(x-x_{i-1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} + f(x_i)\frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} + f(x_{i+1})\frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}$$

Simplifying for $\Delta = x_i - x_{i-1}$ and $2\Delta = x_{i+1} - x_{i-1}$

$$L_2 = f(x_{i-1})\frac{(x-x_i)(x-x_{i-1})}{2\Delta^2} - f(x_i)\frac{(x-x_{i-1})(x-x_{i+1})}{\Delta^2} + f(x_{i+1})\frac{(x-x_{i-1})(x-x_i)}{2\Delta^2}$$

$$L_2' = f(x_{i-1})\frac{1}{2\Delta^2}\left((x-x_{i-1})+(x-x_i)\right) - f(x_i)\frac{1}{\Delta^2}\left((x-x_{i+1})+(x-x_{i-1})\right) + f(x_{i+1})\frac{1}{2\Delta^2}\left((x-x_i)+(x-x_{i-1})\right)$$

$$\boxed{L_2'' = \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{\Delta^2}}$$

### Contents



```
%{

@author: Benjamin Bemis Ph.D Student,
Advisor: Dr Juliano


Description:
AME 60614: Numerical Methods
Homework: 1
Due: 9/10/2024


%}
```

### Preparation of the Workspace

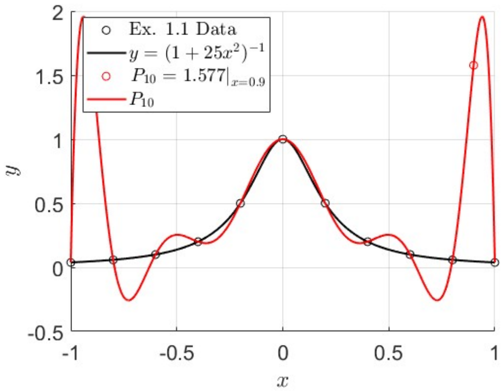
```
clear all
clc
close all
```

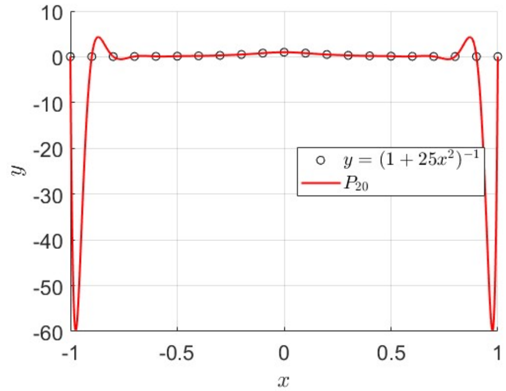
### Preperation of Figures

```
fontsize = 16;
set(0,'DefaultTextInterpreter','latex')
set(0,'DefaultAxesFontSize',fontsize)
set(0,'DefaultLegendFontSize',fontsize)
colors  = ["#000000","#1b9e77","#d95f02","#7570b3","#0099FF","#FF0000"];
```

### Problem 1

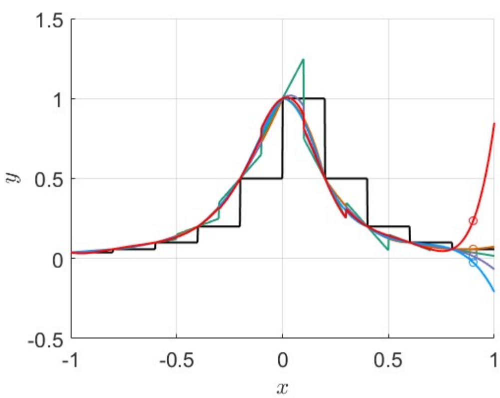
```
% Part a
x = linspace(-1,1,11);
y = [0.038 0.058 0.1 0.2 0.5 1 .5 .2 .1 .058 0.038];  % provided data from ex. 1.1

x_smooth = linspace(min(x),max(x),1e3);
y_exact = 1./(1+25*x_smooth.^2);
x_interp = 0.9; % point to be interpolated

n = length(x)-1;
query = x_interp;
y_interp = interpl(x,y,n,query) ;

for i = 1:length(x_smooth)

y_interp_smooth(i) = interpl(x,y,n,x_smooth(i));

end


figure
hold on
plot(x,y,'ko')
plot(x_smooth,y_exact,"k","LineWidth",1.5)
plot(x_interp,y_interp,"ro")
plot(x_smooth,y_interp_smooth,"r","LineWidth",1.5)
xlabel('$x$');
ylabel('$y$');
grid on
% xlim([0 1])
set(gca,'fontsize', fontsize)
legend("Ex. 1.1 Data","$y = (1+25x^2)^{-1}$",strcat("$\left. P_{",string(n),"}=",string(round(y_interp,3)),'\right|_{x=',string(query),"}","$"),strcat("$ P_{",string(n),"}$"),'Interpreter','Latex','location','best')

% ========================================================================
% part b
x_b = linspace(-1,1,21);
y_exact_b = 1./(1+25*x_b.^2);
n = length(x_b)-1;

for i = 1:length(x_smooth)

y_interp_smooth_b(i) = interpl(x_b,y_exact_b,n,x_smooth(i));

end
figure
hold on
plot(x_b,y_exact_b,'ko')
plot(x_smooth,y_interp_smooth_b,"r","LineWidth",1.5)
xlabel('$x$');
ylabel('$y$');
grid on
% xlim([0 1])
set(gca,'fontsize', fontsize)
legend("$y = (1+25x^2)^{-1}$",strcat("$ P_{",string(n),"}$"),'Interpreter','Latex','location','best')

%{
Comments:
This is functionally similar to the plot in example 1.1. The main
difference is that as the order of the polynomial increases so do the
magnitude of the occilations. This is why the extremes of my interpolation
are much larger than those shown in example 1.1.

%}
```




**Problem 2**

```
% Part a
x = linspace(-1,1,11);
y = [0.038 0.058 0.1 0.2 0.5 1 .5 .2 .1 .058 0.038];  % provided data from ex. 1.1

x_smooth = linspace(min(x),max(x),1e3);
y_exact = 1./(1+25*x_smooth.^2);
x_interp = 0.9; % point to be interpolated


query = x_interp;
for i = 1:6
    n = i;
    y_interp_N(i) = interpN(x,y,n,query);
    y_interp_L(i) = interpl(x,y,n,query);

    for j = 1:length(x_smooth)

        y_interp_smoothN(i,j) = interpN(x,y,n,x_smooth(j));
        y_interp_smoothL(i,j) = interpl(x,y,n,x_smooth(j));

    end

end

figure
hold on
for i = 1:6
plot(x_interp,y_interp_N(i),'o',"Color",colors(i));
% plot(x_interp,y_interp_L(i),'*',"Color",colors(i));
plot(x_smooth,y_interp_smoothN(i,:),"Color",colors(i),"LineWidth",1.5);
% plot(x_smooth,y_interp_smoothL(i,:),"--","Color",colors(i),"LineWidth",1.5);

end
xlabel('$x$');
ylabel('$y$');
grid on
% xlim([0 1])
set(gca,'fontsize', fontsize)
```




**Problem 4**

```
% See paper
```

**Problem 8**

```
% Part a
year = 1993:2:2007;
concen = [12 12.7 13 15.2 18.2 19.8 24.1 28.1];

year_smooth = linspace(min(year),max(year),1e3);

query = 2009; % point to be interpolated

n = length(year)-1;
concen_interp = interpl(year,concen,n,query);

for i = 1:length(year_smooth)
```

```
concen_interp_smooth(i) = interpl(year,concen,n,year_smooth(i));

end

figure
hold on
plot(year,concen,'ko')
plot(year_smooth,concen_interp_smooth,"r","LineWidth",1.5)
xlabel('Year');
ylabel('Toxin Concentration');
grid on
% xlim([0 1])
set(gca,'fontsize', fontsize)
legend("Data",strcat("$ P_{",string(n),"}$"),'Interpreter','Latex','location','best')

%{
Comments:
The prediction of a negative toxin concentration in 2009 is unsensical.
This is because this uses an 7th order polynomial which ocilates wildly
near the end points.
%}

% ======================================================================
% Part b & c

year_b = [year(1:2) year(5:end)];
concen_b = [concen(1:2) concen(5:end)];

query = [1997,1999]; % point to be interpolated

n = length(year_b)-1;
for i = 1:2
concen_interp_b(i) = interpl(year_b,concen_b,n,query(i));
spline_interp_b(i) = interp1(year_b,concen_b,query(i),"spline");
end
disp(concen(3:4))
disp(concen_interp_b)
disp(spline_interp_b)





figure
hold on
plot(year,concen,'ko')
plot(query,concen_interp_b,'ro')
plot(query,spline_interp_b,'bo')
% plot(year_smooth,concen_interp_smooth,"r","LineWidth",1.5)
xlabel('Year');
ylabel('Toxin Concentration');
grid on
% xlim([0 1])
set(gca,'fontsize', fontsize)
legend("Data",strcat("$ P_{",string(n),"}$"),"Cubic Spline",'Interpreter','Latex','location','best')

%{
 Comments on the difference in interpolation using both Lagrangian and Cubic Spline

Both methods over predict the concentration of toxin, however the cubic
spline is closest to the original data. This is because the Lagrangian
polynomial fit to the data set is of 5th order. This is a rather large
polynomial for interpolation and thus causes some ocilations near the
edges of the data set. For example, in this data set the value of the third
and fourth indexes are 3.23 and 2.68 respectively. This is because the
accuracy decays as we move farther from the center point. The same is true
for the cubic polynomial but is less pronounced, resulting in a closer
interpolation to the original data.



%}
```

### Problem 9

```
% Derivation - See paper
```

### Functions

```
function value = interpl(x,y,n,query)
%{
Lagrangian Interpolation

input:
x is a vector of independent points
y is a vector of points dependent on x
n is the order of the polynomial interpolation
query is the independent value to be interpolated

output:

value is the interpolated estimate of query

%}

% nearest n+1 points are needed
indices = zeros(1, n+1);
x_dummy = x;

for k = 1:n+1
    [dummy, index] = min(abs(query -  x_dummy));
    x_close(k) = x(index);
    y_close(k) = y(index);
    indices(k) = index;
    x_dummy(index) = nan;
end

% Sort indices to match the original order
[~, sortOrder] = sort(indices);
x_close = x_close(sortOrder);
y_close = y_close(sortOrder);
```

```
for j = 1:n+1
    for i = 1:n+1
        if i == j
            L_num(i) = nan;
            L_den(i) = nan;
        else
            L_num(i) = query-(x_close(i));
            L_den(i) = x_close(j)-x_close(i);
        end
    end
    L(j) = (prod(L_num(~isnan(L_num)))/prod(L_den(~isnan(L_den))));
end

value = sum(y_close.*L);

end


function value = interpN(x,y,n,query)
%{
Newtonian Interpolation

input:
x is a vector of independent points
y is a vector of points dependent on x
n is the order of the polynomial interpolation
query is the independent value to be interpolated

output:

value is the interpolated estimate of query

%}

% nearest n+1 points are needed
indices = zeros(1, n+1);
x_dummy = x;

for k = 1:n+1
    [dummy, index] = min(abs(query - x_dummy));
    x_close(k) = x(index);
    y_close(k) = y(index);
    indices(k) = index;
    x_dummy(index) = nan;
end

% Sort indices to match the original order
[~, sortOrder] = sort(indices);
x_close = x_close(sortOrder);
y_close = y_close(sortOrder);

% Construct the divided differences table
div_diff = zeros(n+1, n+1);
div_diff(:,1) = y_close';

for j = 2:n+1
    for i = 1:n-j+1
        div_diff(i,j) = (div_diff(i+1,j-1) - div_diff(i,j-1)) / (x_close(i+j-1) - x_close(i));
    end
end

% Perform the interpolation
value = div_diff(1,1);
prod_term = 1;

for i = 1:n
    prod_term = prod_term * (query - x_close(i));
    value = value + div_diff(1,i+1) * prod_term;
end

end
```