Data Smoothing Oefening 3

Ruben Van Assche / s0122623

Doel
Bereken de optimale modellen en vergelijk de residuen en conditiegetallen.

Computer berekening
De GSL Library bevat een hoop functies die dit probleem zal oplossen. Ten eerste zal er een matrix moeten worden opgesteld met de lineaire functies.

$$\textbf{Model1} \begin{bmatrix} T_0(x_0) & T_1(x_0) & T_2(x_0) & T_3(x_0) \\ \vdots & \vdots & \vdots & \vdots \\ T_0(x_{10}) & T_1(x_{19}) & T_2(x_{19}) & T_3(x_{19}) \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \vdots \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_{19} \end{bmatrix}$$

Met
$T_0(x_i) = 0$
$T_1(x_i) = x_i$
$T_2(x_i) = 2x_i^2 - 1$
$T_3(x_i) = 4x_i^3 - 2x_i^2 - 2x_i$

$$\textbf{Model2} \begin{bmatrix} x_0^0 & x_0^1 & x_0^2 & x_0^3 \\ \vdots & \vdots & \vdots & \vdots \\ x_{19}^0 & x_{19}^1 & x_{19}^2 & x_{19}^3 \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \vdots \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_{19} \end{bmatrix}$$

Deze twee modellen volgen de regel $A\lambda = y$
Uiteindelijk willen we r vinden waarbij $r = min(y - A\lambda)$

D.m.v. gsl_linalg_QR_lssolve(…) kunnen we deze minimale residu vector berekenen. Er wordt hierbij gebruik gemaakt van de de euclidische norm.

Hiervoor moet de matrix eerst wel gedecomposeerd worden naar de QR vorm, waarbij de functie gsl_linalg_QR_decomp(…) helpt.

Hierna bekomen we onze waarden voor de λ vector en onze residu vector.

Voor het berekenen van het conditiegetal(dit doen we voor de QR decompositie) maak ik gebruik van de gsl_linalg_SV_decomp(…) functie welke een vector genereert. Hieruit kunnen we dan de minimale en maximale waarden halen en zo berekenen we het conditiegetal: maximale waarde/ minimale waarde.

Omdat onze conditiegetallen vrij groot gaan zijn door onze huidige input van gegevens, en we een beter geconditioneerde matrix willen. Zullen we de x waarden van de originele input waarden herschalen naar het interval [1, -1]. Dit doen we door:

$$x^{\star} = (x/4) - 1$$

Vaststellingen

I	Residu Model 1	Residu Model 1 Scaled	Residu Model 2	Residu Model 2 Scaled
1	0.318262027055292107124984113398	0.318262027055292384680740269687	0.318262027055293161836857507296	0.318262027055292773258798888492
2	-0.203876786263175346913811836203	-0.203876786263176068558777842554	-0.2038767862631751526247825268	-0.203876786263176790203743848906
3	-0.0350792550765729807538129136901	-0.0350792550765675753554617699592	-0.0350792550765750901775597014876	-0.0350792550765678598501118301556
4	-0.107621134652853056579147050797	-0.107621134652853764346325249335	-0.1076211346528532508681763602	-0.107621134652853278623751975829
5	-0.510429765694362713723819524603	-0.510429765694363157813029374665	-0.510429765694362380656912137056	-0.51042976569436304679072691215
6	-0.655997085174607197366469790722	-0.655997085174607419411074715754	-0.655997085174607086344167328207	-0.655997085174607419411074715754
7	-0.466815030067096070620635828163	-0.466815030067097236354811684578	-0.466815030067096570220996909484	-0.466815030067097291865962915836
8	0.337912017816654419721800195475	0.337912017816653087454170645287	0.337912017816654697277556351764	0.337912017816653309498775570319
9	0.86806528513444158878797907164	0.868065285134440256520349521452	0.868065285134442143899491384218	0.868065285134440478564954446483
10	0.884246147632694823315091525728	0.88424614763269426820357921315	0.884246147632695267404301375791	0.884246147632694490248184138181
11	0.484034936475151389423388081923	0.484034936475151778001446700728	0.484034936475151666979144238212	0.484034936475151778001446700728
12	-0.433244209290022830316502222558	-0.433244209290023829517224385199	-0.433244209290022663783048528785	-0.433244209290023662983770691426
13	-0.642637057399754851871875871439	-0.642637057399754185738061096345	-0.642637057399753963693456171313	-0.64263705739975429676036355886
14	-0.138752708644870770271495530324	-0.138752708644869771070773367683	-0.138752708644871519672037152304	-0.138752708644870048626529523972
15	-0.121808988608717250201607384952	-0.121808988608716431412126723899	-0.12180898860871730571275861621	-0.121808988608716611823368225487
16	-0.224246635224743917635947809686	-0.224246635224743862124796578428	-0.224246635224744111924977119088	-0.224246635224743751102494115912
17	-0.0131440546174226915998595188739	-0.0131440546174230177278730025137	-0.0131440546174216091324105093463	-0.013144054617423149566857176751
18	0.503744584551227281288277026761	0.503744584551227836399789339339	0.503744584551226726176764714182	0.503744584551227947422091801855
19	0.88856721582677444803977095944	0.888567215826773226794443871768	0.888567215826773337816746334283	0.888567215826773115772141409252
20	-0.731179503778037198635786353407	-0.731179503778036643524274040828	-0.731179503778036199435064190766	-0.731179503778036532501971578313

I	Verschil model 1-2	Verschil model 1-2 scaled
1	0,000000000000001054711873	0,000000000000000388578058618805
2	0,000000000000000194289029	0,000000000000000721644966006352
3	0,000000000000002109423747	0,000000000000000284494650060196

I	Verschil model 1-2	Verschil model 1-2 scaled
4	0,000000000000000194289029	0,0000000000000000485722573273506
5	0,000000000000000333066907	0,000000000000000111022302462516
6	0,000000000000000111022302	0
7	0,000000000000000499600361	0,0000000000000000555111512312578
8	0,000000000000000277555756	0,000000000000000222044604925031
9	0,000000000000000555111512	0,000000000000000222044604925031
10	0,000000000000000444089210	0,000000000000000222044604925031
11	0,000000000000000277555756	0
12	0,000000000000000166533454	0,000000000000000166533453693773
13	0,000000000000000888178420	0,000000000000000111022302462516
14	0,000000000000000749400542	0,000000000000000277555756156289
15	0,000000000000000055511151	0,000000000000000180411241501588
16	0,000000000000000194289029	0,000000000000000111022302462516
17	0,000000000000001082467449	0,000000000000000131838984174237
18	0,000000000000000555111512	0,000000000000000111022302462516
19	0,000000000000001110223025	0,000000000000000111022302462516
20	0,000000000000000999200722	0,000000000000000111022302462516

We merken dus wanneer we van model veranderen dat dit bijna niets veranderd aan onze residu vectoren. We veranderen eigenlijk van basis en dit zal dan wel een verschillend conditiegetal opleveren maar de residu vector zal niet veel verschillen omdat wisselen van basis feitelijk enkel wijzigingen aanbrengt aan het conditiegetal.

Conditiegetal Model 1	Conditiegetal Model 2	Conditiegetal Model 1 Scaled	Conditiegetal Model 2 Scaled
3887.2909864748357904318254441	1012.158027891432880096544977 28	3.779881441181648593641284605 84	6.661680378667408142234762635 78

Wanneer we onze data niet schalen naar het interval [-1, 1], dan zal het conditiegetal van het eerste model veel hoger zijn dan dat van het tweede model. Dit wil zeggen dat model 2 beter geconditioneerd is en dus bij deze veel beter te vertrouwen is op de correctheid van de oplossing.

Nu blijven deze beide getallen in de 100tallen steken en dit wil zeggen dat onze oplossing alles behalve goed geconditioneerd is. Wanneer we herschalen zien we een merkwaardig fenomeen: waar model 1 voor de herschaling het slechtst geconditioneerd was zien we nu dat dit model 2 is.

Willen we dus de meest correcte oplossing dan kunnen we het best gebruik maken van een herschaalde data set waarop we dan model 1 toepassen.

Code

```
#include <stdio.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_math.h>
#include <iostream>
#include "util.h"
#include <math.h>
#include <vector>
#include <utility>
#include <exception>
#include <limits>

class Point{
public:
  double x;
  double y;

  Point(double fx, double fy){
    this->x = fx;
    this->y = fy;
  };
};

typedef  std::vector<Point> ValueList;

double generateTcoefficientbase2(int i, double x){
  if(i == 0){
    return 1.0;
  }else if(i == 1){
    return x;
  }else if(i == 2){
    return pow(x, 2);
  }else if(i == 3){
    return pow(x, 3);
  }else{
    throw std::runtime_error("Invalid i");
  }
}

double generateTcoefficientbase1(int i, double x){
  if(i == 0){
    return 1.0;
  }else if(i == 1){
    return x;
  }else if(i == 2){
    return 2*pow(x, 2) - 1;
  }else if(i == 3){
    return 4*pow(x, 3) - 2*pow(x, 2) - 2*x;
  }else{
    throw std::runtime_error("Invalid i");
  }
}

gsl_matrix* generatebase1matrix(ValueList* values){
  gsl_matrix* matrix = gsl_matrix_alloc (20, 4);

  for(int i = 0; i < 20;i++){
    for(int j = 0; j < 4;j++){
```

```
      gsl_matrix_set(matrix, i, j, generateTcoefficientbase1(j, values->at(i).x));
    }
  }

  return matrix;
}

gsl_matrix* generatebase2matrix(ValueList* values){
  gsl_matrix* matrix = gsl_matrix_alloc (20, 4);

  for(int i = 0; i < 20;i++){
    for(int j = 0; j < 4;j++){
      gsl_matrix_set(matrix, i, j, generateTcoefficientbase2(j, values->at(i).x));
    }
  }

  return matrix;
}

gsl_vector* generateyvector(ValueList* values){
  gsl_vector* vector = gsl_vector_alloc(20);

  for(int i = 0;i < 20; i++){
    gsl_vector_set(vector, i, values->at(i).y);
  }

  return vector;
}

double calculateConditionNumber(gsl_matrix* matrix){
  gsl_vector* vector = gsl_vector_alloc(4);
  gsl_matrix* tempmatrix = gsl_matrix_alloc(4,4);
  gsl_vector* work = gsl_vector_alloc(4);

  gsl_linalg_SV_decomp(matrix, tempmatrix, vector, work);

  double min = std::numeric_limits<double>::max();
  double max = -std::numeric_limits<double>::min();
  for(int i = 0; i < 4;i++){
    double result = gsl_vector_get(vector, i);
    if(result > max){
      max = result;
    }

    if(result < min){
      min = result;
    }
  }

  gsl_vector_free(vector);
  gsl_matrix_free(tempmatrix);
  gsl_vector_free(work);

  return max/min;
}

ValueList*  initializeValues(){
  ValueList* values = new ValueList();
  values->push_back(Point(0.0, -0.8 ));
  values->push_back(Point(0.6, -0.34));
```

```
  values->push_back(Point(1.5, 0.59));
  values->push_back(Point(1.7, 0.59));
  values->push_back(Point(1.9, 0.23));
  values->push_back(Point(2.1, 0.1));
  values->push_back(Point(2.3, 0.28));
  values->push_back(Point(2.6, 1.03));
  values->push_back(Point(2.8, 1.5));
  values->push_back(Point(3.0, 1.44));
  values->push_back(Point(3.6, 0.74));
  values->push_back(Point(4.7, -0.82));
  values->push_back(Point(5.2, -1.27));
  values->push_back(Point(5.7, -0.92));
  values->push_back(Point(5.8, -0.92));
  values->push_back(Point(6.0, -1.04));
  values->push_back(Point(6.4, -0.79));
  values->push_back(Point(6.9, -0.06));
  values->push_back(Point(7.6, 1.00));
  values->push_back(Point(8.0, 0.00));

  return values;
}

void scaleValues(ValueList* values){
  for(int i = 0;i < 20; i++){
    values->at(i).x = (values->at(i).x/4.0) - 1;
  }
}

void base1(){
  std::cout << "--------------" << std::endl;
  std::cout << "Base 1" << std::endl;
  std::cout << "--------------" << std::endl;

  ValueList* values = initializeValues();

  gsl_matrix* matrixcondition = generatebase1matrix(values);
  std::cout << "Condition number: " << calculateConditionNumber(matrixcondition) << std::endl;

  gsl_matrix* matrix = generatebase1matrix(values);
  gsl_vector* tau = gsl_vector_alloc(4);
  gsl_vector* y = generateyvector(values);
  gsl_vector* t = gsl_vector_alloc(4);
  gsl_vector* residual = gsl_vector_alloc(20);
  gsl_linalg_QR_decomp(matrix, tau);
  print_vector(tau);
  gsl_linalg_QR_lssolve (matrix, tau, y, t, residual);

  std::cout << "t vector:" << std::endl;
  print_vector(t);

  std::cout << "Minimal Residual vector:" << std::endl;
  print_vector(residual);

  // Cleanup
  delete values;
  gsl_matrix_free(matrixcondition);
  gsl_matrix_free(matrix);
  gsl_vector_free(tau);
  gsl_vector_free(y);
  gsl_vector_free(t);
```

```
  gsl_vector_free(residual);
}

void base2(){
  std::cout << "--------------" << std::endl;
  std::cout << "Base 2" << std::endl;
  std::cout << "--------------" << std::endl;

  ValueList* values = initializeValues();

  gsl_matrix* matrixcondition = generatebase2matrix(values);
  std::cout << "Condition number: " << calculateConditionNumber(matrixcondition) << std::endl;

  gsl_matrix* matrix = generatebase2matrix(values);
  gsl_vector* tau = gsl_vector_alloc(4);
  gsl_vector* y = generateyvector(values);
  gsl_vector* t = gsl_vector_alloc(4);
  gsl_vector* residual = gsl_vector_alloc(20);
  gsl_linalg_QR_decomp(matrix, tau);
  print_vector(tau);
  gsl_linalg_QR_lssolve (matrix, tau, y, t, residual);

  std::cout << "t vector:" << std::endl;
  print_vector(t);

  std::cout << "Minimal Residual vector:" << std::endl;
  print_vector(residual);

  // Cleanup
  delete values;
  gsl_matrix_free(matrixcondition);
  gsl_matrix_free(matrix);
  gsl_vector_free(tau);
  gsl_vector_free(y);
  gsl_vector_free(t);
  gsl_vector_free(residual);
}

void base1scaled(){
  std::cout << "--------------" << std::endl;
  std::cout << "Base 1 Scaled" << std::endl;
  std::cout << "--------------" << std::endl;

  ValueList* values = initializeValues();
  scaleValues(values);

  gsl_matrix* matrixcondition = generatebase1matrix(values);
  std::cout << "Condition number: " << calculateConditionNumber(matrixcondition) << std::endl;

  gsl_matrix* matrix = generatebase1matrix(values);
  gsl_vector* tau = gsl_vector_alloc(4);
  gsl_vector* y = generateyvector(values);
  gsl_vector* t = gsl_vector_alloc(4);
  gsl_vector* residual = gsl_vector_alloc(20);
  gsl_linalg_QR_decomp(matrix, tau);
  gsl_linalg_QR_lssolve (matrix, tau, y, t, residual);

  std::cout << "t vector:" << std::endl;
  print_vector(t);
```

```
  std::cout << "Minimal Residual vector:" << std::endl;
  print_vector(residual);

  // Cleanup
  delete values;
  gsl_matrix_free(matrixcondition);
  gsl_matrix_free(matrix);
  gsl_vector_free(tau);
  gsl_vector_free(y);
  gsl_vector_free(t);
  gsl_vector_free(residual);
}

void base2scaled(){
  std::cout << "--------------" << std::endl;
  std::cout << "Base 2 Scaled" << std::endl;
  std::cout << "--------------" << std::endl;

  ValueList* values = initializeValues();
  scaleValues(values);

  gsl_matrix* matrixcondition = generatebase2matrix(values);
  std::cout << "Condition number: " << calculateConditionNumber(matrixcondition) << std::endl;

  gsl_matrix* matrix = generatebase2matrix(values);
  gsl_vector* tau = gsl_vector_alloc(4);
  gsl_vector* y = generateyvector(values);
  gsl_vector* t = gsl_vector_alloc(4);
  gsl_vector* residual = gsl_vector_alloc(20);
  gsl_linalg_QR_decomp(matrix, tau);
  gsl_linalg_QR_lssolve (matrix, tau, y, t, residual);

  std::cout << "t vector:" << std::endl;
  print_vector(t);

  std::cout << "Minimal Residual vector:" << std::endl;
  print_vector(residual);

  // Cleanup
  delete values;
  gsl_matrix_free(matrixcondition);
  gsl_matrix_free(matrix);
  gsl_vector_free(tau);
  gsl_vector_free(y);
  gsl_vector_free(t);
  gsl_vector_free(residual);
}

int main(){
  // set preicison
  std::cout.precision(30);
  base1();
  base2();
  base1scaled();
  base2scaled();
}
```