Homework 9

# Section 4.5

## Problem 5.

Use Romberg integration to approximate the integrals in Exercise 1 to within . Compute the Romberg table until either , or . Compare your results to the exact value of the integrals.

The algorithm that is used is:

The Trapezoid rule:

Extrapolation:

Using a program:

#include <iostream>

#include <vector>

#include <iomanip> // std::setprecision

using namespace std;

//These are the functions that I will be using.

double h(double, double, int);

double F(double);

double T(double, double, double, int);

double Rkj(double, double, int);

int main()

{

//Pi and our boundary conditions.

const double pi = 3.1415926535897;

double a = 0, b = pi / 4;

double Rkj1 = 0, Rk1j1 = 0;

double tol = 1 / pow(10, 6);

//2D vector that will contain the values that I need.

vector< vector<double> > R;

vector<double> Trap; //Where the trapezoid method values are stored.

//Maximum amount of iterations.

int N = 10;

//define R11

vector<double> row;

int k = 1, j = 1;

R.push\_back(row);

//For the first trapezoid.

Trap.push\_back((h(a, b, k) / 2) \* (F(a) + F(b)));

R[k - 1].push\_back(Trap[k - 1]);

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

{

//every iteration will initialize j back to 1. K will be increased according to the index.

j = 1;

k++;

//Trapezoid.

vector<double> row; //Create new row.

R.push\_back(row); //Add it to the array.

Trap.push\_back(T((R[k - 2][j - 1]), h(a, b, k), a, k));

R[k - 1].push\_back(Trap[k - 1]); //Add the trapezoid value.

//Extrapolation.

for (j = 2; j <= k; j++)

{

Rkj1 = R[k - 1][j - 2], Rk1j1 = R[k - 2][j - 2];

R[k - 1].push\_back(Rkj(Rkj1, Rk1j1, j));

}

//Test to see if the accuracy is met.

if (abs(R[k - 1][k - 1] - R[k - 2][k - 2]) < tol)

{

i = 10;

}

//cout << abs(R[k - 1][k - 1] - R[k - 2][k - 2]) << endl;

}

//Displaying the results.

for (int K = 1; K <= k; K++)

{

cout << K << ": ";

for (int J = 1; J <= K; J++)

{

cout << setprecision(9) << R[K - 1][J - 1] << " ";

}

cout << endl;

}

}

//This function will calculate the h value.

double h(double a, double b, int k)

{

double h = (b - a) / pow(2, k - 1);

return h;

}

//This function will calculate the f(x).

double F(double x)

{

double fx = x \* x \* sin(x);

return fx;

}

//This function performs the Trapezoidal Method.

double T(double Rp, double h, double a, int k)

{

double sum = 0;

for (int i = 1; i <= pow(2, (k - 2)); i++)

{

sum += F(a + (2 \* i - 1) \* h);

}

double R = .5 \* Rp + h \* sum;

return R;

}

//This function creates Rkj.

double Rkj(double Rkj1, double Rk1j1, int j)

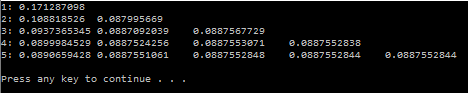
{

double Rkj = Rkj1 + (Rkj1 - Rk1j1) / (pow(4, j - 1) - 1);

return Rkj;

}

We get that the results are:



# Section 4.7

## Problem 3.

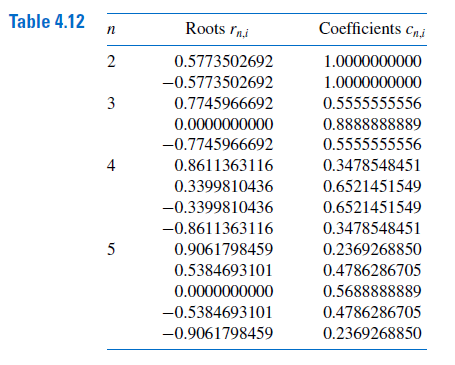
Approximate the following integrals using Gaussian quadrature with , and compare your results to the exact value of the integrals.

An integral of the form

Can be transformed into an integral over via the following transformation:

Since , we know that the polynomial that we are assuming must have order :

Using table 4.12:



We get that the roots should be:

Therefore:

## Problem 5.

Determine constant and that will produce a quadruture formula

That has degree of precision 3.

To get a precision of :

Start by setting the functions equal to the polyniomial to get the solution at the point:

Thus we get a system of equations:

Solving for the coefficients:

Therefore we get that: