Homework 1.3

# Problem 1

1. Use three-digit chopping arithmetic to compute the sum first by and then by . Which method is more accurate, and why?

|  |  |  |  |
| --- | --- | --- | --- |
| Fraction | Decimal | Adding Down | Adding Up |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Using Wolfram Alpha, we get that the correct answer is:

The reason why adding the numbers from smallest to biggest was more accurate than adding from biggest to smallest was because when we started from 1, as we went down and got smaller numbers, their accuracy was decreased since the large number took all the digits for itself. Going down to up, the small numbers were able to keep more of their digits through the process.

1. Write an algorithm to sum the finite series in reverse order.

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int N = 10;

double Sum = 0, x\_j = 0;

//To have it start at the last term, introduce a new variable where it will go from N to 1.

int j = N;

for (int i = 1; i <= N; i++)

{

//Define the function

x\_j = 1 / (pow(j, 2));

Sum = Sum + x\_j;

//Here the counter will be adjusted after the sum so that it can be used for the next iteration.

j = j - 1;

}

cout << "The sum of x\_i from 1 to " << N << " being added backwards is " << Sum << endl;

}

# Problem 2.

The number is defined by Where for and . Use for-digit chopping arithmetic to compute the following approximation to , and determine the absolute and relative errors.

|  |  |  |
| --- | --- | --- |
|  |  | Decimal (4 digit) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Problem 3.

The Maclaurin series for the arctangent function converges for and is given by

1. Use the fact that to determine the number of terms of the series that need to be summed to ensure that .

Using Taylors remainder theorem and noting that

1. The C++ programming language requires the value to be within . How many terms of the series would we need to sum to obtain this degree of accuracy?

Using the same methodology as above

# Problem 4.

Exercise 3 details a rather inefficient means of obtaining an approximation of . The method can be improved substantially by observing that and evaluating the series for the arctangent at and at . Determine the number of terms that must be summed to ensure an approximation to within .

4.

Do not know how to do them.

5.

# Problem 6.

Find the rates of convergence of the following sequences as

Since is the highest error:

# Problem 7.

Find the rates of convergence of the following functions as .

# Problem 8.

1. How many multiplications and additions are required to determine a sum of the form

Consider the following cases:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

So it seems that the number of multiplications is equal to the sum of all natural numbers from 1 to . So the number will be . For additions, the number will be one less addition than multiplication thus the total additions will be

1. Modify the sum in part (a) to an equivalent form that reduces the number of computations.

The amount of multiplications will be reduced down to amount of multiplications. The number of additions will be the sum from to plus . Thus the total amount of additions will be .

# Problem 9.

Let be a polynomial, and let be given. Construct an algorithm to evaluate using nested multiplication.

Using this program, we can adjust the polynomial by changing the array size and it’s entries and we can change the value of so that we can get our solution.

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

//This is where the value of P(x) is going to be stored.

double Px\_0 = 0;

//The coefficients must be inputted from a\_n to a\_0.

//The array size must be one larger than the exponent so that it can contain the constant term.

const int ArraySize = 10;

double Polynomial[ArraySize] = {1,2,3,4,5,6,7,8,9,10};

//This is where the x value will be imputted.

double x0 = 3;

//For the algorithm, we will use a loop

for (int i = 0; i < (ArraySize - 1); i++)

{

Px\_0 = x0 \* (Px\_0 + Polynomial[i]);

}

//Adding the constant term

Px\_0 = Px\_0 + Polynomial[ArraySize-1];

cout << Polynomial[ArraySize - 1];

//Displaying the results

cout << "The value for P(x) = ";

for (int j = 0; j < ArraySize - 1; j++)

{

cout << Polynomial[j] << "\*x^" << ArraySize - 1 - j<< " + ";

}

cout << Polynomial[ArraySize - 1] << "\nevaluated at x0 = " << x0 << " is: ";

cout << Px\_0 << endl;

return 0;

}

# Problem 10.

Equations (1.2) and (1.3) in section 1.2 give alternative formulas for the roots and of . Construct an algorithm with inputs and outputs that computes the roots and (which may be equal to the complex conjugate) using the best formula for each root.

If , we can rewrite the equations to the following:

#include <iostream>

using namespace std;

int main()

{

double a = 1, b = 0, c = 0, discr = 0, x1 = 0, x2 = 0, x1r = 0, x1c = 0, x2r = 0,x2c = 0;

//We shall take in values for the coefficients.

cout << "Please enter the values for a,b, and c" << endl;

cout << "a: ";

cin >> a;

if (a == 0)

{

cout << "THe value a cannot be 0." << endl;

cout << "Please enter the values for a." << endl;

cin >> a;

}

cout << "b: ";

cin >> b;

cout << "c: ";

cin >> c;

cout << endl;

//This will let the user know what the discriminant is and if the roots will be complex.

discr = b \* b - (4 \* a \* c);

cout << "The discriminant is " << discr << endl;

if (discr < 0)

{

cout << "The roots will be complex." << endl;

}

cout << endl;

//Depending on what the discriminant will be, it will decide what kind of solution we will get.

if (discr < 0)

{

x1r = -(2 \* b \* c) / (4 \* a \* c);

x2r = x1r;

x1c = -(2 \* c \* sqrt(-discr)) / (4 \* a \* c);

x2c = -x1c;

}

else if (discr > 0)

{

x1 = -(2 \* c) / (b + sqrt(b \* b - (4 \* a \* c)));

x2 = -(2 \* c) / (b - sqrt(b \* b - (4 \* a \* c)));

}

else

{

x1 = -(b / (2 \* a));

x2 = x1;

}

//Displaying the results.

cout << "The solution to the equation " << a << "\*x^2 + " << b << "\*x + " << c << " = 0 is: " << endl;

if (discr < 0)

{

cout << "x1 = " << x1r << " + " << x1c << "i" << endl;

cout << "x2 = " << x2r << " + " << x2c << "i" << endl;

}

else

{

cout << "x1: " << x1 << endl;

cout << "x2: " << x2 << endl;

}

return 0;

}

# Problem 11.

Constrict an algorithm that has as input an integer , numbers , and a number that produces as output the product .

#include <iostream>

using namespace std;

int main()

{

//This is where the list of numbers will be contained'

//This number can be changed in order to fit however number of elements in the list.

const int ArraySize = 5;

//This is where all the values are going to be set.

double ProductArray[ArraySize] = { 1,2,3,4,5 };

//This is where the x value will be inputted.

double x = 7, Product = 1;

//This loop will perform the product.

for (int i = 0; i < ArraySize; i++)

{

Product = Product \* (x - ProductArray[i]);

}

//This is where the answer will be displayed.

cout << "The value for ";

for (int j = 0; j < ArraySize; j++)

{

cout << "(" << x << " - " << ProductArray[j] << ")";

}

cout << " is: " << Product << endl;

}