Lösung Übungsblatt 7

Aufgabe 1: Lineare Ausgleichsrechnung

Ein mehrdimensionales, lineares Ausgleichungsproblem lässt sich folgendermaßen darstellen:

$$f(\boldsymbol{x}) = \sum_{k=1}^{N} a_k X_k(\boldsymbol{x}) \tag{1.1}$$

Dabei ist $f(\boldsymbol{x})$ eine skalarwertige Vektorfunktion und die Koeffizienten a_k ergeben sich aus den N Messwerten $\{x_i, f_i\}$

a) Es gilt zu zeigen, dass ein Fit von N Wertepaaren an eine Kreisfunktion

$$(x - x_c)^2 + (y - y_c)^2 = R^2 \tag{1.2}$$

ein lineares Ausgleichsproblem mit den Basisfunktionen $X_1 = 1$, $X_2 = x$ und $X_3 = y$ darstellt.

Dies ergibt sich so:

$$\begin{aligned}
(x - x_c)^2 + (y - y_c)^2 &= R^2 \\
x^2 - 2xx_c + x_c^2 + y^2 - 2yy_x + y_c^2 &= R^2 \\
\underbrace{x^2 + y^2}_{f(\boldsymbol{x})} &= \underbrace{(R^2 - x_c^2 - y_c^2)}_{a_1} + \underbrace{2x_c}_{a_2} \cdot x + \underbrace{2y_c}_{a_3} \cdot y \\
f(\boldsymbol{x}) &= a_1 + a_2 x + a_3 y \\
&= \sum_{k=1}^{3} a_k X_k(\boldsymbol{x})
\end{aligned}$$

mit $X_1 = 1, X_2 = x$ und $X_3 = y$. $\square$

b) Für die Elemente der Designmatrix $\boldsymbol{C}$ gilt:

$$\boldsymbol{C}_{ik} = \frac{X_k(\boldsymbol{x}_i)}{\varrho_i} \tag{1.3}$$

Mit den X_k aus Teilaufgabe a) und $\varrho_{xi} = \varrho_{yi} = 1$ ergibt sich:

$$\begin{aligned} \boldsymbol{C} &= \begin{pmatrix} X_1(\boldsymbol{x}_1) & X_2(\boldsymbol{x}_1) & X_3(\boldsymbol{x}_1) \\ X_1(\boldsymbol{x}_2) & X_2(\boldsymbol{x}_2) & X_3(\boldsymbol{x}_2) \\ \vdots & \vdots & \vdots \\ X_1(\boldsymbol{x}_n) & X_2(\boldsymbol{x}_n) & X_3(\boldsymbol{x}_n) \end{pmatrix} \\ &= \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & y_n \end{pmatrix} \end{aligned}$$

c) Das Normalgleichungssystem $\boldsymbol{A}\boldsymbol{a} + \boldsymbol{b} = 0$ kann als $\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{a} + \boldsymbol{C}^T\boldsymbol{d} = 0$ geschrieben werden, wobei für $\boldsymbol{d}_i = -\frac{f_i}{\varrho_i}$ gilt.

Für $\boldsymbol{A}$ ergibt sich also:[1]

$$\begin{aligned} \boldsymbol{A} = \boldsymbol{C}^T\boldsymbol{C} &= \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_3 \\ y_1 & y_2 & \dots & y_n \end{pmatrix} \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & y_n \end{pmatrix} \\ &= \begin{pmatrix} 1+1+\dots+1 & x_1+x_2+\dots+x_n & y_1+y_2+\dots+y_n \\ x_1+x_2+\dots+x_n & x_1^2+x_2^2+\dots+x_n^2 & x_1y_1+x_2y_2+\dots+x_ny_n \\ y_1+y_2+\dots+y_n & x_1y_1+x_2y_2+\dots+x_ny_n & y_1^2+y_2^2+\dots+y_n^2 \end{pmatrix} \\ &= \begin{pmatrix} N & \sum x_i & \sum y_i \\ \sum x_i & \sum x_i^2 & \sum x_iy_i \\ \sum y_i & \sum x_iy_i & \sum y_i^2 \end{pmatrix} \end{aligned}$$

[1] Die Summen verlaufen von $i = 1 \dots N$

$$
\begin{aligned}
\boldsymbol{b} = -\boldsymbol{\beta} &= \boldsymbol{C}^T \cdot \boldsymbol{d} \\
&= \begin{pmatrix} 1 & 1 & ... & 1 \\ x_1 & x_2 & ... & x_n \\ y_1 & y_2 & ... & y_n \end{pmatrix} \begin{pmatrix} -x_1^2 - y_1^2 \\ -x_2^2 - y_2^2 \\ \vdots \\ -x_n^2 - y_n^2 \end{pmatrix} \\
&= -\begin{pmatrix} x_1^2 + y_1^2 + x_2^2 + y_2^2 + ... + x_n^2 + y_n^2 \\ x_1^3 + x_1 y_1^2 + x_2^2 + x_2 y_2^2 + ... + x_n^3 + x_n y_n^2 \\ y_1 x_1^2 + y_1^3 + y_2 x_2^2 + y_2^3 + ... + y_n x_n^2 + y_n^3 \end{pmatrix} \\
&= -\begin{pmatrix} \sum (x_i^2 + y_i^2) \\ \sum x_i (x_i^2 + y_i^2) \\ \sum y_i (x_i^2 + y_i^2) \end{pmatrix}
\end{aligned}
$$

Beispiel:

Die folgenden Wertepaare sollen durch einen Kreis angenähert werden:

x	0.4	3.1	5.9	6.9	6.1	4.3	0.4	-0.9
y	3.7	4.6	3.7	1.5	-1.7	-3.0	-2.3	1.6

Mit obigen Gleichungen lautet das Normalgleichungssystem bei dreistelliger Genauigkeit:

$$
\begin{pmatrix} 8.000 & 26.200 & 8.100 \\ 26.200 & 148.860 & 22.290 \\ 8.100 & 22.290 & 70.530 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 219.390 \\ 1093.075 \\ 289.244 \end{pmatrix}
$$

Der Lösungsvektor ergibt sich zu:[2]

$$
\boldsymbol{a} = \begin{pmatrix} 6.136 \\ 6.040 \\ 1.487 \end{pmatrix}
$$

[2] Ich habe das LGS mit dem relaxierten Jacobi-Verfahren mittels der Software aus Aufgabenblatt 6 gelöst.

Mit den Beziehungen aus Aufgabenteil a) folgt:

$$\begin{aligned} x_c &= \frac{a_2}{2} = \frac{6.040}{2} \\ &= 3.020 \\ y_c &= \frac{a_3}{2} = \frac{1.487}{2} \\ &\approx 0.744 \\ R &= \sqrt{a_1 + \frac{a_1^2}{4} + \frac{a_3^2}{4}} \\ &= \sqrt{6.136 + \frac{6.040^2}{4} + \frac{1.487^2}{4}} \\ &\approx 3.976 \end{aligned}$$

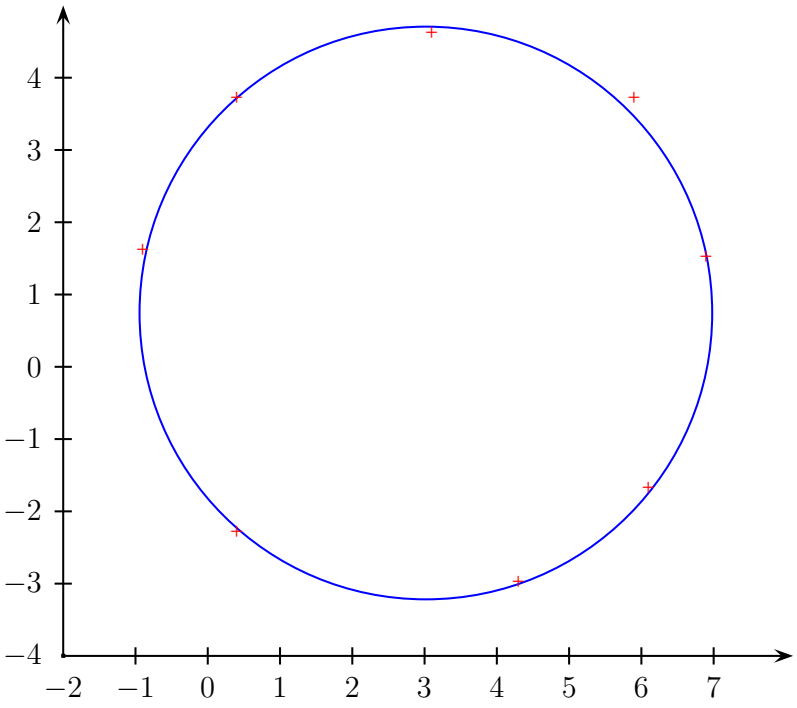

Abbildung 1.1: Werte und Ausgleichung zum obigen Beispiel.

Aufgabe 2: Householder-Spiegelung

a) und b)

Das angegebene Vorgehen zur Householder-Transformation kann durch ein paar Überlegungen für die Umsetzung in einem Computer-Programm vereinfacht werden.

Das Vorgehen wird anhand der ersten Householder-Matrix beschrieben:

$$\boldsymbol{H}_1 = \mathbb{1} - \frac{2\tilde{\boldsymbol{\omega}}_1 \tilde{\boldsymbol{\omega}}_1^T}{\|\tilde{\boldsymbol{\omega}}_1\|_2^2} \tag{2.1}$$

$$\tilde{\boldsymbol{\omega}}_1 = \boldsymbol{x}_1 \pm \|\boldsymbol{x}_1\|_2 \boldsymbol{e}_1 \tag{2.2}$$

Als erstes wird $\|\boldsymbol{x}_1\|_2$ berechnet. Die Berechnung der später benötigten euklidischen Norm von $\tilde{\boldsymbol{\omega}}_1$ zum Quadrat $\|\tilde{\boldsymbol{\omega}}_1\|_2^2$ kann nun direkt erfolgen, denn:

$$\begin{aligned}
\|\tilde{\boldsymbol{\omega}}_1\|_2^2 &= (a_{11} \pm \|\boldsymbol{x}_1\|_2)^2 + a_{21}^2 + ... + a_{n1}^2 \\
&= \|\boldsymbol{x}_1\|_2^2 \pm 2a_{11}\|\boldsymbol{x}_1\|_2 + \underbrace{a_{11}^2 + a_{21}^2 + ... + a_{n1}^2}_{\|\boldsymbol{x}_1\|_2^2} \\
&= 2\|\boldsymbol{x}_1\|_2^2 \pm 2|a_{11}|\|\boldsymbol{x}_1\|_2 \\
&= 2\|\boldsymbol{x}_1\|_2 \cdot (\|\boldsymbol{x}_1\|_2 + \operatorname{sign}(a_{11})a_{11}) \qquad (2.3)
\end{aligned}$$

a_{11} stammt aus der gegebenen Matrix und $\|\boldsymbol{x}_1\|_2$ wurde bereits berechnet, (2.3) spart also eine komplette Schleife!

Das herkömliche Vorgehen diktiert, zunächst die Matrix $\boldsymbol{H}_1$ zu ermitteln, um dann das Matrixprodukt $\boldsymbol{H}_1 A$ zu bilden.
Auch dieser Schritt lässt sich vereinfachen. Dazu spalten wir die Matrix $\boldsymbol{A}$ in Spaltenvektoren auf:

$$\boldsymbol{A} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$$

Das Matrixprodukt wird dann zu

$$\boldsymbol{H}_1 \boldsymbol{A} = (\boldsymbol{H}_1\boldsymbol{x}_1, \boldsymbol{H}_1\boldsymbol{x}_2, ..., \boldsymbol{H}_1\boldsymbol{x}_n)$$

Die Idee besteht also darin, die einzelnen Vektoren $\boldsymbol{H}_1\boldsymbol{x}_i$ zu berechnen und dann in die Matrix $\boldsymbol{A}$ zurückzuschreiben.

Für $\boldsymbol{H}_1 \boldsymbol{x}_i$ gilt:

$$\begin{aligned} \boldsymbol{H}_1 \boldsymbol{x}_i &= \left(\mathbb{1} - \frac{2\tilde{\boldsymbol{\omega}}_1 \tilde{\boldsymbol{\omega}}_1^T}{\|\tilde{\boldsymbol{\omega}}_1\|_2^2} \right) \cdot \boldsymbol{x}_i \\ &= \boldsymbol{x}_i - \frac{2\tilde{\boldsymbol{\omega}}_1}{\|\tilde{\boldsymbol{\omega}}_1\|_2^2} \underbrace{\tilde{\boldsymbol{\omega}}_1^T \boldsymbol{x}_i}_{<\tilde{\boldsymbol{\omega}}_1, \boldsymbol{x}_i>} \\ &= \boldsymbol{x}_i - \frac{2 \langle \tilde{\boldsymbol{\omega}}_1, \boldsymbol{x}_i \rangle}{\|\tilde{\boldsymbol{\omega}}_1\|_2^2} \cdot \tilde{\boldsymbol{\omega}}_1 \end{aligned} \tag{2.4}$$

So kann $\boldsymbol{H}_1 \boldsymbol{A}$ direkt berechnet werden, ohne überhaupt $\boldsymbol{H}_1$ bestimmt zu haben! Analoges gilt für $\boldsymbol{H}_{N-1} \cdot \ldots \cdot \boldsymbol{H}_1 \boldsymbol{A}$.
Gleichung (2.3) und (2.4) wurden in unten stehenden Algorithmus integriert.

C++ Quellcode:

```
/*-------------------------------------------------------------------------
              M4 Numerik fuer Physiker Sommersemester 2008
                        Uebungsblatt 7 - A.2
  -------------------------------------------------------------------------
  Titel:    QR-Zerlegung
  Datei:    householder.cpp
  Erstellt: 02.06.2008
  Autor:    Stefan Flaischlen
  -------------------------------------------------------------------------*/

#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

enum DisplayType {
    displayTitle,
    displayLine
};

// Funktionen deklarieren:
double** createMatrix   (int);
void     displayMatrix  (int, double**, char*);
void     deleteMatrix   (int, double**);
void     deleteVector   (double*);
void     displaySpecials(DisplayType = displayLine, char* = 0);
void     Householder    (int, double**);

//-------------------------- Hauptfunktion ------------------------------
int main() {

    // Zeilen-/Spaltenzahl:
    int dimN = 4;

    // Programmtitel ausgeben:
    displaySpecials(displayTitle, "QR-Zerlegung");

    // Matrix A initalisieren:
    double** matrixA = createMatrix(dimN);

    for (int i = 0; i < dimN; i++) {
        for (int j = 0; j < dimN; j++) {
            matrixA[i][j] = 1.0 / (1 + i + j);
        }
    }

    // Matrix ausgeben:
    displayMatrix(dimN, matrixA, "A");

    // Householder-Transformation:
    Householder(dimN, matrixA);

    cout << "\nAusgabe\n";
    displaySpecials();
    displayMatrix(dimN, matrixA, "R");

```

```
58      // Speicher freigeben:
59      deleteMatrix(dimN, matrixA);
60  }
61
62  //-------------- Funktion: Householder-Transformation ------------------
63  void Householder(int N, double** A) {
64
65      /* ------------------------------------------------------------
66         Eingabe:  Matrix A und dessen Dimension.
67         Funktion: Transformiert eine quadratische Matrix A mittels
68                   Householder-Spiegelung.
69         ------------------------------------------------------------ */
70
71      double* tmpVector = new double[N];
72      double tmpRowABS, tmpVectorABS, tmpValue;
73      int i;
74
75      for (int aktRow = 0; aktRow < N - 1; aktRow++) {
76
77          // Betrag des Spaltenvektors berechnen:
78          tmpRowABS = A[aktRow][aktRow] * A[aktRow][aktRow];
79          for (i = aktRow + 1; i < N; i++) {
80              tmpRowABS += A[i][aktRow] * A[i][aktRow];
81
82              // Untere Komponenten des Spaltenvektors uebertragen:
83              tmpVector[i] = A[i][aktRow];
84          }
85          tmpRowABS = sqrt(tmpRowABS);
86
87          // Erste Komponente des temporaeren Vektors berechnen:
88          if (A[aktRow][aktRow] < 0) tmpRowABS *= -1;
89          tmpVector[aktRow] = A[aktRow][aktRow] + tmpRowABS;
90
91          // Betragsquadrat des temporaeren Vektors berechnen:
92          tmpVectorABS = 2 * tmpRowABS * (tmpRowABS + A[aktRow][aktRow]);
93
94          // Spaltenvektoren berechnen: (Ergibt Produkt Q...*A)
95          for (int k = aktRow; k < N; k++) {
96              tmpValue = 0;
97              for (i = aktRow; i < N; i++) {
98                  tmpValue += A[i][k] * tmpVector[i];
99              }
100             tmpValue *= 2 / tmpVectorABS;
101             for (i = aktRow; i < N; i++) {
102                 A[i][k] -= tmpValue * tmpVector[i];
103             }
104         }
105     }
106
107     // Speicher freigeben:
108     deleteVector(tmpVector);
109 }
```

```
110
111 //----------------- Hilfsfunktion: Matrix ausgeben ---------------------
112 void displayMatrix(int dimN, double** sMatrix, char* sTitle) {
113
114     cout << "Matrix " << sTitle << ": " << "\n" << fixed;
115
116     for (int i = 0; i < dimN; i++) {
117         cout << setw(11) << "["
118              << setprecision(3) << setw(6) << sMatrix[i][0];
119         for (int j = 1; j < dimN; j++) {
120             cout << "  " << setprecision(3) << setw(6) << sMatrix[i][j];
121         }
122         cout << right << "]" << "\n";
123     }
124     cout << "\n";
125 }
126
127 //------------ Hilfsfunktion: Matrix dynamisch erzeugen ----------------
128 double** createMatrix(int dimN) {
129
130     double** dMatrix = new double*[dimN];
131
132     for (int i = 0; i < dimN; i++) {
133         dMatrix[i] = new double[dimN];
134         dMatrix[i][0] = 0;
135     }
136
137     return dMatrix;
138 }
139
140 //----------------- Hilfsfunktion: Matrix loeschen ---------------------
141 void deleteMatrix(int dimN, double** sMatrix) {
142
143     for (int i = 0; i < dimN; i++) {
144         deleteVector(sMatrix[i]);
145         delete sMatrix;
146     }
147 }
148
149 //----------------- Hilfsfunktion: Vektor loeschen ---------------------
150 void deleteVector(double* sVector) {
151    delete[] sVector;
152 }
153
154 //------------------ Hilfsfunktion: Titel, Linien ---------------------
155 void displaySpecials(DisplayType sType, char* sTitle) {
156
157     int textWidth = 60;
158     int stringLen = 0;
159
160     switch (sType) {
161         case displayTitle:
162             // Ueberschrift zentrieren:
163             while(sTitle[stringLen] != 0) stringLen++;
164             cout << setfill('-') << setw(textWidth) << "\n" << setfill(' ')
165                  << setw((textWidth + stringLen) / 2) << sTitle << "\n"
166                  << setfill('-') << setw(textWidth + 1) << "\n\n"
167                  << "Householder-Transformation der Matrix A zur Matrix R."
168                  << "\n\n\n"
169                  << "Eingabe" << "\n"
```

```
170                     << setfill('-') << setw(textWidth) << "\n" << setfill(' ');
171                 break;
172             case displayLine:
173                 cout << setfill('-') << setw(textWidth) << "\n" << setfill(' ');
174                 break;
175         }
176     }
```

Ausgabe:

```
------------------------------------------------------------
                        QR-Zerlegung
------------------------------------------------------------

Householder-Transformation der Matrix A zur Matrix R.


Eingabe
------------------------------------------------------------
Matrix A:
          [ 1.0000   0.5000   0.3333   0.2500]
          [ 0.5000   0.3333   0.2500   0.2000]
          [ 0.3333   0.2500   0.2000   0.1667]
          [ 0.2500   0.2000   0.1667   0.1429]


Ausgabe
------------------------------------------------------------
Matrix R:
          [-1.1932  -0.6705  -0.4749  -0.3698]
          [ 0.0000  -0.1185  -0.1257  -0.1175]
          [ 0.0000   0.0000  -0.0062  -0.0096]
          [ 0.0000   0.0000   0.0000   0.0002]
```