Homework 5

# Section 3.1

## Problem 12.

Use the Lagrange interpolation polynomial of degree three or less and four-digit chopping arithmetic to approximate using the following values. Find an error bound for the approximation.

The actual value of is (to four decimal places). Explain the discrepancy between the actual error and the error bounds.

Using the Lagrange method:

Where:

Evaluating the polynomials:

Now for multiplication:

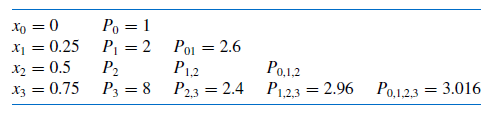
The error that I got was . The error that we are supposed to get is:

The reason for this discrepancy is the fact that we used four-digit arithmetic and if we kept more digits, then we will get a better error.

# Section 3.2

## Problem 5.

Neville’s method is used to approximate , given the following table:



Determine .

# Section 3.3

## Problem 2.

Use Eq. (3.10) or Algorithm 3.2 to construct interpolating polynomials of degree one, two, and three for the following data. Approximate the specific value using each of the polynomials.

Equation 3.10:

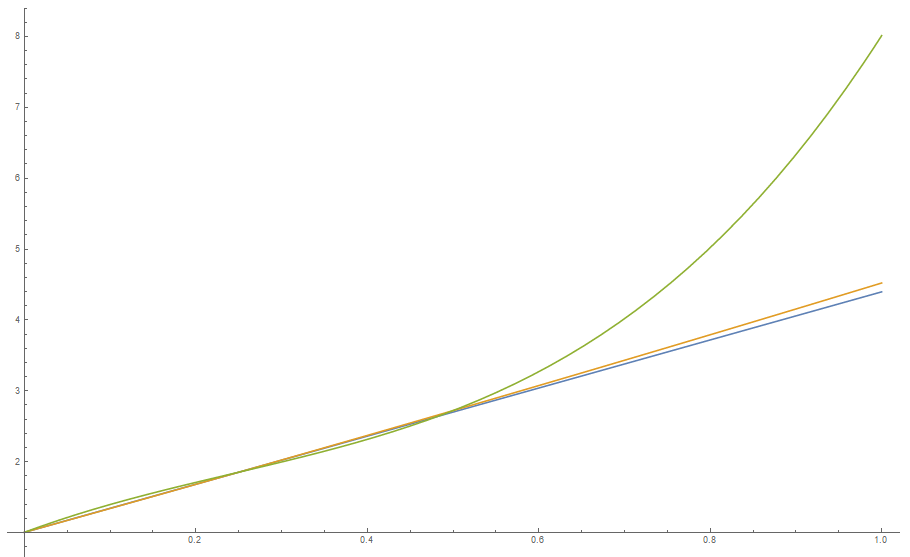
1st order:

2nd order:

3rd order:

Using Mathematica to solve for the polynomials at the point :

Graphing these polynomials.



## Problem 3

Use Newton the forward-difference formula to construct interpolating polynomials of degree one, two and three for the following data. Approximate the specified value using each of the polynomials.

Now using a program:

#include <iostream>

using namespace std;

//This is the function of the two terms.

double F(int n, double array[], double array2[])

{

double fx = 0;

fx = (array2[n] - array2[n - 1]) / (array[n] - array[n - 1]);

return fx;

}

//This is the product portion of the formula.

double Product(int n, double x, double array[])

{

double XX = 1;

int i = 0;

while (n > 0)

{

XX = XX \* (x - array[i]);

i += 1;

n -= 1;

}

return XX;

}

int main()

{

//These are the variables.

const int Size = 4;

double Xn[Size] = { -.75,-.50,-.25,0 };

double Fn[Size] = { -.07181250,-.02475000,.33493750,1.10100000 };

double F1 = 0, F2 = 0, F3T = 0, F3B = 0, F3 = 0;

double P1 = 0, P2 = 0, P3 = 0;

//This is the value and where the solution will be in.

double x = -.3333333333, fx = 0;

//Algorithm for the Polynomials.

F1 = F(1, Xn, Fn);

F2 = (F(2, Xn, Fn) - F1) / (Xn[2] - Xn[0]);

F3T = ((F(3, Xn, Fn) - F(2, Xn, Fn)) / (Xn[3] - Xn[1]))

- F2;

F3B = Xn[3] - Xn[0];

F3 = F3T / F3B;

//The Polynomials are now.

P1 = Fn[0] +( F1 \* Product(1, x, Xn));

P2 = P1 + F2 \* Product(2, x, Xn);

P3 = P2 + F3 \* Product(3, x, Xn);

//Output the solutions.

cout << "The first order polynomial evaluated at x = " << x << " is " << P1 << "." << endl;

cout << "The second order polynomial evaluated at x = " << x << " is " << P2 << "." << endl;

cout << "The third order polynomial evaluated at x = " << x << " is " << P3 << "." << endl;

return 0;

}

We get that:



## Problem 18.

1. The introduction to this chapter included a tale listing the population of the United States from 1950 to 2000. Use approximate divided difference to approximate the population in the years 1940, 1975, and 2020.

Using the data points:



Using a Program:

#include <iostream>

using namespace std;

//This is the function of the two terms.

double F(int n, double array[], double array2[])

{

double fx = 0;

fx = (array2[n] - array2[n - 1]) / (array[n] - array[n - 1]);

return fx;

}

//This is the product portion of the formula.

double Product(int n, double x, double array[])

{

double XX = 1;

int i = 0;

while (n > 0)

{

XX = XX \* (x - array[i]);

i += 1;

n -= 1;

}

return XX;

}

int main()

{

//These are the variables.

const int Size = 6;

double Xn[Size] = { 1950, 1960, 1970, 1980, 1990, 2000 };

double Fn[Size] = { 151326, 179323, 203302, 226542, 249633, 281422 };

double F1 = 0, F2 = 0, F3 = 0,

F4 = 0, F5 = 0, F6 = 0;

double P1 = 0, P2 = 0, P3 = 0, P4 = 0, P5 = 0, P6 = 0;

//This is the value and where the solution will be in.

double x = 1940, fx = 0;

//Algorithm for the Polynomials.

F1 = F(1, Xn, Fn);

//F2.

F2 = (F(2, Xn, Fn) - F1) / (Xn[2] - Xn[0]);

//F3

double F123 = ((F(3, Xn, Fn) - F(2, Xn, Fn)) / (Xn[3] - Xn[1]));

F3 = (F123 - F2) / (Xn[3] - Xn[0]);

//F4.

double F234 = (F(4, Xn, Fn) - F(3, Xn, Fn)) / (Xn[4] - Xn[2]);

double F1234 = (F234 - F123) / (Xn[4] - Xn[1]);

F4 = (F1234 - F3) / (Xn[4] - Xn[0]);

//F5.

double F345 = (F(5, Xn, Fn) - F(4, Xn, Fn)) / (Xn[4] - Xn[2]);

double F2345 = (F345 - F234) / (Xn[5] - Xn[2]);

double F12345 = (F2345 - F1234) / (Xn[5] - Xn[1]);

F5 = (F12345 - F4) / (Xn[5] - Xn[0]);

//F6.

double F456 = (F(6, Xn, Fn) - F(5, Xn, Fn)) / (Xn[6] - Xn[4]);

double F3456 = (F456 - F345) / (Xn[6] - Xn[3]);

double F23456 = (F3456 - F2345) / (Xn[6] - Xn[2]);

double F123456 = (F23456 - F12345) / (Xn[6] - Xn[1]);

F6 = (F123456 - F5) / (Xn[6] - Xn[0]);

//The Polynomials are now.

P1 = Fn[0] + F1 \* Product(1, x, Xn);

P2 = P1 + F2 \* Product(2, x, Xn);

P3 = P2 + F3 \* Product(3, x, Xn);

P4 = P3 + F4 \* Product(4, x, Xn);

P5 = P4 + F5 \* Product(5, x, Xn);

P6 = P5 + F6 \* Product(6, x, Xn);

//Output the solutions.

cout << "The first order polynomial evaluated at x = " << x << " is " << P1 << "." << endl;

cout << "The second order polynomial evaluated at x = " << x << " is " << P2 << "." << endl;

cout << "The third order polynomial evaluated at x = " << x << " is " << P3 << "." << endl;

cout << "The fourth order polynomial evaluated at x = " << x << " is " << P4 << "." << endl;

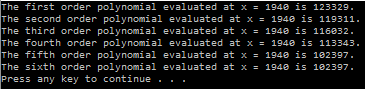
cout << "The fifth order polynomial evaluated at x = " << x << " is " << P5 << "." << endl;

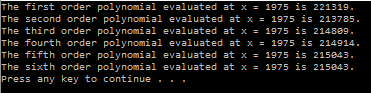
cout << "The sixth order polynomial evaluated at x = " << x << " is " << P6 << "." << endl;

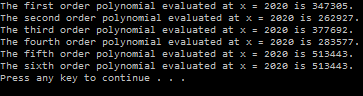
return 0;

}

Displaying the results when is changed.







1. The population in 1940 was approximately 132,165,000. How accurate do you think your 1975 and 2020 figures are?

Using the value that we got with the sixth order polynomial:

The errors are:

Since , we are sure that the value that we interpolated is going to be very close to our value.

Since 2020 is away from our range, our polynomial is most likely far from the correct amount.