University of Antwerp

Scientific Programming

Function Approximation Exercise 4

Armin Halilovic - s0122210

December 4, 2015

Contents

1 Problem

We are given the following set of data function:

$$f(x) = \sqrt{1 + \frac{x}{5}},\ where\ x\ \epsilon\ [-1, 1]$$

We will approximate this function using fourier method:

$$g(t) = \frac{a_0}{2} + a_1\cos(t) + a_2\cos(2t) + ... + a_n\cos(nt) + b_1\sin(t) + b_2\sin(2t) + ... + b_m\sin(nt)$$

This will be done using C++ and the GNU Scientific Library. In section 3, we will describe how we reached each solution, using the most important parts from the code.

2 Using the program

All of the C++ code for the program can be found in the main.cpp and in appendix A of this document. main.cpp comes accompanied by function_approximation.sh, which contains all of the necessary UNIX commands to generate the graph images. This file relies on the graph program in the GNU plotutils package to plot graphs, so make sure that it is installed.

To compile and run the program, execute the following commands in the build/ directory:

```
cmake ..
make
chmod +x ./function_approximation.sh
./function_approximation.bin
```

Do not forget the .bin extension. All of the graphs will be present in the build/images/ directory. If no graphs are present, they can be generated manually by executing function_approximation.sh.
Output files which give some extra information about the solutions can be found in console output and appendix B.1.

3 Solutions

In these solutions, m denotes the amount of data points we will use to approximate $f(x)$ with.

3.1 Calculating the data points

We will generate m equidistant data points in the interval $[-1, 1]$, using the formula $-1 + \frac{2*(i-1)}{m}$.
The array t will hold these points, array x will hold $f(t)$. The points t_i wil then be rescaled to lie in $[-\pi, \pi]$:

```
double t[m], x[m];

for (int i = 1; i <= m; i++) {
    t[i-1] = -1 + (2*(i-1) / (double) (m-1));
    x[i-1] = fx(t[i-1]);
    t[i-1] = M_PI * t[i-1];
}
```

3.2 Calculating the coefficients

To use the approximating formula, we first need to find the coefficients $a_0, a_1, b_1, a_2, b_2, ...$ The formulas for the coefficients are:

$$a_j = \frac{2}{m} \sum_{k=0}^{m-1} x_k \cos(j * t_k), \quad b_j = \frac{2}{m} \sum_{k=0}^{m-1} x_k \sin(j * t_k)$$

where $j = 0, ..., n$ and x_k are the given data points we work with. In the code, this is done by the function "calcCoeff":

```
double calcCoeff(double *t, double *x, bool a, int j, int m) {
    double result = 0;

    for (int k = 0; k < m; k++) {
        result += x[k] * ((a) ? gsl_sf_cos(j*t[k]) : gsl_sf_sin(j*t[k]));
    }

    result *= 2.0/(double) m;
    return result;
}
```

3.3 Approximating the function

We can now fill in and use the function

$$g(t) = \frac{a_0}{2} + a_1\cos(t) + a_2\cos(2t) + ... + a_n\cos(nt) + b_1\sin(t) + b_2\sin(2t) + ... + b_m\sin(nt)$$

to approximate $f(x)$:

```
for (double t = -M_PI; t < M_PI; t = t + 0.01) {
    double y = coeff[0]; //y = a_0/2
    for (int i = 0; i < n; i++) {
        int j = 2*i+1;
        y += coeff[j]   * gsl_sf_cos((i+1)*t);
        y += coeff[j+1] * gsl_sf_sin((i+1)*t);
    }
}
```

The approximation functions $f(t)$ can be found in Appendix B.1. The graphs that are generated can be found in Appendix B.2 and B.3. The graphs in Appendix B.3 are approximations for $f(x) = |x|$.

We notice in both cases that, generally, the approximation becomes more accurate with an increase in m or n. This is to be expected with fourier methods.

Appendices

A Code

A.1 main.cpp

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <algorithm>
#include <gsl/gsl_sf_trig.h>

void printArray(const double *x, const int m) {
    for (int i = 0; i < m; i++) {
        std::cout << x[i];
        if (i != m-1) std::cout << ", ";
        else std::cout << std::endl;
    }
}

double fx(double x) {
    //return (x > 0) ? x : -x;
    return sqrt( 1 + (x/5.0) );
}

double calcCoeff(double *t, double *x, bool a, int j, int m) {
    double result = 0;

    for (int k = 0; k < m; k++) {
        result += x[k] * ((a) ? gsl_sf_cos(j * t[k]) : gsl_sf_sin(j * t[k]));
    }

    result *= 2.0/(double) m;
    return result;
}

void solveForN(int n, double *t, double *x, int m, std::ofstream &out) {
    double coeff[2 * n + 1];
    coeff[0] = calcCoeff(t, x, true, 0, m)/2;

    //Generate coefficients
    for (int i = 0; i < n; i++) {
        int j = 2*i+1;
        coeff[j] = calcCoeff(t, x, true, i+1, m);
        coeff[j+1] = calcCoeff(t, x, false, i+1, m);

        if (i == n-1 and n == m/2) {
            coeff[j] /= 2;
        }
    }

    //Print out the approximation function
    std::cout << "f(x) = " << coeff[0] << " + " ;
    for (int i = 0; i < n; i++) {
        int j = 2*i+1;
        std::cout << coeff[j] << "*cos(" << i+1 << "x) + ";
        std::cout << coeff[j+1] << "*sin(" << i+1 << "x)";
        std::cout << ((i != n-1) ? " + " : "\n");
    }
```

```
    //Generate approximated values
    for (double xx = -M_PI; xx < M_PI; xx = xx + 0.01) {
        double y = coeff[0];
        for (int i = 0; i < n; i++) {
            int j = 2*i+1;
            y += coeff[j]   * gsl_sf_cos ((i+1)*xx);
            y += coeff[j+1] * gsl_sf_sin ((i+1)*xx);
        }

        out << xx << " " << y << std::endl;
    }

}

void doExercise(int const m, int n) {
    //t = equidistant points, x = f(t)
    double t[m], x[m];

    std::stringstream mm, nn;
    mm << m;  nn << n;
    std::ofstream out;
    out.open("images/m" + mm.str() + "n" + nn.str() + ".dat");
    out << "#m=0,S=2\n";
    for (int i = 1; i <= m; i++) {
        t[i-1] = -1 + (2*(i-1) / (double) (m-1));
        x[i-1] = fx(t[i-1]);
        t[i-1] = M_PI * t[i-1];
        out << t[i-1] << " " << x[i-1] << std::endl;
    }

    std::cout << "m = " << m << " equidistant points as time values in [-1, 1] rescaled to
     [-pi,pi]: \n\t";
    printArray(t, m);

    out << "\n#m=1,S=0\n";
    for (double xx = -M_PI; xx < M_PI; xx = xx + 0.01) {
        out << xx << " " << fx(xx / M_PI) << std::endl;
    }

    out << "#m=1,S=0\n";
    std::cout << "n = " << n << " approximation: \n\t";
    solveForN(n, t, x, m, out);
    std::cout << std::endl;

    out.close();
}

int main (int argc, char *argv[]) {
    std::ofstream fxx;
    fxx.open("images/fx.dat");
    for (double xx = -M_PI; xx < M_PI; xx = xx + 0.01) {
        fxx << xx << " " << fx(xx) << std::endl;
    }

    doExercise(3, 1);
    doExercise(5, 1);
    doExercise(10, 1);
    doExercise(25, 1);
    doExercise(3, 3);
    doExercise(5, 3);
    doExercise(10, 3);
```

```
    doExercise(25, 3);

    fxx.close();
    system("./function_approximation.sh");
    return 0;
}
```

A.2 function_approximation.sh

```
#!/bin/bash

DIR="images"

#rm images/*.png

for f in images/*.dat
do
    graph -T png -F HersheySans -L ${f:7:-4} --bitmap-size 820x820 < "$f" >
     ${f::-4}".png"
done

#rm images/*.dat
```

B Output

B.1 Console output

```
m = 3 equidistant points as time values in [−1, 1] rescaled to [−pi,pi]:
    −3.14159, 0, 3.14159
n = 1 approximation:
    f(x) = 0.996624 + −0.329957∗cos(1x) + 1.64117e−17∗sin(1x)

m = 5 equidistant points as time values in [−1, 1] rescaled to [−pi,pi]:
    −3.14159, −1.5708, 0, 1.5708, 3.14159
n = 1 approximation:
    f(x) = 0.997473 + −0.395949∗cos(1x) + 0.0400502∗sin(1x)

m = 10 equidistant points as time values in [−1, 1] rescaled to [−pi,pi]:
    −3.14159, −2.44346, −1.74533, −1.0472, −0.349066, 0.349066, 1.0472, 1.74533, 2.44346,
      3.14159
n = 1 approximation:
    f(x) = 0.997944 + −0.197067∗cos(1x) + 0.0550512∗sin(1x)

m = 25 equidistant points as time values in [−1, 1] rescaled to [−pi,pi]:
    −3.14159, −2.87979, −2.61799, −2.35619, −2.0944, −1.8326, −1.5708, −1.309, −1.0472,
      −0.785398, −0.523599, −0.261799, 0, 0.261799, 0.523599, 0.785398, 1.0472, 1.309, 1.5708,
      1.8326, 2.0944, 2.35619, 2.61799, 2.87979, 3.14159
n = 1 approximation:
    f(x) = 0.99818 + −0.0776187∗cos(1x) + 0.0608852∗sin(1x)

m = 3 equidistant points as time values in [−1, 1] rescaled to [−pi,pi]:
    −3.14159, 0, 3.14159
n = 3 approximation:
    f(x) = 0.996624 + −0.659915∗cos(1x) + 1.64117e−17∗sin(1x) + 1.99325∗cos(2x) +
      −3.28235e−17∗sin(2x) + −0.659915∗cos(3x) + 4.92352e−17∗sin(3x)

m = 5 equidistant points as time values in [−1, 1] rescaled to [−pi,pi]:
    −3.14159, −1.5708, 0, 1.5708, 3.14159
n = 3 approximation:
    f(x) = 0.997473 + −0.395949∗cos(1x) + 0.0400502∗sin(1x) + 0.396952∗cos(2x) +
      −1.47893e−17∗sin(2x) + −0.395949∗cos(3x) + −0.0400502∗sin(3x)

m = 10 equidistant points as time values in [−1, 1] rescaled to [−pi,pi]:
    −3.14159, −2.44346, −1.74533, −1.0472, −0.349066, 0.349066, 1.0472, 1.74533, 2.44346,
      3.14159
n = 3 approximation:
    f(x) = 0.997944 + −0.197067∗cos(1x) + 0.0550512∗sin(1x) + 0.198437∗cos(2x) +
      −0.0239339∗sin(2x) + −0.198684∗cos(3x) + 0.0115998∗sin(3x)

m = 25 equidistant points as time values in [−1, 1] rescaled to [−pi,pi]:
    −3.14159, −2.87979, −2.61799, −2.35619, −2.0944, −1.8326, −1.5708, −1.309, −1.0472,
      −0.785398, −0.523599, −0.261799, 0, 0.261799, 0.523599, 0.785398, 1.0472, 1.309, 1.5708,
      1.8326, 2.0944, 2.35619, 2.61799, 2.87979, 3.14159
n = 3 approximation:
    f(x) = 0.99818 + −0.0776187∗cos(1x) + 0.0608852∗sin(1x) + 0.0790865∗cos(2x) +
      −0.029984∗sin(2x) + −0.0793618∗cos(3x) + 0.0194047∗sin(3x)
```

B.2 $f(x) = \sqrt{1 + \frac{x}{5}}$

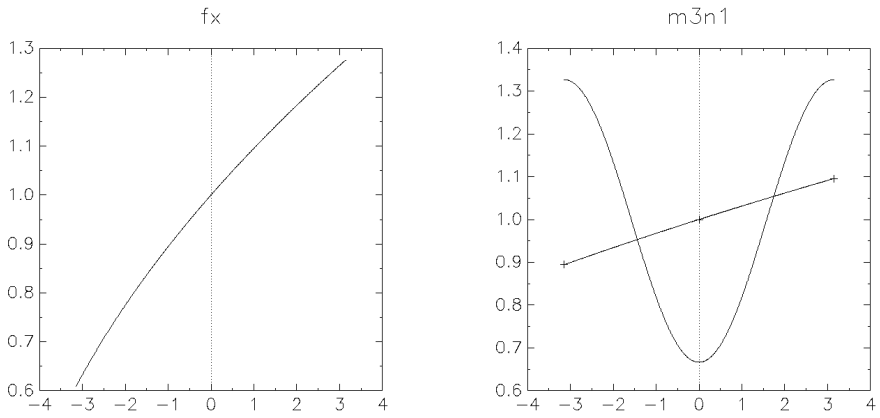


Figure 1: left: f(x), right: approximation where m=3 and n=1

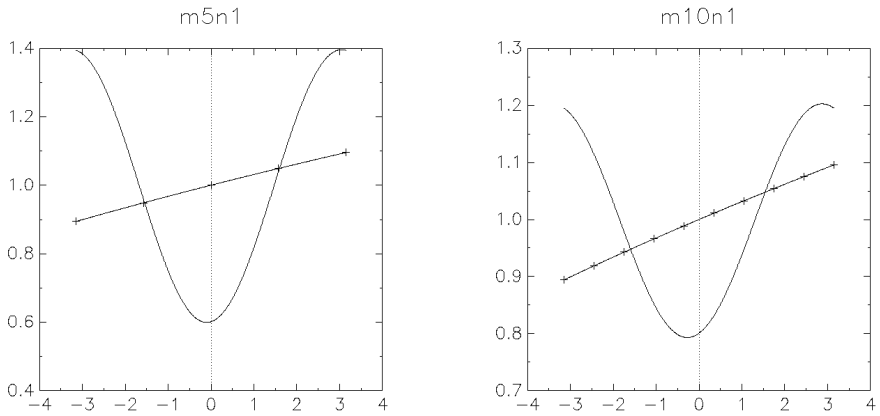


Figure 2: approximations where m=5, n=1, and m=10, n=1

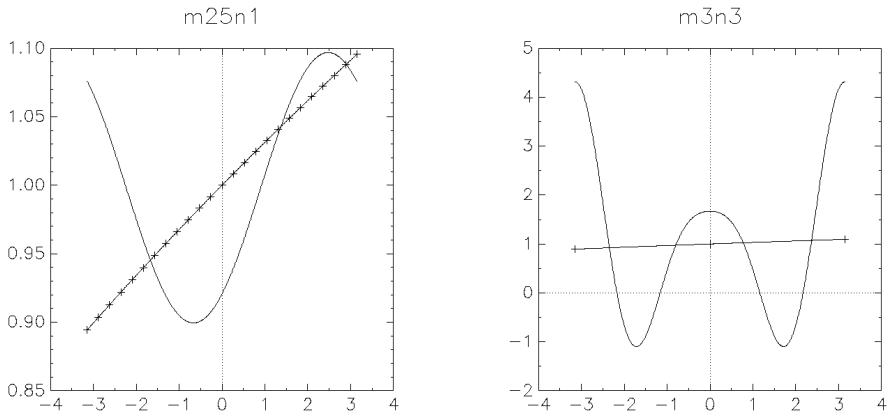


Figure 3: approximations where m=25, n=1, and m=3, n=3

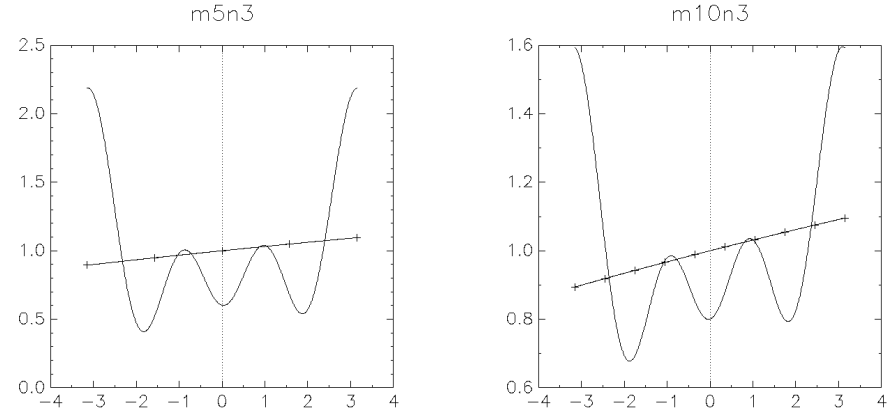


Figure 4: approximations where m=5, n=3, and m=10, n=3

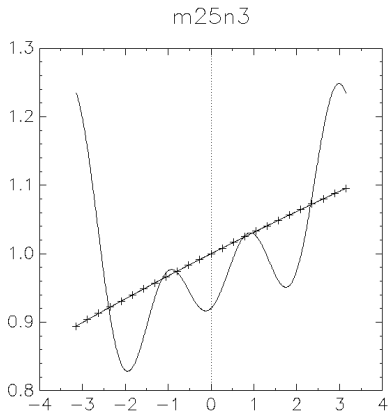
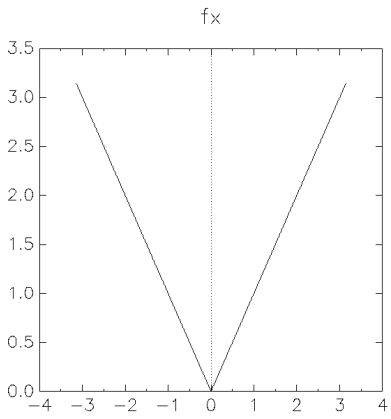
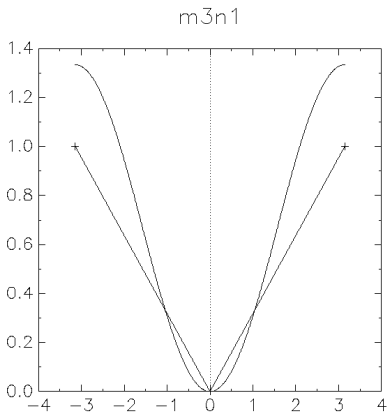
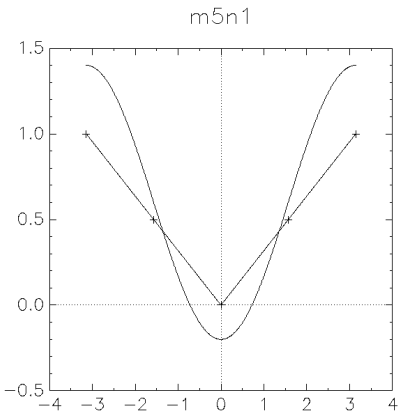


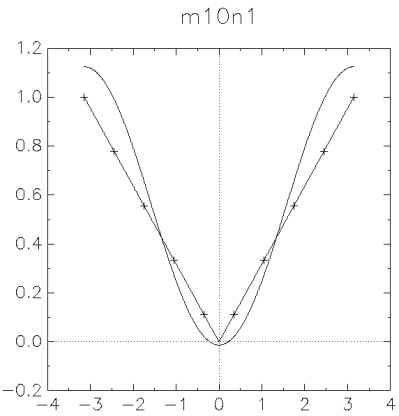
Figure 5: approximation where m=25, n=3

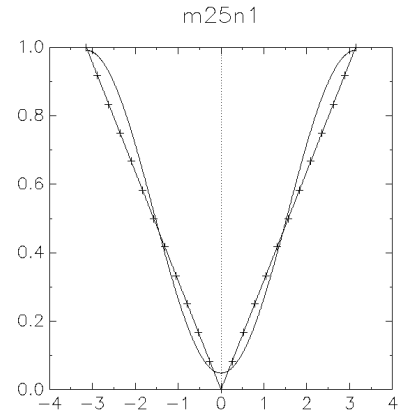
B.3 Extra: $f(x) = |x|$

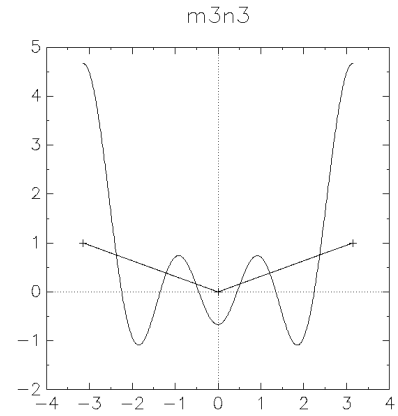


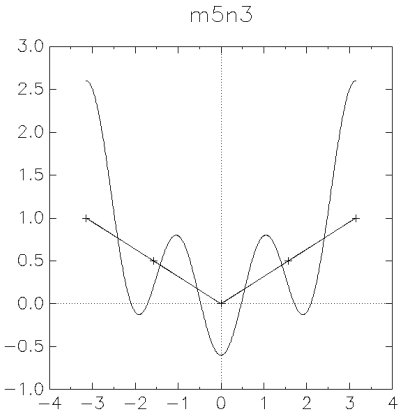


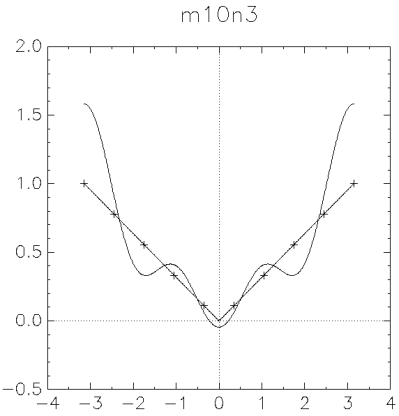

m5n1

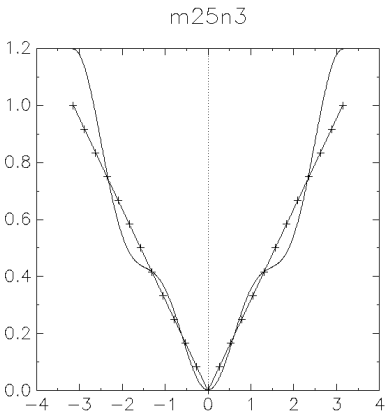

m10n1


m25n1


m3n3


m5n3


m10n3


m25n3
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0
1
2
3
4