Wetenschappelijk Programmeren: Orthagonale Baisifuncties - Oefening 1

Ruben Van Assche - S0122623

9 december 2016

1 Opgave

In deze opgave moest een functie $e^{-x} * sin(\pi * x)$ benaderd worden door een least squares approximatie van graad 4 waarbij de datapunten bepaald worden door de nulpunten van het Chebychev Polynoom van graad 11. Dit op het interval [-1, 1]. Vervolgens moest een kleinste kwadraten trigoniometrische benadering bepaald worden van dezelfde vergelijking op het interval [-π, π]. Waarbij de nulpunten gegeven worden door $-\pi + 2\pi i/11$.

2 Chebychev

Eerst moeten we de nulpunten van $T_{11}(x)$ berkenen, dit kan door:

$$cos(\frac{(2j-1) * \pi}{2 * i})$$

hierbij is $i = 11$ de graad van het chebychev polynoom en loopt j van 1 tot 11. Voor deze elf punten rekenen we vervolgens de y waarden uit d.m.v. de functie $e^{-x} * sin(\pi * x)$.
Hierna is het nog een kwestie van d.m.v.gsl˙multifit˙linear het stelsel op te lossen:

$$\begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & x_1^3 & x_1^4 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{10}^0 & x_{10}^1 & x_{10}^2 & x_{10}^3 & x_{10}^4 \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \vdots \\ \lambda_4 \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_{10} \end{bmatrix}$$

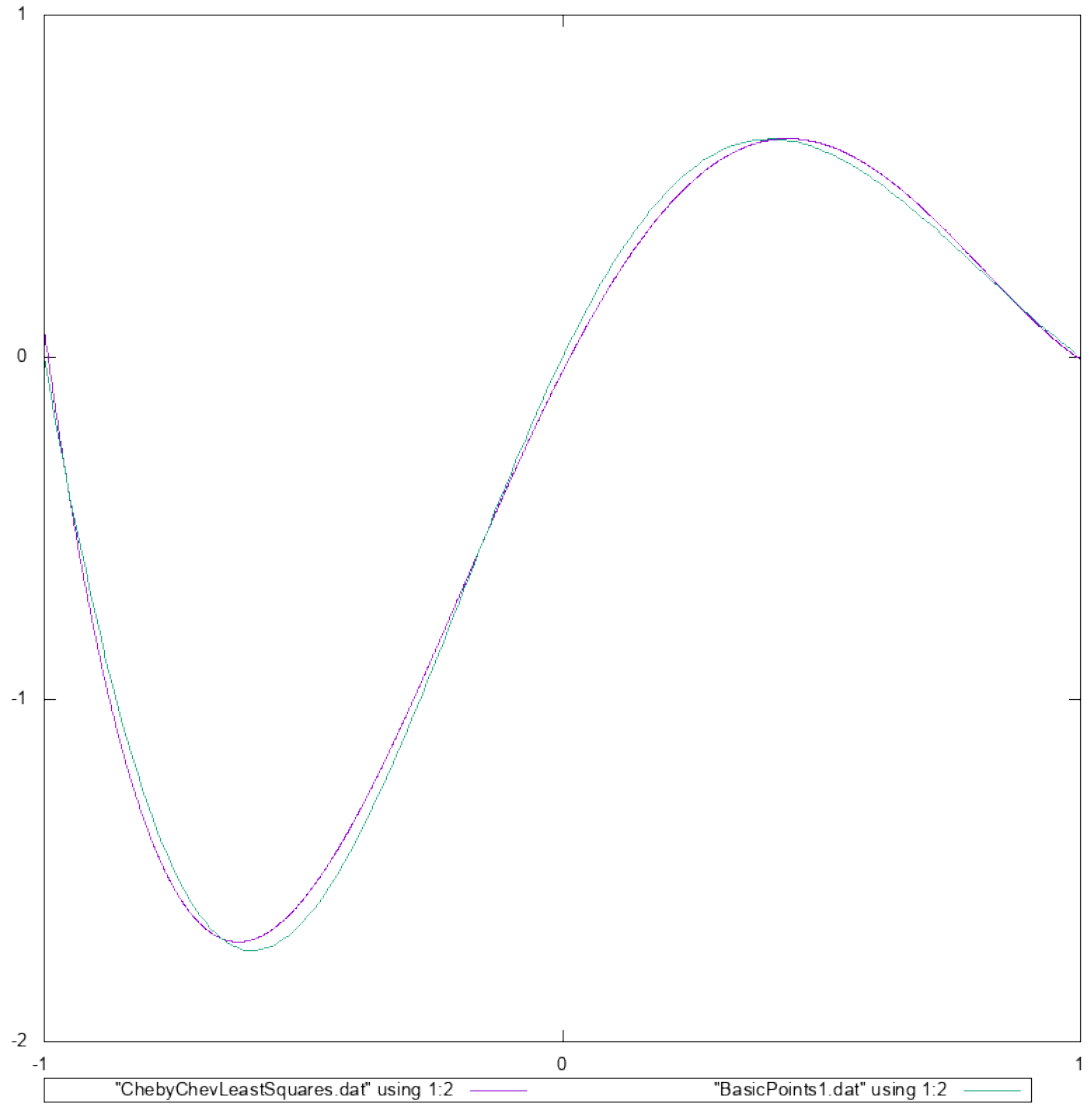
We verkrijgen hierdoor een approximatie voor $\lambda_0, ..., \lambda_4$. Deze worden ingevuld in g(x) en vervolgens verkijgen we:

$$g(x) = \lambda_0 + \lambda_1 * x + \lambda_2 * x^2 + \lambda_3 * x^3 + \lambda_4 * x^4$$

In het geval van $e^{-x} * sin(\pi * x)$ wordt g(x):

$$g(x) = -0.042053 + 2.9601 * x - 2.3619 * x^2 - 3.0056^3 + 2.4429 * x^4$$

Vervolgens plotten we f(x) (BasicPoints1.dat) en g(x) (ChebyChevLeastSquares.dat):



We zien dat dit een mooie approximatie is voor f(x).

2.1 Orthogonaliteit

Pas vrij laat besefte ik dat deze opgave ook op te lossen was d.m.v. de orthogonaliteit van Chebychev functies. Hierbij worden de $\lambda_i$niet meer berekend op basis van het oplossen van een stelsel maar d.m.v. een simpele vergelijking:

$$\lambda_j = \frac{\sum_{i=1}^{n} T_{j-1}(x_i) * y_i}{m/2}$$

Hierbij is n = 11 en m = 5. Helaas verkreeg ik nooit een goed resultaat en heb ik de berkening achterwege gelaten. Mijn 2 pogingen staan nog wel in de broncode met functienamen: generateChebychevLeastSquares2, generateChebychevLeastSquares3 in het bestand Trio.cpp.

3 Trigoniometrisch

Om een trigoniometrische least squares approximatie te verkrijegen is het niet verreist dat een complexe matrix moet opgelost worden. D.m.v. orthagonale basisfuncties weten we inmiddels dat $g(x) = \frac{a_0}{2} + a_1 * cos(x) + b_1 * cos(x) + b_2 * cos(2x) + a_2 * cos(2x)$. Hierbij is $g(x)$ de benaderde functie en geldt dit voor graad 2. Nu rest ons enkel nog a_0, a_1, a_2, b_1, b_2 te berekenen.

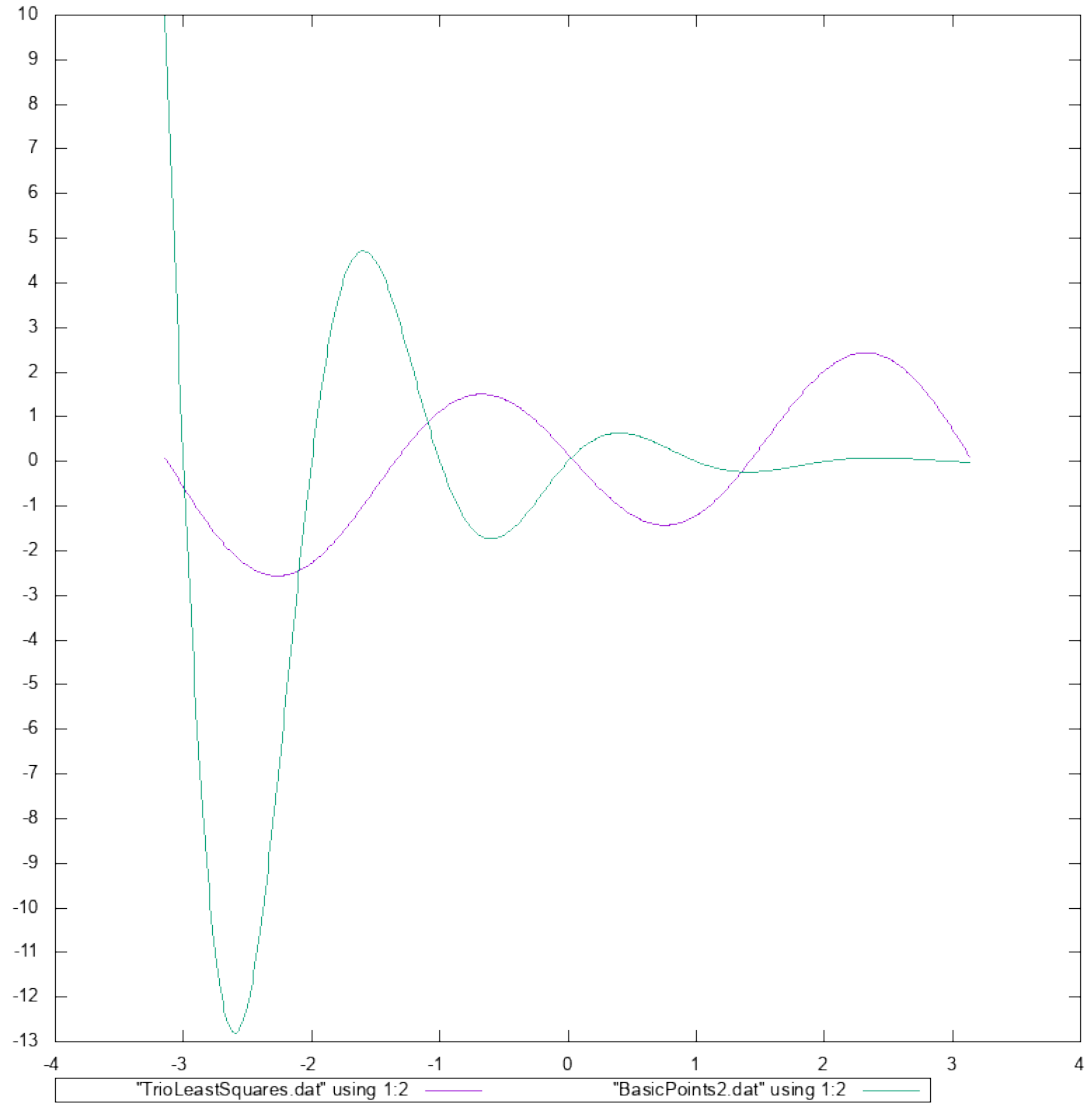
$$a_j = \frac{2}{11} \sum_{k=0}^{10} y_k * cos(j * x_k)$$

$$b_j = \frac{2}{11} \sum_{k=0}^{10} y_k * sin(j * x_k)$$

Deze a_i en b_i zijn zeer makkelijk te berekenen en voor de opgave komen we dan volgende waarden uit:

i	**a**	**b**
0	-0.029835274480727081575	0
1	0.047513231220430197921	0.73378196483313440357
2	0.1349790515049160144	-1.9614672973839133441

Deze worden dan ingevuld in $g(x) = \frac{a_0}{2} + a_1 * cos(x) + b_1 * cos(x) + b_2 * cos(2x) + a_2 * cos(2x)$ en onze approximatie is klaar!



Het eerste wat opvalt is dat dit geen goede approximatie is, de grafiek volgt de originele curve wel wat(dalen en stijgen) maar komt zeker niet in de buurt van een goede approximatie. Eerst dacht ik dat er een fout in mijn code zat maar andere functies zoals $sin(x), cos(x) en x^2$ hebben wel goede approximaties. Waarschijnlijk kan het verhogen van de graad helpen(weliswaar oppassen dat deze niet groot wordt $n > 2m + 1$) of datapunten kiezen op andere plaatsen.

4 Code

Util.cpp - Wat basiscode voor beide opgaven

```
#include "util.h"
```

```
template <typename T>
std::string to_pstring(const T a_value, const int n)
{
    std::ostringstream out;
    out << std::setprecision(n) << a_value;
    return out.str();
}

void print_matrix(gsl_matrix * m){
    int rows = m->size1;
    int columns = m->size2;

    for (int i = 0; i < rows; i++){  /* OUT OF RANGE ERROR */
        for (int j = 0; j <  columns; j++){
            std::cout << gsl_matrix_get (m, i, j) << "    ";
        }
        std::cout << std::endl;
    }
}

void print_vector(gsl_vector * v){
    int rows = v->size;

    for (int i = 0; i < rows; i++){  /* OUT OF RANGE ERROR */
        std::cout << gsl_vector_get (v, i) << std::endl;
    }
}


void writePointsFile(std::vector< std::pair<double, double> >& points, std::str
    std::ofstream file;
    file.open(filename);

    file << "#  X   Y" << std::endl;
    for(auto i: points){
        file << i.first << "   " << i.second << std::endl;
    }

    file.close();
}

double f(double x){
    double e = 2.718281828459045;
    double pi = 3.141592653589793;

    return pow(e, -x)*sin(pi*x);
}

void calculateBasicPoints(double a, double b, std::string filename){
    std::vector< std::pair<double, double> > points;
```

```
    for(double i = a;i <= b;i += 0.01){
        points.push_back(std::make_pair(i, f(i)));
    }

    writePointsFile(points, filename);
}
```

Chebychev.cpp - Code voor de Chebychev opgave

```
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <fstream>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_blas.h>
#include <sstream>
#include <iomanip>
#include "util.h"
#include <gsl/gsl_chebyshev.h>

double pi = 3.141592653589793;

double chebychevZero(int j){
    int i = 11;
    return cos(((2.0*j-1.0)*pi)/(2.0*i));
}

std::vector< std::pair<double, double> > calculateChebychevPoints(){
    std::vector< std::pair<double, double> > points;

    int i = 11;
    for(int j = 1;j <= i;j++){
        double x = chebychevZero(j);

        points.push_back(std::make_pair(x, f(x)));
    }

    return points;
}



gsl_vector* buildYVector(int m){
    gsl_vector* v = gsl_vector_alloc(m);
    for(int i = 1;i <= m;i++){
        gsl_vector_set(v, i - 1, f(chebychevZero(i)));
    }

    return v;
```

```
}

gsl_matrix* buildMatrixA(int n, int m){
    gsl_matrix* A = gsl_matrix_alloc(m, n);

    for(int i = 0;i < m; i++){
        for(int j = 0;j < n;j++){
            double x = chebychevZero(i+1);

            gsl_matrix_set(A, i, j, pow(x, j));
        }
    }


    return A;
}

gsl_vector* generateChebychevLeastSquares(){
    int n = 5;
    int m = 11;

    gsl_vector* c = gsl_vector_alloc(n);
    gsl_vector* y = buildYVector(m);
    gsl_vector* r = gsl_vector_alloc(m);
    gsl_matrix* A = buildMatrixA(n, m);
    gsl_matrix* cov = gsl_matrix_alloc(n, n);
    gsl_multifit_linear_workspace * work = gsl_multifit_linear_alloc (m, n);
    double chisq;

    gsl_multifit_linear(A, y, c, cov, &chisq, work);
    gsl_multifit_linear_residuals(A, y, c, r);


    std::cout << "Residu: " << gsl_blas_dnrm2(r) << std::endl;

    gsl_multifit_linear_free(work);
    gsl_matrix_free(cov);
    gsl_matrix_free(A);
    gsl_vector_free(y);
    gsl_vector_free(r);


    return c;
}

gsl_vector* generateChebychevLeastSquares2(){
    int n = 5;
    int m = 11;

    gsl_vector* c = gsl_vector_alloc(n);
```

```
    gsl_vector* y = buildYVector(m);

    for(int j = 0;j <= n-1;j++){
        double top = 0.0;
        double bottom = 0.0;

        for(int i = 1;i <= m;i++){
            double fj = pow(chebychevZero(i), j);
            top += fj*gsl_vector_get(y, i-1);
            bottom += pow(fj , 2);

        }
        std::cout << "\n";

        gsl_vector_set(c, j, top/bottom);
    }

    gsl_vector_free(y);

    return c;
}

// n = 0
inline double T0(double x)
{
    return 1.0 ;
}

// n = 1
inline double T1(double x)
{
    return x ;
}

// n = 2
inline double T2(double x)
{
    return (2.0 * x*x) - 1.0 ;
}


inline double Tn(unsigned int n, double x)
{
    if (n == 0)
    {
        return T0(x) ;
    }
    else if (n == 1)
    {
        return T1(x) ;
```

```
    }
    else if (n == 2)
    {
        return T2(x) ;
    }

    double tnm1(T2(x)) ;
    double tnm2(T1(x)) ;
    double tn(tnm1) ;

    for (unsigned int l = 3 ; l <= n ; l++)
    {
        tn = (2.0 * x * tnm1) - tnm2 ;
        tnm2 = tnm1;
        tnm1 = tn;
    }

    return tn ;
}

gsl_vector* generateChebychevLeastSquares3(){
    int m = 5;
    int n = 11;

    gsl_vector* c = gsl_vector_alloc(m);
    gsl_vector* y = buildYVector(n);

    for(int j = 1;j <= m;j++){
        double total = 0.0;
        for(int i = 1;i <= n;i++){
            gsl_cheb_series *cs = gsl_cheb_alloc(j-1);
            total += Tn(j-1,chebychevZero(i))*gsl_vector_get(y, i-1);

        }

        gsl_vector_set(c, j-1, total*(2.0/m));
    }

    gsl_vector_free(y);

    return c;
}

double calculateChebychevLeastSquares(gsl_vector* c, double x){
    double output = 0.0;

    for(int i = 0;i < c->size;i++){
        double value = gsl_vector_get(c, i);
        double xValue = pow(x, i);
```

```
        output += value*xValue;
    }

    return output;
}

void fitChebychevLeastSquares(gsl_vector* c){
    std::vector< std::pair<double, double> >* points = new std::vector< std::pa

    for(double x = -1.0;x <= 1.0;x += 0.001){
        double y = calculateChebychevLeastSquares(c, x);

        std::pair<double, double> point = std::make_pair(x,y);
        points->push_back(point);
    }

    writePointsFile(*points, "ChebyChevLeastSquares.dat");
}

int main (void){
    // Set COUT Precision
    std::cout.precision(5);

    calculateBasicPoints(-1.0, 1, "BasicPoints1.dat");

    calculateChebychevPoints();
    gsl_vector* c = generateChebychevLeastSquares();
    print_vector(c);
    fitChebychevLeastSquares(c);
}
```

Trio.cpp - Code voor de trigoniometrische opgave

```
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <fstream>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_blas.h>
#include <sstream>
#include <iomanip>
#include "util.h"
#include <gsl/gsl_chebyshev.h>
#include <gsl/gsl_vector_double.h>

double pi = 3.141592653589793;

double calculateA(gsl_vector* ti, gsl_vector* xi, int j){
    int n = ti->size;
```

```
    double total = 0.0;
    for(int k = 0;k < n;k++){
        double xk = gsl_vector_get(xi, k);
        double tk = gsl_vector_get(ti, k);
        total += xk*cos(j*tk);
    }

    total *= 2.0/n;
    return total;
}

double calculateB(gsl_vector* ti, gsl_vector* xi, int j){
    int n = ti->size;

    double total = 0.0;
    for(int k = 0;k < n;k++){
        double xk = gsl_vector_get(xi, k);
        double tk = gsl_vector_get(ti, k);
        total += xk*sin(j*tk);
    }

    total *= 2.0/n;
    return total;
}

double calculatePoint(std::vector<double>* a, std::vector<double>* b, double x)
    double y = 0.0;

    // A0
    y += a->at(0)/2.0;

    for(int i = 1;i < a->size();i++){
        y += a->at(i)*cos(i*x);
        y += b->at(i)*sin(i*x);
    }

    return y;
}

void fit(int m){
    gsl_vector* ti = gsl_vector_alloc(11);
    for(int i = 0;i <= 10;i++){
        gsl_vector_set(ti, i, -pi+((2*pi*i)/(11.0)));
    }

    gsl_vector* xi = gsl_vector_alloc(11);
    for(int i = 0;i <= 10;i++){
        double t = gsl_vector_get(ti, i);
        gsl_vector_set(xi, i, f(t));
    }
```

```
    std::vector<double>* a = new std::vector<double>;
    std::vector<double>* b = new std::vector<double>;

    for(int i = 0;i <= m;i++){
        double ai = calculateA(ti, xi, i);
        double bi = calculateB(ti, xi, i);
        a->push_back(ai);
        b->push_back(bi);

        std::cout << "a" << i << ": " << ai << ",    b" << i << ": " << bi << s
    }

    std::vector< std::pair<double, double> > points;
    for(double x = -pi;x < pi;x += 0.01){
        points.push_back(std::make_pair(x, calculatePoint(a, b, x)));
    }

    writePointsFile(points, "TrioLeastSquares.dat");

}

int main (void){
    // Set COUT Precision
    std::cout.precision(20);
    calculateBasicPoints(-pi, pi, "BasicPoints2.dat");
    fit(2);
}
```