



**Ernst-Abbe-Fachhochschule Jena**  
Hochschule für angewandte Wissenschaften

Beleg

## **Numerische Mathematik - Problem2: schlecht konditioniertes LGS**

Marc Ludwig  
Marvin Hohlfeld  
Matthias Bukovsky  
Matthias Springstein

Jena, 24. Juni 2013

Betreuer: Prof. P. Wilde



# Inhaltsverzeichnis

<b>1. Entwickle die Funktion <math>f</math> aus obigem Problem in eine Taylorreihe</b>	<b>1</b>
1.1. Die Taylorreihe . . . . .	1
1.2. Definition . . . . .	1
1.3. Eigenschaften . . . . .	2
1.4. Konstruktion . . . . .	2
1.5. Entwicklung von $f(x)$ in eine Taylor Reihe . . . . .	2
<b>2. Nachweis der hermiteschen Form von <math>H</math></b>	<b>4</b>
2.1. Definition - hermitesch . . . . .	4
2.1.1. direkte Folgen aus der Definition . . . . .	4
2.2. Mathematischer Beweis für die gegebene Matrize 'H' . . . . .	5
<b>3. Eigenwerte und Kondition von H</b>	<b>6</b>
<b>4. Numerische Lösung des LGS für eine fünfstellige Genauigkeit</b>	<b>7</b>
<b>5. Visualisierung der Koeffezienten sowie des Taylorpolynoms</b>	<b>8</b>
5.1. Ausblick . . . . .	9
<b>Abbildungsverzeichnis</b>	<b>10</b>
<b>Quelltextverzeichnis</b>	<b>11</b>
<b>A. Anhang</b>	<b>12</b>
A.1. Quelltexte . . . . .	12



## **Abstract**

Begleitend zur Lehrveranstaltung „Numerische Mathematik“ im Sommersemester 2013, soll Anhand eines schlecht konditionierten Gleichungssystems aufgezeigt werden wie sich minimale Störungen der „rechten Seite“ auf die Lösung des Systems auswirken. Dieses Projekt wurde zudem mit einem Versionskontrollsystem bearbeitet und steht ausdrücklich zur Weiterbearbeitung zur Verfügung, unter <https://github.com/befedo/numa>.

# 1. Entwickle die Funktion $f$ aus obigem Problem in eine Taylorreihe

## 1.1. Die Taylorreihe

In der Analysis verwendet man Taylorreihen (auch Taylor-Entwicklungen, nach dem Mathematiker Brook Taylor<sup>1</sup>), um Funktionen in der Umgebung bestimmter Punkte durch Potenzreihen darzustellen (Reihenentwicklung). So kann ein komplizierter analytischer Ausdruck durch eine nach wenigen Gliedern abgebrochene Taylorreihe (oftmals gut) angenähert werden, z.B. in der Physik oder bei der Ausgleichung geodätischer Netze: So ist die oft verwendete Kleinwinkelnäherung des Sinus eine nach dem ersten Glied abgebrochene Taylorreihe dieser Funktion.

## 1.2. Definition

Sei  $I \subset \mathbb{R}$  ein offenes Intervall,  $f: I \rightarrow \mathbb{R}$  eine unendlich oft differenzierbare Funktion und  $a$  ein Element von  $I$ . Dann heißt die unendliche Reihe

$$P_f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots \quad (1.1)$$

$$= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \quad (1.2)$$

die Taylor-Reihe von  $f$  mit Entwicklungsstelle  $a$ .

Hierbei bezeichnet

- $n!$  die Fakultät  $n! = 1 \cdot 2 \cdot \cdots \cdot n$  und
- $f^{(n)}(a)$  die  $n$ -te Ableitung von  $f$  an der Stelle  $a$ , wobei man  $f^{(0)} := f$  setzt.

---

<sup>1</sup>Brook Taylor (\* 18. August 1685 in Edmonton, Middlesex; † 29. Dezember 1731 in Somerset House, London) war ein britischer Mathematiker. Nach ihm wurde u.a. die Taylorreihe benannt.

### 1.3. Eigenschaften

Die Taylorreihe  $P_f(x)$  zur Funktion  $f(x)$  ist eine Potenzreihe  $P$  in  $x$ . Im Fall einer analytischen Funktion  $f$  hat die Taylor-Reihe in jedem Punkt  $a$  einen positiven Konvergenzradius  $r$  und stimmt in ihrem Konvergenzbereich mit  $f$  überein, d.h. es gibt ein  $r > 0$ , sodass für  $|x - a| < r$  gilt

$$P_f(x) = \sum_{n=0}^{\infty} a_n (x - a)^n = f(x) \quad (1.3)$$

. Außerdem stimmen die Ableitungen der Reihe im Entwicklungspunkt  $a$  mit den tatsächlichen Ableitungswerten überein:

$$P_f^{(k)}(a) = f^{(k)}(a) \quad (1.4)$$

.

### 1.4. Konstruktion

Damit die Ableitungen der Reihe im Entwicklungspunkt  $a$  mit den tatsächlichen Ableitungswerten übereinstimmen, sind die Koeffizienten  $a_n$  der Taylorreihe wie folgt konstruiert:

$$a_n = \frac{f^{(n)}(a)}{n!} \Rightarrow P_f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x - a)^n \quad (1.5)$$

.

### 1.5. Entwicklung von $f(x)$ in eine Taylor Reihe

Bei gegebener Funktion

$$f(x) = \frac{1}{1+x} \quad (1.6)$$

und den Gleichungen 1.1 und 1.2 folgt für deren Ableitungen

$$f'(x) = -\frac{1}{(1+x)^2} \quad (1.7)$$

$$f''(x) = \frac{2(1+x)}{(1+x)^4} = \frac{2}{(1+x)^3} \quad (1.8)$$

$$f'''(x) = \frac{2(1+x)^4 - 8(1+x)(1+x)^3}{(1+x)^8} = -\frac{6}{(1+x)^4} \quad (1.9)$$

Hieraus lässt sich folgende Bildungsvorschrift für eine beliebige Ableitung bestimmen

$$f^n(x) = (-1)^n \frac{n!}{(1+x)^{n+1}} \quad (1.10)$$

hieraus folgt nach Gleichung 1.5

$$f(x) = P_f(x) = \sum_{n=0}^{\infty} (-1)^n \frac{n!}{n! (1+a)^{n+1}} (x-a)^n \quad (1.11)$$

$$= \sum_{n=0}^{\infty} (-1)^n \frac{(x-a)^n}{(1+a)^{n+1}} \quad (1.12)$$



## 2. Nachweis der hermiteschen Form von $H$

Eine hermitesche Matrix oder selbstadjungierte Matrix wird im mathematischen Teilgebiet der Linearen Algebra untersucht. Es handelt sich um eine spezielle Art von quadratischen Matrizen. Benannt sind diese nach dem Mathematiker Charles Hermite<sup>1</sup>.

### 2.1. Definition - hermitesch

Eine  $n \times n$ -Matrix  $A$  mit Einträgen in  $\mathbb{C}$  heißt hermitesch, wenn sie mit ihrer (hermitesch) Adjungierten  $A^*$ , also der transponierten und komplex konjugierten Matrix übereinstimmt. Das heißt, wenn

$$A = A^* = \overline{A}^T = \overline{A^T} \quad (2.1)$$

gilt.

Für die adjungierte Matrix finden sich auch die Bezeichnungen  $A^\dagger$  und  $A^H$ . Die Schreibweise  $A^*$  wird auch für die komplex konjugierte Matrix verwendet.

Für die Einträge einer hermiteschen Matrix gilt also:

$$a_{jk} = \overline{a_{kj}} \quad (2.2)$$

Anders formuliert ist eine Matrix  $A$  genau dann hermitesch, wenn ihre Transponierte gleich ihrer komplex Konjugierten ist, d.h.  $A^T = \overline{A}$ .

#### 2.1.1. direkte Folgen aus der Definition

- Der Realteil einer hermiteschen Matrix ist symmetrisch,  $\operatorname{Re}(a_{jk}) = \operatorname{Re}(a_{kj})$ , der Imaginärteil ist schiefsymmetrisch,  $\operatorname{Im}(a_{jk}) = -\operatorname{Im}(a_{kj})$ .
- Ist  $A$  hermitesch, dann ist  $iA$  schiefhermitesch.
- Die Hauptdiagonalelemente einer hermiteschen Matrix sind reell.
- Für reelle Matrizen fallen die Begriffe hermitesch und symmetrisch zusammen.
- Hermitesche Matrizen sind normal, d. h.  $A^* \cdot A = A \cdot A^*$

---

<sup>1</sup>Charles Hermite (\* 24. Dezember 1822 in Dieuze (Lothringen); † 14. Januar 1901 in Paris) war ein französischer Mathematiker.

## 2.2. Mathematischer Beweis für die gegebene Matrize 'H'

Mit der gegebenen Bildungsvorschrift

$$H := \left( \frac{1}{j+k+1} \right)_{j,k=0}^n \quad (2.3)$$

Gleichung 2.2 und den Aussagen aus 2.1.1 folgt, dass für  $H$  der folgende Zusammenhang gilt:

$$H = H^T \quad | \text{ da rein reelwertig } \quad (2.4)$$

$$H_{j,k} = H_{k,j} \quad | \text{ Indizees vertauschbar } \quad (2.5)$$

$$\left( \frac{1}{j+k+1} \right)_{j,k=0}^n = \left( \frac{1}{j+k+1} \right)_{k,j=0}^n \quad (2.6)$$

somit ist  $H$  symmetrisch und auch hermitesch.

### 3. Eigenwerte und Kondition von $H$

Die Erstellung der Matrix  $H$  wird durch ein Octave-Script<sup>1</sup> organisiert. Siehe hierzu A.1 und A.2.

Die berechneten Daten werden in zwei Dateien geschrieben,

---

```

1      # öffnen und schreiben in eine Datei
2      # mögliche Modi: r-read, w-write, a-append
3      fileId = fopen ( 'fileName', 'mode' );
4      # do something
5      ...
6      # done!
7      # schließen der Datei nicht vergessen
8      fclose ( fileId );
```

---

um sie später mit einem externen Programm (z.B. gnuplot) weiter verarbeiten zu können. Ihnen sind die Werte in Tabelle 3.1 entnommen.

Tabelle 3.1.: Eigenwerte & Konditionen für 3..10xn Matrizen

N											Kond.
3	0.00268	0.12232	1.40831								524.06
4	9.6702 e-05	6.7383 e-03	1.6914 e-01	1.5002 e+00							1.5514 e+04
5	3.2879 e-06	3.0590 e-04	1.1407 e-02	2.0853 e-01	1.5671 e+00						4.7661 e+05
6	1.0828 e-07	1.2571 e-05	6.1575 e-04	1.6322 e-02	2.4236 e-01	1.6189 e+00					1.4951 e+07
7	3.4939 e-09	4.8568 e-07	2.9386 e-05	1.0086 e-03	2.1290 e-02	2.7192 e-01	1.6609 e+00				4.7537 e+08
8	1.1115 e-10	1.7989 e-08	1.2943 e-06	5.4369 e-05	1.4677 e-03	2.6213 e-02	2.9813 e-01	1.6959 e+00			1.5258 e+10
9	3.4998 e-12	6.4609 e-10	5.3856 e-08	2.6730 e-06	8.7581 e-05	1.9789 e-03	3.1039 e-02	3.2163 e-01	1.7259 e+00		4.9315 e+11
10	1.0931 e-13	2.2667 e-11	2.1474 e-09	1.2290 e-07	4.7297 e-06	1.2875 e-04	2.5309 e-03	3.5742 e-02	3.4293 e-01	1.7519 e+00	1.6025 e+13

---

<sup>1</sup>GNU Octave ist eine freie Software zur numerischen Lösung mathematischer Probleme, wie zum Beispiel Matrizenrechnung, Lösen von (Differential-)Gleichungssystemen, Integration etc. Berechnungen können in Octave mit einer Skriptsprache durchgeführt werden, die weitgehend zu dem proprietären MATLAB kompatibel ist.

## 4. Numerische Lösung des LGS für eine fünfstellige Genauigkeit

Falls  $H$  eine  $n \times n$ -Matrix ist, wird von Octave Gaußelimination<sup>1</sup> verwendet, sonst wird via QR-Zerlegung eine Lösung im Sinne der kleinsten Fehlerquadrate berechnet.

Ist  $A$  schlecht konditioniert oder singulär, wird eine Warnung ausgegeben. Um zu zeigen, dass das Eliminationsverfahren nach Gauß ungeeignet ist haben wir hierfür eine eigene Funktion geschrieben, welche im Listing 4.1 zu sehen ist.

---

```

1  function [x] = gaussElim(A,b)
2  % File gaussElim.m
3  % This subroutine will perform Gaussian elimination
4  % on the matrix that you pass to it.
5  % i.e., given A and b it can be used to find x,
6  %      Ax = b
7  %
8  % To run this file you will need to specify several
9  % things:
10 % A - matrix for the left hand side.
11 % b - vector for the right hand side
12 %
13 % The routine will return the vector x.
14 % ex: [x] = gaussElim(A,b)
15 %      this will perform Gaussian elimination to find x.
16 %
17 %
18
19  N = max(size(A));
20
21 % Perform Gaussian Elimination
22
23  for j=2:N,
24      for i=j:N,
25          m = A(i,j-1)/A(j-1,j-1);
26          A(i,:) = A(i,:) - A(j-1,:)*m;
27          b(i) = b(i) - m*b(j-1);
28      end
29  end
30
31 % Perform back substitution
32
33  x = zeros(N,1);
34  x(N) = b(N)/A(N,N);
35
36  for j=N-1:-1:1,
37      x(j) = (b(j)-A(j,j+1:N)*x(j+1:N))/A(j,j);
38  end

```

---

Quelltext 4.1: Gauß-Eliminationsverfahren als Matlab-/Octavescript

---

<sup>1</sup>Johann Carl Friedrich Gauß (latinisiert Carolus Fridericus Gauss; \* 30. April 1777 in Braunschweig; † 23. Februar 1855 in Göttingen) war ein deutscher Mathematiker, Astronom, Geodät und Physiker mit einem breit gefächerten Feld an Interessen.

## 5. Visualisierung der Koeffizienten sowie des Taylorpolynoms

Das resultierende Polynom  $p(x)$  welches mittels der Gauß'elimination gefunden wurde ist in Abbildung 5.1 dargestellt. Wie zu sehen ist, oszilliert diese Lösung sehr stark und ist somit unbrauchbar.

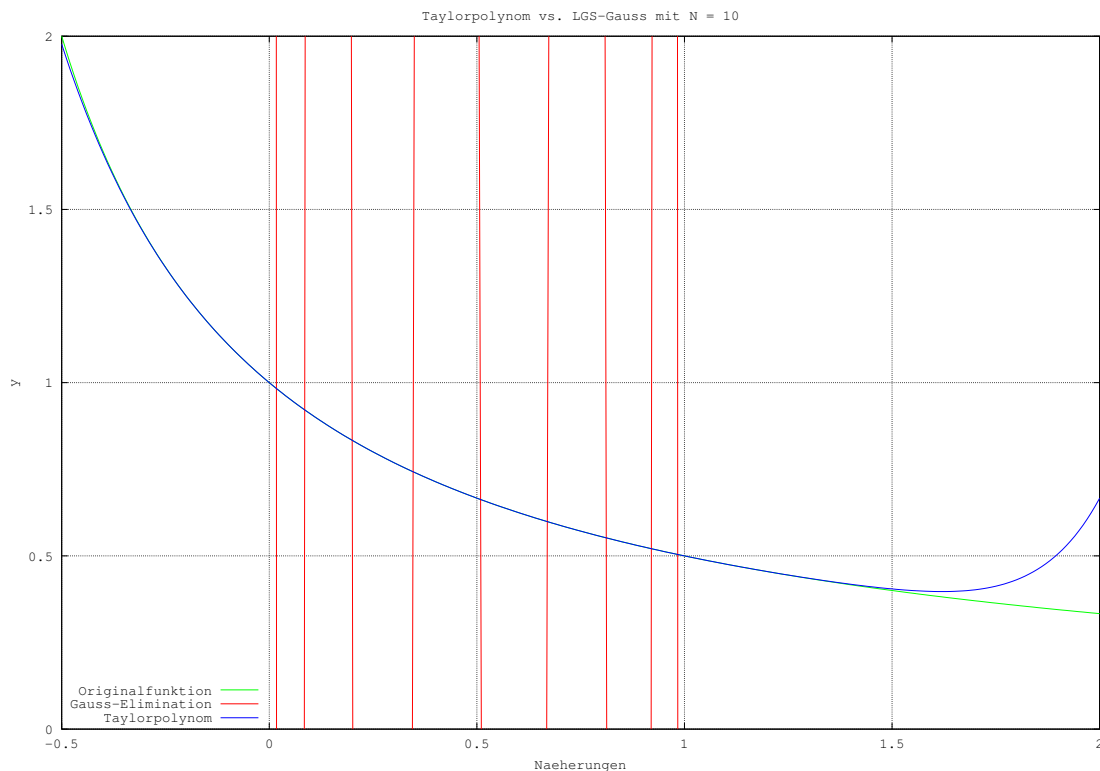


Abbildung 5.1.: Plot des Taylorpolynoms vs. Gauss-Elimination

## 5.1. Ausblick

Um das schlecht konditionierte Gleichungssystem dennoch lösen zu können, bietet sich das Regularisierungsverfahren nach Tikhonov<sup>1</sup> an. An dieser Stelle soll jedoch dieses nicht weiter erläutert, sondern nur dessen Implementierung in Octave aufgezeigt werden. Für das Listing siehe A.3 das hiermit resultierende Polynom ist in Abbildung 5.2 zu sehen.

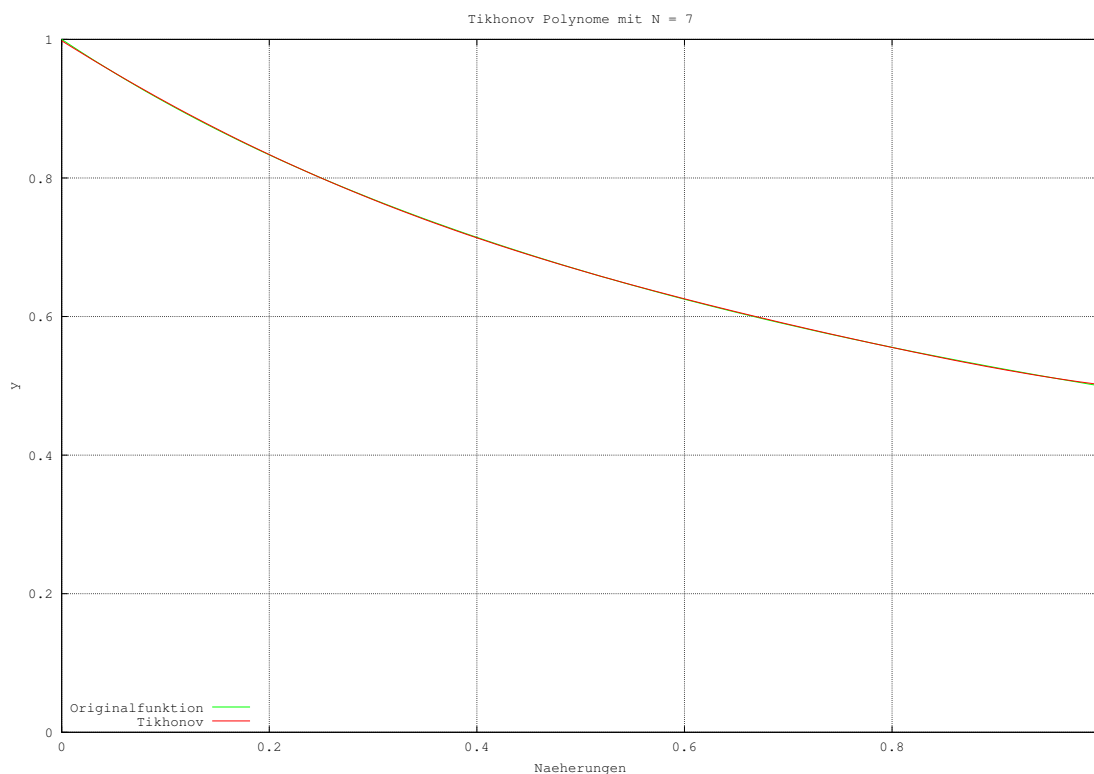


Abbildung 5.2.: Plot der Tikhonov Regularisierung

<sup>1</sup>Andrei Nikolajewitsch Tichonow (russisch, wiss. Transliteration Andrej Nikolaevič Tichonov; englische Transliteration Andrei Tikhonov; \*30. Oktober 1906 in Gschatsk; †7. November 1993 in Moskau) war ein russischer Mathematiker. Oft wird auch die Schreibung Tychonoff verwendet

## Abbildungsverzeichnis

5.1. Plot des Taylorpolynoms vs. Gauss-Elimination . . . . .	8
5.2. Plot der Tikhonov Regularisierung . . . . .	9

## Quelltextverzeichnis

4.1. Gauß-Eliminationsverfahren als Matlab-/Octavescript . . . . .	7
A.1. Aufruf über den Laufindex . . . . .	12
A.2. Berechnungen für eine nxn Matrix . . . . .	12
A.3. Regularisierung nach Tikhonov . . . . .	14



# A. Anhang

## A.1. Quelltexte

---

```

1 clear all;
2 clc;
3 #
4 # Öffnen der Datei und merken der File-Id, um später in Sie zu schreiben.
5 # ACHTUNG! Wird 'fopen' mit dem Parameter 'w' aufgerufen, so werden existente
6 # Dateien überschrieben.
7 #
8 fileIdData = fopen('data', 'w');
9 fileIdTask3 = fopen('aufgabe3', 'w');
10 #
11 # Iterationen um 3x3 bis 10x10 Matrizzen zu berechnen
12 #
13 for i = 3:10
14     if (!numa(i, fileIdData, fileIdTask3))
15         break;
16     endif
17     fflush(stdout);
18     pause;
19     clc;
20 endfor
21 # Schließen der Dateien
22 fclose(fileIdData);
23 fclose(fileIdTask3);

```

---

### Quelltext A.1: Aufruf über den Laufindex

---

```

1 function retVal = numa(N, fileIdData, fileIdTask3)
2     retVal = false; # default Zuweisung
3
4     if (isnumeric(N) && is_valid_file_id(fileIdData) && is_valid_file_id(
5         fileIdTask3))
6         N--;
7         entwPkt = 0.5; # und dessen Entwicklungspunkt
8         xmin = -0.5;
9         xmax = 2;
10        xval = [xmin:0.01:xmax];
11        ymin = 0;
12        ymax = 2;
13        #
14        # Erzeugen der Matrix
15        #
16        for i = 0:N
17            for j = 0:N
18                H(i+1, j+1) = 1/(1 + i + j);
19            endfor
20        endfor
21        # Anlegen des Lösungsvektors
22        R(1) = log(2); # r0 ist gleich ln2
23        for i = 1:N
24            sum = 0;
25            for j = 1:i
26                sum = sum + ((-1)^j)/j;

```

```

26     endfor
27     R(i+1) = (-1)^i * (log(2) + sum);
28 endfor
29 # Lösungsvektor mit 5-Stelliger Genauigkeit
30 Rtrim(1) = 0.69315;
31 for i = 1:N
32     sum = 0;
33     for j = 1:i
34         sum = sum + ((-1)^j)/j;
35     endfor
36     Rtrim(i+1) = (-1)^i * (0.69315 + sum);
37 endfor
38 #
39 # Lösen des LGS
40 #
41 solved = H\R';
42 printf("-----\n");
43 printf("-----N=%2d-----\n", N+1);
44 printf("-----\n");
45 printf("Lösungsvector mit voller Genauigkeit:\n");
46 disp(solved')
47 #
48 printf("\nLösungsvector mit 5-stelliger Genauigkeit:\n");
49 solvedTrim = gaussElim (H, Rtrim);
50 disp(solvedTrim')
51 #
52 # Ausgabe
53 #
54 printf ("\nMatrix 'H' %ix%i:\n", N+1, N+1)
55 disp (H)
56 commentLine = strcat ("#\n# 'H' Matrix-", num2str (N+1), "x", num2str (N+1),
57     "\n\n#");
58 fdisp (fileIdData, commentLine);
59 fdisp (fileIdData, H);
60 printf ("\nzugehörige Lösungsvektoren 'r' und 'rTrim':\n")
61 format long; # größeres Anzeigeformat
62 disp (R)
63 disp (Rtrim)
64 format short; # normales Anzeigeformat
65 commentLine = "#\n# 'R' Lösungsvektor\n#";
66 fdisp (fileIdData, commentLine);
67 fdisp (fileIdData, R);
68 commentLine = "#\n# Lösung des LGS\n#";
69 fdisp (fileIdData, commentLine);
70 fdisp (fileIdData, solved');
71 printf ("\nEigenwerte:\n")
72 lambda = eig (H);
73 disp (lambda)
74 commentLine = strcat ("#\n# Eigenwerte der Matrizze 'H' -", num2str (N+1), "x",
75     "\n", num2str (N+1), "\n#");
76 fdisp (fileIdTask3, commentLine);
77 fdisp (fileIdTask3, lambda'); # lambda ist transponiert, um besser lesbar in
78     eine Datei zu schreiben
79 printf ("\nKondition:\n")
80 cond = abs(max(lambda))/abs(min(lambda));
81 disp (cond)
82 commentLine = "#\n# Kondition\n#";
83 fdisp (fileIdTask3, commentLine);
84 fdisp (fileIdTask3, cond);
85 printf ("\n")
86 #

```

---

```

84     # Berechnen des Taylorpolynoms
85     #
86     taylor = zeros (1, size (xval, 2));
87     for i = 1:size (xval, 2)
88         for n = 0:N+1
89             taylor(i) += (-1)^n * ( (xval(i)-entwPkt)^n / ((1+entwPkt)^(n+1)) );
90         endfor
91     endfor
92     solved = solved(end:-1:1); # Potenzen umsortieren
93     solvedTrim = solvedTrim(end:-1:1);
94     # berechne die ursprüngliche Funktion
95     ref = zeros (1, N);
96     for n = 1:size (xval, 2)
97         ref(n) = 1/(1+xval(n));
98     endfor
99     #
100    # Plot des Polynoms
101    #
102    printf("Zum Anzeigen des Plots Taste drücken...");
103    pause;
104    plotPdf (xval, ref, solvedTrim, taylor, xmin, xmax, ymin, ymax); # Aufruf des
        externen Plot Scripts
105    printf("\nZum Fortfahren Taste drücken...");
106    #
107    # Return bool 'true'
108    #
109    retVal = true;
110    else
111        printf("Fehlerhafter Funktionsaufruf!\nSyntax: function(numericValue, fileId
            , fileId)\n");
112    endif
113 endfunction

```

---

### Quelltext A.2: Berechnungen für eine nxn Matrix

---

```

1  clear all;
2  clc;
3  N = 7;
4  xmin = 0;
5  xmax = 1;
6  xRange = [xmin:0.01:xmax];
7  # berechne die ursprüngliche Funktion
8  reference = zeros (1, N);
9  for n = 1:size(xRange, 2)
10     reference(n) = 1/(1+xRange(n));
11 end;
12 # Hilbert Matrix
13 for i = 0:N-1
14     for j = 0:N-1
15         A(i+1, j+1) = 1/(1 + i + j);
16     end;
17 end;
18 # Lösungsvektor
19 b(1) = 0.69315;
20 for i = 1:N-1
21     sum = 0;
22     for j = 1:i
23         sum = sum + ((-1)^j)/j;
24     end;
25     b(i+1) = (-1)^i * (0.69315 + sum);

```

---

```

26 end;
27 # Singulärwertzerlegung
28 m = rows (A); n = columns (A);
29 [U, S, V] = svd (A);
30 # Berechnung des Regularisierungsparameters alpha
31 alpha = 1/100;
32 delta = log(2) - 0.69315;
33 p = 1.01;
34 do
35     F = 0;
36     alpha = alpha/p;
37     for index = 1:N
38         F = F + alpha^2/((alpha+(S(index, index))^2)^2)*(abs(dot(b,V(:, index))))^2;
39     end;
40     F = F - delta^2;
41 until F < 0
42 F
43 alpha
44 # Thikonov decomposition mit neu gewähltem alpha
45 sum=0;
46 alpha=1e-10; # Parameter angepaßt, um Oszillationen zu reduzieren
47 for j=1:N
48     sum= sum.+S(j, j)/(alpha+S(j, j)^2)*dot(b,V(:, j))*U(:, j);
49 end;
50 x=flipud(sum)
51 # Plot
52 figure;
53 hold on;
54 grid;
55 plot (xRange, polyval (x, xRange), "r")
56 plot (xRange, reference', "g")
57 hold off;

```

---

Quelltext A.3: Regularisierung nach Tikhonov