Wetenschappelijk Programmeren: Lineaire Stelsels Vegelijkingen - Oefening 4

Ruben Van Assche

3 november 2016

1 Opgave

In deze opgave moest voor $n = 3, 6, 9, 12$ de Hilbert Matrix berekend worden, van deze matrix moest dan een determinant en conditiegetal berekend worden. Vervolgens moest deze via GEPP opgelost worden met een kolom uit de eenheidsmatrix. Daarna moest de fout berekend worden.

2 Hilbert Matrix Berekenen

De Hilbert matrix is makkelijk te berekenen met de gegeven formule. Volgende matrices zijn berekend met Maple(het programma bij dit rapport geschreven berekent deze ook maar Maple kan deze matrices makkelijker omzetten naar een Latex formaat).

n = 3

$$\begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$$

n = 6

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{bmatrix}$$

n = 9

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 \\ 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 \\ 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 \\ 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 & 1/17 \end{bmatrix}$$

n = 12

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 & 1/17 \\ 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 & 1/17 & 1/18 \\ 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 & 1/17 & 1/18 & 1/19 \\ 1/9 & 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 & 1/17 & 1/18 & 1/19 & 1/20 \\ 1/10 & 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 & 1/17 & 1/18 & 1/19 & 1/20 & 1/21 \\ 1/11 & 1/12 & 1/13 & 1/14 & 1/15 & 1/16 & 1/17 & 1/18 & 1/19 & 1/20 & 1/21 & 1/22 \\ 1/12 & 1/13 & 1/14 & 1/15 & 1/16 & 1/17 & 1/18 & 1/19 & 1/20 & 1/21 & 1/22 & 1/23 \end{bmatrix}$$

3 Determinant

Om de determinant te berekenen wordt er gebruik gemaakt van een LU decompositie. We zien dat naarmate de matrix groter wordt de determinant exponentieel kleiner wordt. Dit is logisch gezien bij de berkening van de determinant het product van de diagonale elementen van de U matrix worden genomen. Het aantal kleine diagonale kleine waarden stijgt naarmate n stijgt waardoor het totale product met een stijging van n kleiner wordt.

n	**Determinant**
3	0.00046296296296296135433
6	5.3672998868168426532e-18
9	9.7202660411305632762e-43
12	2.8332547472051984479e-78

4 Konditiegetal

Om het konditiegetal te berekenen wordt gebruik gemaakt van een singuliere waarden decompositie, hierbij worden uit de singuliere waarden van de matrix A de maximale en de minimale waarde gehaald. Vervolgens is het konditiegetal max/min. Hierdoor krijgen we volgende konditiegetallen:

n	**Konditiegetal**
3	524.05677758605759209
6	14951058.64061903581
9	493153283608.33990479
12	17797394126083356

Het valt direct op dat het konditiegetal exponentieel stijgt, dit is logisch: naarmate de matrix groter wordt is deze veel slechter geconditioneerd. Dit komt omdat bij elke stijging van n, er kleinere waarden in de matrix bijkomen waardoor het verschil in grootte tussen de waarde in de linker bovenhoek en de rechter onderhoek zeer groot is. Dit veroorzaakt een slecht gekonditioneerde matrix.

5 Oplossen met GEPP

Voor het oplossen van de matrix A met GEPP moest voor het rechterlid een kolom uit de eenheidsmatrix genomen worden. Voor elke n heb ik hierbij de eerste kolom uit de eenheidsmatrix genomen waardoor deze altijd ÃÂ©ÃÂ©n enkele 1 bovenaan heeft en voor de rest nullen. Omdat deze berekende oplossing moest vergeleken worden met een exacte oplossing om zo de fout in te schatten heb ik eerst d.m.v. MAPLE de matrix opgelost. Hieronder staat een stukje voorbeeld code van hoe een matrix op te lossen in MAPLE met $n = 3$.

```
m := HilbertMatrix(3, 3)
b := Vector[column]([1, 0, 0])
x := LinearSolve(m, b)
```

De vectoren die hieruit komen zijn hard-coded in de code om later bij de foutenafschatting te gebruiken.

Om d.m.v. GEPP de oplossing te berekenen doen we eerst een LU decompositie(zie determinant), hierdoor bekomen we dan een matrix welke we zeer makkelijk kunnen oplossen d/m/v/ $Ax = y$ omdat deze A driehoekig is. Hieronder staan de berekende x vectoren:

n	**3**	**6**	**9**	**12**
1	9	36	81	141.7
2	-36	-630	-3240	-1.001e+04
3	30	3360	4.158e+04	2.312e+05
4		-7560	-2.495e+05	-2.579e+06
5		7560	8.108e+05	1.639e+07
6		-2772	-1.514e+06	-6.46e+07
7			1.622e+06	1.652e+08
8			-9.266e+05	-2.79e+08
9			2.188e+05	3.088e+08
10				-2.154e+08
11				8.587e+07
12				-1.492e+07

6 Factor C

De factor c moet worden berekend uit volgende ongelijkheid: $\|x - \tilde{x}\| \leqslant c * K(A) * \|\tilde{x}\| * ULP$. Met een klein beetje rekenwerk kunnen we deze ongelijkheid omvormen tot volgende: $\frac{\|x-\tilde{x}\|}{K(A)*\|\tilde{x}\|*ULP} \leqslant c$. Voor ULP wordt gebruik gemaakt van de standaard library van c++ welke volgende functie voorziet: *std::numericlimits¡double¿::epsilon()*. De andere factoren in de ongelijkheid zijn al in vorige delen van de oefening berekend. Hieruit moeten enkel nog de normen berekend worden, hierbij maken we gebruik van de euclidische l_2 norm, GSL voorziet hiervoor een functie om deze te berekenen. Uit de ongelijkheid halen we dan voor de verschillende n volgende c's:

n	**C**
3	0.035236789371484285305
6	7034286.1681871246547
9	0.028988609754939251989
12	0.019105287208575325725

Wat moet worden vermeld is dat in de deze ongelijkheid de factor c groter mag uitkomen dan in de tabel hierboven gegegeven. Dit omdat de c's hierboven gelijk zijn aan $\frac{\|x-\tilde{x}\|}{K(A)*\|\tilde{x}\|*ULP}$. Dit is belangrijk voor het volgende deel van de oefening.

7 Het Residu

Als laatste moet er gecheckt worden of volgende ongelijkheid: $\|y - AX\| \leqslant c * \|A\| * \|\tilde{x}\| * ULP$ klopt. Hiervoor moet de norm van de matrix A berekend worden waarvoor GSL geen enkele functie voorziet maar dit is geen probleem gezien dat de euclidische norm van de matrix gelijk is aan de maximale singuliere waarde(deze zijn al eerder berekend). Om een matrix te vermeningvuldigen met een vector werd er gebruk, gemaakt van de BLAS subroutines. Wanneer we dan de ongelijkheden voor verschillende n uitrekenen bekomen we volgende waarden:

n	Ongelijkheid
3	8.882e-16 ¡= 5.258e-16
6	1.705e-13 ¡= 2.924e-05
9	4.584e-11 ¡= 2.843e-11
12	8.248e-09 ¡= 3.876e-09

Het probleem is dat de ongelijkheden niet kloppen, nu is dit eigenlijk geen probleem. Als we zien naar de originele ongelijkheid dan staat er in het rechterlid de factor c. In de vorige sectie werd al gezegd dat de gegeven factor c hetzelfde is of groter. Laat ons nu een grotere factor c nemen, dan is het zeer makkelijk om de ongelijkheden te doen laten kloppen. Gezien de niet al te grote verschillen zou het vermenigvuldigen van c met 3 al alle problemen oplossen. Hierdoor kloppen de ongelijkheden.

8 Code

Main.cpp - De code voor alle berekeningen

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_math.h>
#include <iostream>
#include <vector>
#include <utility>
#include <fstream>
#include <limits>
#include <string>
#include <gsl/gsl_matrix_double.h>
#include <gsl/gsl_vector_double.h>

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;
    }
}

gsl_matrix* vectorToMatrix(gsl_vector* v){
    gsl_matrix* m = gsl_matrix_alloc(v->size, 1);

    for(int i = 0;i < v->size;i++){
        gsl_matrix_set(m, i, 0, gsl_vector_get(v, i));
    }

    return m;
}

gsl_vector* matrixToVector(gsl_matrix* m){
    gsl_vector* v = gsl_vector_alloc(m->size1);


    for(int i = 0;i < m->size1;i++){
        gsl_vector_set(v, i, gsl_matrix_get(m, i, 0));
    }

    return v;
}

class HilbertMatrix{
public:
    int n = 0;
    int s = 0;
    double c = 0.0;
    double determinant = 0.0;
    double conditionNumber = 0.0;
    double mNorm = 0.0;
    gsl_matrix* m = nullptr;
    gsl_matrix* mLU = nullptr;
    gsl_matrix* mLUS = nullptr;
    gsl_permutation* p = nullptr;
    gsl_vector* x = nullptr;
    gsl_vector* y = nullptr;


    HilbertMatrix(int n){
        this->n = n;

        // Initiate Hilbert Matrix
        this->m = gsl_matrix_alloc (n, n);
        for(int i = 1;i <= n;i++){
            for(int j = 1;j <= n;j++){
                double hilbert = 1.0/(i+j-1);
                gsl_matrix_set(this->m, i - 1, j - 1, hilbert);
```

```
        }
    }

    // Do an LU Decomposition
    this->mLU = gsl_matrix_alloc (n, n);

    gsl_matrix_memcpy(this->mLU, this->m);

    // Do an LU decompostion
    this->p = gsl_permutation_alloc(n);
    gsl_linalg_LU_decomp(this->mLU, this->p, &(this->s));
}

double getDeterminant(){
    this->determinant = gsl_linalg_LU_det(this->mLU, this->s);
    return this->determinant;
}

double getConditionNumber(){
    gsl_vector* vector = gsl_vector_alloc(this->n);
    gsl_matrix* tempmatrix = gsl_matrix_alloc(this->n,this->n);
    gsl_matrix* tempmatrix2 = gsl_matrix_alloc(this->n,this->n);
    gsl_vector* work = gsl_vector_alloc(this->n);

    gsl_matrix_memcpy(tempmatrix, this->m);

    gsl_linalg_SV_decomp(tempmatrix, tempmatrix2, vector, work);

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

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

    this->mNorm = max;
    this->conditionNumber = max/min;
    return this->conditionNumber;
}

double getC(){
    double ULP = std::numeric_limits<double>::epsilon();

    // x* - x
```

```
    gsl_vector* xx = gsl_vector_alloc(this->n);
    gsl_vector_memcpy(xx, this->solveViaMaple());
    gsl_vector_sub(xx, this->solveViaGEPP());
    double xxNorm = gsl_blas_dnrm2(xx);

    // K(A)*x**ULP
    double xNorm = gsl_blas_dnrm2(this->solveViaGEPP());

    this->c = xxNorm/(this->conditionNumber*xNorm*ULP);


    std::cout << xxNorm << " <= " << this->c*this->conditionNumber*xNorm*UL

    return this->c;
}

double getResidu(){
    double ULP = std::numeric_limits<double>::epsilon();

    // Ax*
    gsl_matrix* axm = gsl_matrix_alloc(this->n, 1);
    gsl_blas_dgemm (CblasNoTrans, CblasNoTrans, 1.0, this->m, vectorToMatri
    gsl_vector* ax  = matrixToVector(axm);

    // y - AX*
    gsl_vector* yy = gsl_vector_alloc(this->n);
    gsl_vector_memcpy(yy, this->y);
    gsl_vector_sub(yy, ax);


    double yax = gsl_blas_dnrm2(yy);

    // c*A*x**ULP
    double caxu = this->c*this->mNorm*gsl_blas_dnrm2(this->x)*ULP;

    std::cout << yax << " <= " << caxu << std::endl;

    return yax;
}


gsl_vector* solveViaGEPP(){
    // Initiate vectors
    this->y = gsl_vector_alloc(this->n);
    this->x = gsl_vector_alloc(this->n);

    // Inititiate MLUS matrix
    this->mLUS = gsl_matrix_alloc (this->n, this->n);

    for(int i = 0;i < this->n;i++){
```

```
        if(i == 0){
            gsl_vector_set(this->y, i, 1.0);
        }else{
            gsl_vector_set(this->y, i, 0.0);
        }
    }

    // Copy the original LU matrix to an temporary matrix
    gsl_matrix_memcpy(this->mLUS, this->mLU);


    // Solve the temporary LU matrix
    gsl_linalg_LU_solve (this->mLUS, this->p, this->y, x);

    return x;
}

gsl_vector* solveViaMaple(){
    /*
     * This vectors are pre calculated with Maple
     * using the following commands:
     * m := HilbertMatrix(3, 3)
     * b := Vector[column]([1, 0, 0])
     * x := LinearSolve(m, b)
     */

    // Initiate vectors
    this->x = gsl_vector_alloc(this->n);
    std::vector<double> values;

    if(this->n == 3){
        values.push_back(9);
        values.push_back(-36);
        values.push_back(30);
    }else if(this->n == 6){
        values.push_back(36);
        values.push_back(-360);
        values.push_back(3360);
        values.push_back(-7560);
        values.push_back(7560);
        values.push_back(-2772);
    }else if(this->n == 9){
        values.push_back(81);
        values.push_back(-3240);
        values.push_back(41580);
        values.push_back(-249480);
        values.push_back(810810);
        values.push_back(-1513512);
        values.push_back(1621620);
        values.push_back(-926640);
```

```
            values.push_back(218790);
        }else if(this->n == 12){
            values.push_back(144);
            values.push_back(-10296);
            values.push_back(240240);
            values.push_back(-2702700);
            values.push_back(17297280);
            values.push_back(-68612544);
            values.push_back(176432256);
            values.push_back(-299304720);
            values.push_back(332560800);
            values.push_back(-232792560);
            values.push_back(93117024);
            values.push_back(-16224936);
        }else{
            std::cout << "Couldn't find pre calculated vector, using linear sol
            return this->solveViaGEPP();
        }

        for(int i = 0;i < this->n;i++){
            gsl_vector_set(x, i, values.at(i));
        }

        return x;
    }

    void printMatrix(){
        int rows = this->m->size1;
        int columns = this->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 (this->m, i, j) << "
";
            }
            std::cout << std::endl;
        }
    }

    void calc(){
        std::cout << "-------------------------------------------------" << std::endl;
        std::cout << "Hilbert(" << this->n << ")" << std::endl;
        std::cout << "-------------------------------------------------" << std::endl;
        this->printMatrix();
        this->solveViaGEPP();
        std::cout << "-------------------------------------------------" << std::endl;
        std::cout << "Determinant: " << this->getDeterminant() << std::endl;
        std::cout << "Condition Number: " << this->getConditionNumber() << std:
        std::cout << "---------------- Y Vector ----------------" << std::endl;
        print_vector(this->y);
```

```
        std::cout << "---------------- X Vector ----------------" << std::endl;
        print_vector(this->x);
        std::cout << "------------------------------------------" << std::endl;
        this->getC();
        std::cout << "c: " << this->c << std::endl;
        std::cout << "Residu:" << std::endl;
        this->getResidu();
        std::cout << std::endl << std::endl << std::endl;
    }
};


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


    for(int i = 3; i < 15; i += 3){
        auto h = HilbertMatrix(i);
        h.calc();
    }
}
```