Wetenschappelijk Programmeren: Data Smoothing - Oefening 5

Ruben Van Assche - S0122623

18 november 2016

1 Opgave

In deze opgave moest voor $n = 6, 7, 8, 11, 13, 15$ een veelterm worden bepaald d.m.v. de kleinste kwadraten methode die de 17 punten tussen [-1, 1] benadert van de Runge functie.

2 Opstellen van het model

Bij Data Smoothing gaan we proberen het residu te minimaliseren. Dit betekent dus dat we volgende vergelijkingen willen minimaliseren:

$$\sum_{i=1}^{m} \left[y_i - \sum_{j=1}^{n} a_{ij} \lambda_j \right]^2$$

Hierbij is y_i een vector met de y-Waarden van de Runghe functie. Deze kunnen makkelijk worden berekend uit de opgave: 0.03846, 0.04965, 0.06639, 0.09289, 0.1379, 0.2215, 0.3902, 0.7191, 1, 0.7191, 0.3902, 0.2215, 0.1379, 0.09289, 0.06639, 0.04965, 0.03846.

De bijhorende x-waarden zijn: -1, -0.875, -0.75, -0.625, -0.5, -0.375, -0.25, -0.125, 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.

De A matrix waaruit we a_{ij} halen is een $mxn + 1$ matrix waarbij op elke rij een x waarde tot een bepaalde macht wordt verhoffen. Deze macht wordt bepaald door de kolom waar deze x waarde in staat. Staat de waarde bijvoorbeeld in de 3de kolom dan wordt deze tot de tweede macht verhoffen. Staat deze in de 4de kolom dan wordt deze tot de 3de macht verhoffen enzovoorts.

De matrix A ziet er dan als volgt uit:

$$\begin{bmatrix} x_1^0 & x_1^1 & \cdots & x_1^n \\ x_2^0 & x_2^1 & \cdots & x_2^n \\ \vdots & \vdots & & \vdots \\ x_m^0 & x_m^1 & \cdots & x_m^n \end{bmatrix}$$

De λ vector is op deze moment onbekend maar heeft een grootte van n + 1.

3 Berekenen van de lambda vector

De factoren in de lambda vector worden berekend met een lineaire multiparameter regressie. GSL voorziet hiervoor een functie *gsl multifit linear*. Aan deze functie worden de A matrix, y vector en λ vector meegegeven. GSL berekent dan de waarden in de λ vector d.m.v. singular value decomposition van de matrix A gebruikmakend van het gewijzigde Golub-Reinsch SVD algoritme.

De factoren in de λ vector zullen uiteindelijk de coefficienten van de veelterm vergelijking vormen die de gegeven punten probeert te benderen.

Vervolgens berekenen we d.m.v. *gsl multifit linear residuals* de residu's deze worden bepaald door: $y - A * \lambda$ en zullen later nog gebruikt worden.

4 De verglijkingen

Van zodra de factoren in de λ vector bepaald zijn kunnen we vergelijkingen voor de veelterm opstellen:

n = 6

$$Y = 0.7689993515 + 9.414854674e-16X - 4.336109672X^2 + 1.281502612e-15X^3 + 7.777329573X^4 - 1.282551025e-15X^5 - 4.196381387X^6$$

n = 7

$$Y = 0.7689993515 + 2.688093862e-16X - 4.336109672X^2 - 4.841921829e-16X^3 + 7.777329573X^4 + 1.168264296e-15X^5 - 4.196381387X^6 - 1.184527236e-15X^7$$

n = 8

$$Y = 0.8501998604 - 7.258487361e-15X - 7.161857918X^2 + 1.683022793e-14X^3 + 22.70956181X^4 - 4.307021306e-15X^5 - 28.9036082X^6 - 1.385842305e-14X^7 + 12.54970251X^8$$

n = 11

$$Y = 0.9072324888 + 7.140610879e-14X - 10.37101681X^2 - 2.101452267e-12X^3 + 50.95571241X^4 + 1.247554915e-11X^5 - 114.269923X^6 - 3.017527929e-11X^7 + 116.0762224X^8 + 3.187811671e-11X^9 - 43.26056639X^{10} - 1.218867951e-11X^{11}$$

n = 13

$$Y = 0.9471737963 - 1.584623709e-13X - 13.82653137X^2 + 8.181339107e-12X^3 + 98.4047674X^4 - 1.02533859e-10X^5 - 347.9856535X^6 + 4.680857313e-10X^7 + 626.8761862X^8 - 9.684471789e-10X^9 - 548.6596796X^{10} + 9.222148256e-10X^{11} + 184.2822627X^{12} - 3.274590204e-10X^{13}$$

n = 15

$$Y = 0.9754849108 + 1.116712052e-12X - 17.55205188X^2 + 3.664490199e-12X^3 + 176.6106197X^4 - 5.865735805e-11X^5 - 949.2311491X^6 - 2.544832873e-10X^7 + 2775.990946X^8 + 2.826537973e-09X^9 - 4398.964036X^{10} - 7.514213553e-09X^{11} + 3525.205695X^{12} + 8.033802204e-09X^{13} - 1112.997048X^{14} - 3.038144153e-09X^{15}$$

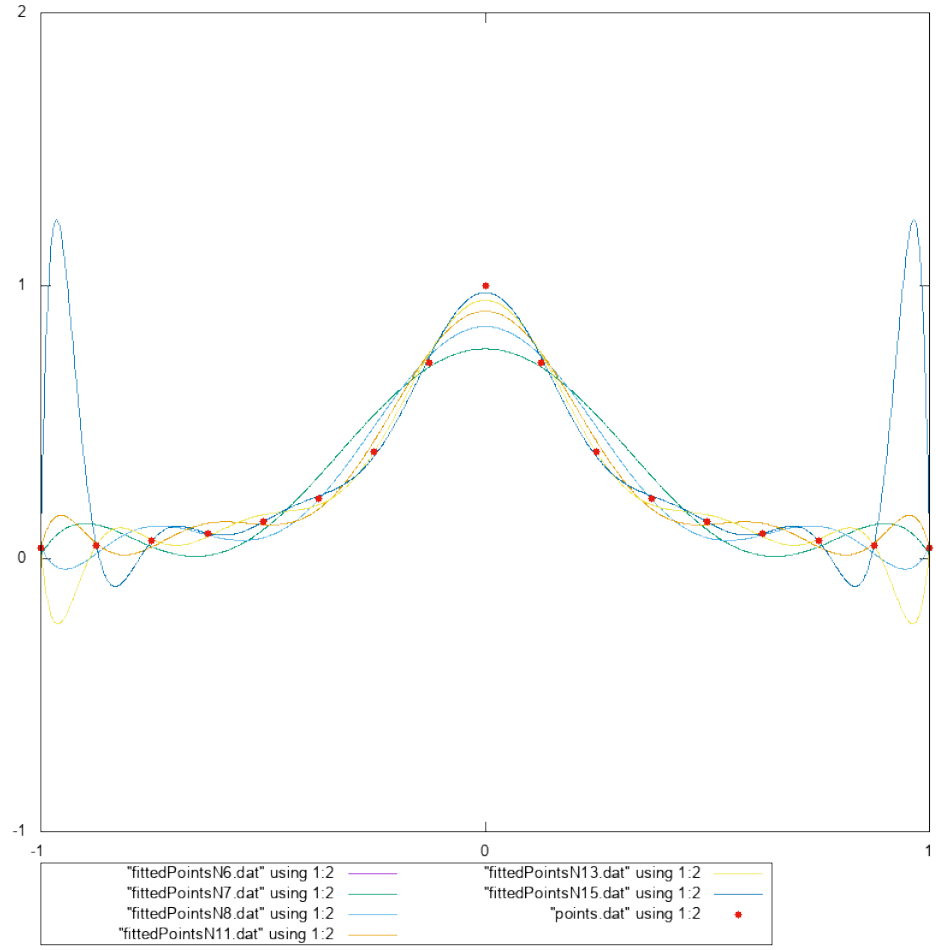
5 De residu's

De residu's zijn ook berekend, om deze beter te interpreteren heb ik de l_2 norm van de vector genomen:

N	**Residu**
6	0.36469983584691223477
7	0.3646998358469124013
8	0.23782656759792672463
11	0.14938135629848825481
13	0.08844594204494805878
15	0.046700465578771409303

6 Plots

Hieronder een plot van de originele gegeven punten + voor elke n een verglijking geplot.



7 Bevindingen

7.1 Minimale Residu's

Naarmate n stijgt, worden de residu's kleiner waardoor de veelterm steeds dichter bij de gegeven punten ligt(zie hiervoor ook de plots).

7.2 Beste plot

We zien dat de veelterm met een hogere n dichter bij de gegeven punten liggen maar natuurlijk zorgt dit ook weer voor het Runghe fenomeen(zie de uiteindes). Dus dan lijkt een veelterm met een lagere n wel beter. Het wijzigen van basis zou er voor kunnen zorgen dat we toch een veelterm met hogere n kunnen gebruiken met minder last van het Runghe fenomeen.

7.3 Ontbreken van n = 7

Dit fenomeen snap ik niet echt: de vergelijkingen gegenereerd uit de λ vector zijn dan wel verschillend voor $n = 6$ en $n = 7$ de punten die eruit worden gegenereerd zijn exact hetzelfde. Dit is raar want voor alle andere n komen er andere punten uit. Ik heb ook een als test $n = 5$ berekend(omdat misschien bij lagere n om een of andere reden de punten hetzelfde zijn) maar deze genereert ook verschillende punten t.o.v. $n = 6$ en $n = 7$.

8 Code

Main.cpp - De code voor alle berekeningen

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_spline.h>
#include <iostream>
#include <vector>
#include <utility>
#include <fstream>
#include <string>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_vector_double.h>
#include <gsl/gsl_blas.h>
#include <sstream>
#include <iomanip>

template <typename T>
std::string to_pstring(const T a_value, const int n = 6)
{
    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();
}

std::vector< std::pair<double, double> >* calculatePoints(){
    std::vector< std::pair<double, double> >* points = new std::vector< std::pa


    for(double x = -1.0;x <= 1.0;x += 0.125){
        double y = 1.0/(1.0+25*pow(x, 2));


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

    return points;
}
```

```
gsl_matrix* buildMatrixA(int n, int m){
    gsl_matrix* A = gsl_matrix_alloc(m, n);
    double step = 2.0/(m-1);

    for(int i = 0;i < m; i++){
        for(int j = 0;j < n;j++){
            double x = -1 + i*step;
            double y = 1.0/(1.0+25*pow(x, 2));

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

    return A;
}

gsl_vector* buildYVector(int m){
    gsl_vector* Y = gsl_vector_alloc(m);
    double step = 2.0/(m-1);

    for(int j = 0;j < m;j++){
        double x = -1 + j*step;
        double y = 1.0/(1.0+25*pow(x, 2));

        gsl_vector_set(Y, j, y);
    }

    return Y;

}

// Generate the coefficients with m points and of grade n
gsl_vector* fit(int n, int m){
    n = n + 1; // Grade 6 needs 7 terms

    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;
}

std::string buildEquation(gsl_vector* c){
    std::string output = "Y = ";

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

        // Add the sign
        if(i != 0){
            if(value >= 0.0){
                output += " + ";
            }
        }

        // Add value
        output += to_pstring(value, 10);
        output += " ";

        // Add x^i
        if(i > 1){
            output += "X^{";
            output += std::to_string(i);
            output += "} ";
        }else if(i == 1){
            output += "X ";
        }else{}
    }

    return output;
}

double fittedFunction(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;
}
```

```
std::vector< std::pair<double, double> >* calculateFittedPoints(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 = fittedFunction(c, x);

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

    return points;
}



int main (void){
    // Set COUT Precision
    std::cout.precision(20);

    // Write original points to file
    auto originalPoints = calculatePoints();
    writePointsFile(*originalPoints, "points.dat");

    std::vector<int> indices = {6,7,8,10,11,12,13,14,15};
    for(auto i : indices){
        std::cout << "--------------------------" << std::endl;
        std::cout << "n = " << i << std::endl;
        std::cout << "--------------------------" << std::endl;
        gsl_vector* c = fit(i, 17);

        std::string equation = buildEquation(c);
        std::cout << equation << std::endl;

        std::string filename = "fittedpointsN";
        filename += std::to_string(i);
        filename += ".dat";

        auto points = calculateFittedPoints(c);
        writePointsFile(*points, filename);

        gsl_vector_free(c);
    }



}
```