Technische Universität Ilmenau
Fakultät für Mathematik
und Naturwissenschaften
Institut für Mathematik
http://www.mathematik.tu-ilmenau.de/Math-Net/index_de.html

Postfach 10 05 65
D - 98684 Ilmenau
Germany
Tel.: 03677/69 3267
Fax: 03677/69 3272
Telex: 33 84 23 tuil d.
email: werner.neundorf@tu-ilmenau.de

Preprint No. M 10/03

Spezielle Aspekte zu CAS Maple und Matlab

Rechengenauigkeit, Listen, Felder, Faktorisierungen,
Dateiarbeit, TP → Maple, Maple → Matlab
mit Beispielen

Werner Neundorf

Juni 2003

‡MSC (2000): 65-01, 65-04, 65F05, 65F10, 65Y99, 68U99, 68Q25

Zusammenfassung

Es gibt viele Möglichkeiten, mathematische und andere Aufgabenstellungen mit Computeralgebrasystemen wie Maple, Matlab, *Mathematica* oder Derive zu behandeln und zu lösen. Dazu findet der Leser eine immense Fülle von Informationen und Literatur sowie Demonstrationen und Hilfen in den Systemen selbst.
Trotzdem zeigt sich, dass zu spezifischen Fragestellungen eine detaillierte Aufbereitung der Sachverhalte sowie ihre Darstellung und Erläuterung sehr nützlich erscheint. So sind in dieser Arbeit eine Reihe von wichtigen Problemen mit den CAS Maple und Matlab untersucht worden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Naturgemäß sind Publikationen, die auf Software Bezug nehmen und diese für spezielle Aufgabenstellungen verwenden, einer ständigen Gefahr ausgesetzt. Upgrades oder Updates von Software sowie unterschiedliche Versionen für verschiedene Rechnerplattformen führen unter Umständen dazu, dass die in den Arbeiten dargestellten und mit der Software zusammenhängenden Sachverhalten nicht mit der Version übereinstimmen, die dem Leser zur Verfügung steht.
Die CAS erfahren eine ständige Weiterentwicklung und werden immer größer und leistungsfähiger. Programme und Arbeitsblätter, die unter älteren Versionen entstanden und gelaufen sind, sollte man bezüglich der Möglichkeiten und Veränderungen in neuen Versionen stets kritisch begutachten und eventuell anpassen. Das betrifft auch die breite Palette der Menüfunktionen.
Alles dies trifft auch auf das CAS Maple 8 zu. Manche von früheren Maple-Versionen erzeugte Resultate haben sich in Darstellung und Inhalt geändert. Darüber hinaus ist es ein ständiges Anliegen der Hersteller, Unzulänglichkeiten und Fehler in den Programmen und ihren Entwicklungsumgebungen zu beseitigen.
Es sollen zunächst nur einige Aspekte und Probleme zu **Maple 8** angeschnitten werden. Detaillierte Untersuchungen erfolgen in den weiteren Abschnitten.

- So berechnet die aktuelle Version des Programms einige Grenzwerte, die früher ungelöst blieben.

- Die Grenzwerte gemäß

  ```
  > Limit((1+1)*k,k=infinity): %=value(%);
    Limit(((-1)^(2*k)+1)*k,k=infinity): %=value(%);
  ```

  sind

  $$\lim_{k\to\infty} 2k \;=\; \infty$$

  $$\lim_{k\to\infty} \left((-1)^{(2k)}+1\right) k \;=\; undefined$$

  während als zweiter Wert eigentlich auch ∞ erwartet wird.

- Als Grenzwert der Folge

```
> a:=k->piecewise(k mod 3 =1,1,k mod 3 =2,2,3);
  Limit(a(k),k=infinity): %=value(%);
```

ergibt sich nur einer der drei Häufungspunkte

$$\lim_{k\to\infty} \begin{cases} 1 & k=1 \\ 2 & k=2 \\ 3 & otherwise \end{cases} = 3.$$

- Mit Stellen, wo eine Funktion an Glattheit verliert, gibt es einige Probleme mit der `piecewise`-Funktion.
Um die Menge der Unstetigkeitsstellen der Betragsfunktion zu ermitteln, wird man bei Verwendung des Kommandos `discont(abs(x),x);` auf die leere Menge { } kommen. Benutzt man aber `discont(piecewise(x<0,-x,x),x);` oder `discont(piecewise(x<0,-x,x>0,x,0),x);`, so ist das Ergebnis {0} und damit falsch. Auch das numerische Auffinden der Unstetigkeitsstellen für reelle Argumente mit `fdiscont` hilft nicht.

```
> iscont(abs(x),x);
  discont(abs(x),x);
```

$$true$$
$$\{\ \}$$

```
> iscont(piecewise(x<0,-x,x),x);
  iscont(piecewise(x<0,-x,x>0,x,0),x);  # analog

  discont(piecewise(x<0,-x,x),x);
  discont(piecewise(x<0,-x,x>0,x,0),x=-1..1,0.000001);
  fdiscont(piecewise(x<0,-x,x),x);
  fdiscont(piecewise(x<0,-x,x>0,x,0),x=-1..1,0.000001);
```

$$true$$
$$true$$
$$\{0\}$$
$$\{0\}$$
$$[-0.236329365582496776\,10^{-6}\,..\,0.295917599188524856\,10^{-6}]$$
$$[-0.241614831437123905\,10^{-6}\,..\,0.290634315037803701\,10^{-6}]$$

- Die cos-Funktion ist Maple's Liebling.
Eine Vereinfachung einer Formel mit trigonometrischen Funktionen gemäß `simplify(1/(sin(x)^2)^(3/2));` dürfte eigentlich nichts bringen, aber das Ergebnis bevorzugt die cos-Funktion und ist

$$\frac{1}{(1-\cos(x)^2)^{(3/2)}}$$

- Zur Vereinfachung des folgenden trigonometrischen Ausdrucks bedarf es der Einbeziehung der zusätzlichen Konvertierungsroutinen für diese Funktionenklasse. Einfache Versionen greifen nicht.

```
> t:=1/sqrt(1+tan(x)^2);

  simplify(t);
  simplify(t,symbolic);
  simplify(convert(t,sincos));
  simplify(convert(t,sincos),symbolic);
  #  csgn(z) = 1, if Re(z)>0 or Re(z)=0 and Im(z)>0
  #           -1, if Re(z)<0 or Re(z)=0 and Im(z)<0
```

$$t := \frac{1}{\sqrt{1+\tan(x)^2}}$$

$$\frac{1}{\sqrt{1+\tan(x)^2}}$$

$$\frac{1}{\sqrt{1+\tan(x)^2}}$$

$$\mathrm{csgn}\left(\cos\left(\frac{|x|^2}{x}\right)\right)\cos(x)$$

$$\cos(x)$$

Das gleiche trifft auf den Ausdruck in Bruchform zu.

```
> s:=(sqrt(a[1]^4*b[0]^2*a[2]^4/b[1]^6)*
      sqrt(b[0]^3*a[1]^3*sqrt(a[1]^4*b[0]^2*a[2]^4/b[1]^6)*a[2]^4)*
      b[1]^8*sqrt(a[1]*b[0]))/(a[2]^4*a[4]^2*b[0]^3*a[1]^4);

  simplify(s);                          # keine Vereinfachungen
  simplify(convert(s,rational)):
  simplify(s,symbolic);                 # Vereinfachungen
  simplify(convert(s,rational),symbolic):
```

$$s := \frac{\sqrt{\frac{a_1^4 b_0^2 a_2^4}{b_1^6}}\sqrt{b_0^3 a_1^3\sqrt{\frac{a_1^4 b_0^2 a_2^4}{b_1^6}}\,a_2^4\;b_1^8\sqrt{a_1 b_0}}}{a_2^4 a_4^2 b_0^3 a_1^4}$$

$$\frac{\sqrt{\frac{a_1^4 b_0^2 a_2^4}{b_1^6}}\sqrt{b_0^3 a_1^3\sqrt{\frac{a_1^4 b_0^2 a_2^4}{b_1^6}}\,a_2^4\;b_1^8\sqrt{a_1 b_0}}}{a_2^4 a_4^2 b_0^3 a_1^4}$$

$$\frac{a_2 b_1^{(\frac{7}{2})} a_1\sqrt{b_0}}{a_4^2}$$

- Einschränkungen gibt es bei Verwendung des Pfeiloperators in Funktionsdefinitionen im Gegensatz zu `unapply`, denn

```
> f:=x->x^3+x+1:
  Diff(f(x),x)=diff(f(x),x);

  fs1:=x->diff(f(x),x);        # fuehrt zu Fehler
  fs1(2);                      # fehlerhaft

  fs2:=unapply(diff(f(x),x),x);
  fs2(2);
```

liefert

$$\frac{d}{dx}(x^3 + x + 1) = 3x^2 + 1$$

$$fs1 := x \rightarrow \mathrm{diff}(\mathrm{f}(x), x)$$

```
Error, (in fs1) wrong number (or type) of parameters in function diff
```

$$fs2 := x \rightarrow 3x^2 + 1$$

$$13$$

- Manche Optionen und Menüfunktionen sind weggefallen, andere sind neu.

- Bei bedingten und Schleifenanweisungen haben sich die schließenden Klammern verändert, so z. B `od` → `end do`, `fi` → `end if`.

- Der Verknüpfungsoperator (concatenation) `.` für Zeichenketten (string) in den älteren Versionen wurde ersetzt durch das Doppelzeichen ||.
  Er wurde auch für die Bildung sogenannter mehrstufiger Bezeichner (Punktbezeichner) verwendet, wie sie z. B. auch bei Datensatzvariablen in Pascal auftreten.
  Als erstes Beispiel für die Verknüpfung nehmen wir die Bildung von Formaten im Zusammenhang mit der Ausgabe von Vektoren der Dimension n und einigen Zusatzinformationen. Wir verwenden sowohl die `cat`-Funktion (concatenating expressions) als auch das Doppelzeichen ||.

```
> fh1:=‘%+13.9f‘;       # Ausgabeformate einstellen
  fh:=fh1;
  for i from 2 to n do
   fh:=cat(fh,‘ ‘,fh1);
  end do;
  format1:=‘%2d                          ‘||fh1||‘     ‘||fh||‘\n‘;
  format2:=‘%2d     ‘||fh1||‘      ‘||fh1||‘     ‘||fh||‘\n‘;
  format3:=‘xs                                        ‘||fh||‘\n‘;
```

Als zweites Beispiel betrachten wir die Bernstein-Polynome n-ten Grades

$$B_j^n(t) \;=\; \binom{n}{j} t^j (1-t)^{n-j}, \quad j = 0, 1, ..., n,$$

die bei der Approximation verwendet werden.
Mit dem Punktbezeichner in Maple V notieren wir

```
> # Bernstein-Polynome Bnj(t), j=0,1,...,n
  n:=4;
  for j from 0 to n do
   Bn.j:=unapply(binomial(n,j)*t^j*(1-t)^(n-j),t);
  od;
```

und erhalten

$$\begin{gathered} n := 4 \\ Bn0 := (t \to 1-t)^4 \\ Bn1 := t \to 4t(1-t)^3 \\ Bn2 := t \to 6t^2(1-t)^2 \\ Bn3 := t \to 4t^3(1-t) \\ Bn4 := (t \to t)^4 \end{gathered}$$

wobei die Notation der Ergebnisse für $B_0^n(t)$ und $B_4^n(t)$ nicht unbedingt günstig erscheint.

Mit dem neuen Symbol || in Maple 8 ist nun

```
> n:=4;
  for j from 0 to n do
   Bn||j:=unapply(binomial(n,j)*t^j*(1-t)^(n-j),t);
  end do;
```

und man erhält mit den korrigierten Darstellungen für $B_0^n(t)$ und $B_4^n(t)$

$$\begin{gathered} n := 4 \\ Bn0 := t \to (1-t)^4 \\ Bn1 := t \to 4t(1-t)^3 \\ Bn2 := t \to 6t^2(1-t)^2 \\ Bn3 := t \to 4t^3(1-t) \\ Bn4 := t \to t^4 \end{gathered}$$

- Auffallend sind auch unverständliche Ungenauigkeiten bei der numerischen Berechnung von Ausdrücken, z. B. für Wurzeln mit `Digits:=10:` (entspricht dem *real*-Format).
  Andererseits werden Fehler und Problemsituationen kontinuierlich beseitigt und die in den Kommandos enthaltenen Algorithmen verbessert.

- Dass es Unterschiede zwischen der Behandlung von Folgen und Listen gibt, z. B. bei den Kommandos `op` und `nops`, daran hat man sich schon gewöhnt. Ungewöhnlich ist es jedoch, dass es zwar lange Listen gibt, aber die Indizierung der Listenelemente, wie dies bei Vektorkomponenten üblich ist, an der Grenze 100 sich verändert.

- Ein Beispiel noch zur Dateiarbeit.
  Resultate speichert man oft in eine Textdatei außerhalb des Hauptspeichers. Dazu benutzt man eine String-Variable für den Dateinamen mit eventueller Angabe des Laufwerks und Verzeichnispfads.

```
> dd:=‘C:/D/Neundorf/Maple4/aus.res‘:
  # auch moeglich
  # dd:="C:/D/Neundorf/Maple4/aus.res":
  file:=fopen(dd,WRITE):
  fprintf(file,‘%9.6f‘,12.345678);
  fclose(file):
```

  Nun soll aber der Dateiname in der Parameterliste einer Prozedur stehen.

```
> Ausgabe1:=proc(datei::string)
            local file1;
   file1:=fopen(datei,WRITE):
   fprintf(file1,‘%9.6f‘,12.345678):
   fclose(file1):
   0:       # Rueckgabewert
  end;
```

  Dann ist ein aktueller Dateiname festzulegen und die Prozedur damit aufzurufen.

```
> dd1:="C:/D/Neundorf/Maple4/aus1.res":
  erg:=Ausgabe1(dd1);
```

  Jedoch führt die Dateinamensdeklaration mit anderen Stringklammern

```
dd1:=‘C:/D/Neundorf/Maple4/aus1.res‘:
```

  zur Fehlermeldung

```
Error, invalid input: Ausgabe1 expects its 1st argument, datei,
to be of type string, but received C:/D/Neundorf/Maple4/aus1.res
```

Diese Palette lässt sich beliebig fortsetzen. Einiges davon sowie weitere wichtige Fragestellungen und Zusammenhänge werden in dieser Arbeit eingehender untersucht.

Unter diesem Aspekt sind auch die Informationen und Arbeitsblätter unter Maple V in [9] und [10] zu sehen und gegebenenfalls für Maple 8 zu modifizieren.

Im Preprint *Lösungsmethoden mit Maple. Betrachtung von Fehlern und Kondition sowie Darstellungsmöglichkeiten* [15] wurden zu einigen ausgewählten Aufgabentypen verschiedene Methoden vorgestellt sowie entsprechende Möglichkeiten, schöne

Effekte, aber auch Besonderheiten und Grenzen bei der Nutzung von diversen Programmen und Maple 8 diskutiert.

Dabei handelt es sich um
- Lösung von linearen Gleichungssystemen,
- Matrixeigenschaften, wie Norm, Kondition, Eigenwerte,
- korrekt gestellte Aufgaben,
- rekursive und iterative Algorithmen,
- Veranschaulichung des Stellenwertsystems und Operationen darin,
- interne Zahlendarstellung und Zahlenformate,
- Subtraktionskatastrophe,
- Termumformungen, mathematische und numerische Äquivalenz,
- Fehlerfortpflanzung und Fehleranalyse,
- Stabilität,
- Nutzung von Gleichungen und Funktionsgrafen, Kurvendiskussion,
- algebraische, geometrische und grafische Elemente, Animation,
- Darstellung von Formeln und Tabellen,
- Datenspeicherung und -manipulation,
- Verarbeitung von großen Datenmengen,

Geht es sich bei den zu lösenden Problemen um akademische oder solcher mit moderater Größenordnung, so bietet sich der Einsatz des CAS Maple 8 an. Die symbolischen Rechnungen, die unter Verwendung einer exakten Arithmetik auch exakt durchgeführt werden, sind dann mit vertretbarem Zeitaufwand machbar. Jedoch kann z. B. eine wiederholte symbolische Matrix-Vektor-Multiplikation, wie sie in der linearen Algebra ständig auftritt, für große Felddimensionen erhebliche Zeit kosten.
Hier und bei typischen numerischen Algorithmen ist dann die Verwendung der Gleitpunktarithmetik zu empfehlen, wie das in Matlab geschieht.

Die Softwarewerkzeuge verfügen heute über zahlreiche tutorielle Elemente und sie können in vielfältiger Weise zu tiefergreifenden Überlegungen und neuen Ideen führen. So bieten sich insbesondere der Einsatz von CAS wie Maple (vgl. [9], [10], [15]), Matlab (vgl. [17] – [20]), *Mathematica* oder Derive an. Auch der gegenseitige Zugriff dieser Systeme erschließt neue Möglichkeiten und Potenzen.

Die hier durchgeführten Überlegungen sollen nunmehr an einigen ausgewählten Beispielen demonstriert werden. Sie beinhalten zahlreiche Anwendungsfälle, Probleme, Begriffe, Eigenschaften, Darstellungen und Verfahren.

Der Leser findet dieses Preprint als Postscript-File *maple4.ps* sowie die Maple *mws*-Arbeitsblätter oder die Matlab *m*-Files sowie weitere Dateien auf der persönlichen Homepage im Internet.

Homepage
http://www.mathematik.tu-ilmenau.de/~neundorf/index_de.html

Navigator → Publications → Computeralgebra → MAPLE4

Kapitel 2

Anwendungen und Demonstrationen

In diesem Preprint wollen wir im Zusammenhang mit Maple 8, teilweise im Vergleich mit Matlab, zu folgenden Themenbereichen einige Erläuterungen machen und entsprechende Teste durchführen.

- Genauigkeit bei der Berechnung von Ausdrücken mit Wurzeln,
- Verarbeitung langer Listen sowie Folgen,
- Matrixgenerierung,
- Felder und das Kommando `evalm`,
- Produkte, Differenzen und Potenzen mit Matrizen und Vektoren,
- Informationen und Tests zu Systemkomponenten von Rechnern,
- Aufwand und Zeitmessungen bei der Generierung von Matrizen sowie Matrix-Vektor-Multiplikation,
- LU-Faktorisierung,
- Orthogonalisierung und QR-Faktorisierung,
- Grafikdateien in Turbo Pascal und ihre Einbeziehung in Maple-Kommandos für grafische Ausgaben,
- Varianten der Dateiarbeit in Maple,
- Nutzung von Matlab-Möglichkeiten in Maple,
- diverse Beispiele.

2.1 Rechengenauigkeit

Dass Ungenauigkeiten bei der numerischen Berechnung von Ausdrücken mit Wurzeln sowie numerischen Auswertung von Funktionen/Kommandos entstehen, ist verbunden mit der benutzten Gleitpunktarihmetik (GPA) und dem Rundungsregime. Dabei erwartet man von neueren Versionen von CAS im Allgemeinen bessere Lösungen. Beim Aufruf der Befehle oder in den Paketen verweisen die Hersteller oft explizit auf solche Veränderungen.
Der Nutzer sollte sich im Zweifelsfalle der Mühe unterziehen, das zu kontrollieren. An drei Beispielen wollen wir das Spektrum der Möglichkeiten anhand von Maple V und Maple 8 demonstrieren.

2.1.1 Auswertung von Ausdrücken

Eigentlich erwartet man bei der Rechnung mit `Digits:=10:` (entspricht dem *real*-Format und ist Standard), dass die letzte Ziffer der dargestellten dezimalen Mantisse aus einem vernünftigen Rundungseffekt zur nächsten Ziffer entsteht. Dass dies in Maple V schon funktioniert hat, aber in der neueren Version Maple 8 anders, aber eigentlich schlechter gelöst worden ist, kann einen Nutzer eher schockieren als zufriedenstellen. Es ist oft so, dass bei der Gleitpunktdarstellung (GP-Darstellung) von exakten Ergebnissen die erste Dezimale außerhalb der angezeigten Mantisse einfach abgeschnitten wird (siehe Normberechnung).
Als eines von zahlreichen Beispielen soll die Auswertung des Ausdrucks mit Bruch und Wurzel $(1+\sqrt{5})/2$ demonstriert werden.

Rechnungen in Maple (Datei: *calc1.mws*)

Maple V: vernuenftige Rundung

```
> t:=(1+sqrt(5))/2;  evalf(t);

  Digits:=8:  evalf(t);      # letzte Mantissenstelle gerundet
  Digits:=9:  evalf(t);
  Digits:=10: evalf(t);
  Digits:=11: evalf(t);
  Digits:=12: evalf(t);
  Digits:=20: evalf(t);  Digits:=10:
```

$$t := \frac{1}{2} + \frac{1}{2}\sqrt{5}$$

$$1.618033989$$

$$1.6180340$$

$$1.61803399$$

$$1.618033989$$

$$1.6180339888$$

$$1.61803398875$$

$$1.6180339887498948482$$

Gefahr des Rundungseffekts bei Weiterverarbeitung

```
> z:=(1+sqrt(5))/2;
  Digits:=10:
  z10:=evalf((1+sqrt(5))/2);
  Digits:=20:
  evalf(z-z10);
  1000*evalf(z-z10); # gefaehrlich, da gerundete Stelle in der Mantisse
                     # nach vorne rutscht
```

$$z := \frac{1}{2} + \frac{1}{2}\sqrt{5}$$

$$z10 := 1.618033989$$

$$-0.2501051518\ 10^{-9}$$

$$-0.2501051518000\ 10^{-6}$$

Maple 8: Warum Verschlechterung gegenueber Maple V?

```
> t:=(1+sqrt(5))/2;  evalf(t);
  Digits:=8:  evalf(t);
  Digits:=9:  evalf(t);
  Digits:=10: evalf(t);   # letzte Mantissenstelle falsch
  Digits:=11: evalf(t);
  Digits:=12: evalf(t);
  Digits:=20: evalf(t); Digits:=10:
```

$$t := \frac{1}{2} + \frac{\sqrt{5}}{2}$$

$$1.618033988$$

$$1.6180340$$

$$1.61803399$$

$$1.618033988$$

$$1.6180339888$$

$$1.61803398875$$

$$1.6180339887498948482$$

Groessere Gefahr des Rundungseffekts bei Weiterverarbeitung

```
> z:=(1+sqrt(5))/2;
  Digits:=10:
  z10:=evalf((1+sqrt(5))/2);
  Digits:=20:
  evalf(z-z10);
  1000*evalf(z-z10); # gefaehrlich, da falsche Stelle in der Mantisse
                     # nach vorne rutscht
```

$$z := \frac{1}{2} + \frac{\sqrt{5}}{2}$$

$$z10 := 1.618033988$$

$$0.7498948482\ 10^{-9}$$

$$0.7498948482000\ 10^{-6}$$

2.1.2 Auswertung von Funktionen und Kommandos

(1) Kommando `linsolve` zur Lösung von linearen Gleichungssystemen

Wir verwenden das Kommando für ein LGS mit der Hilbert-Matrix H als Koeffizientenmatrix. Diese ist zwar symmetrisch und positiv definit, aber schlecht konditioniert. So genügt es, ein kleines System zu betrachten. Wenn die Rechengenauigkeit `Digits:=10:` eingestellt ist und die Kondition $\text{cond}(H) = \mathcal{O}(10^t)$ beträgt, dann muss man in der Lösung des LGS einen Genauigkeitsverlust von ca. t Mantissenstellen einkalkulieren.
Maple 8 liefert dabei rund eine Dezimale mehr Genauigkeit als Maple V.

Rechnungen in Maple

LGS mit Hilbert-Matrix und exakter Loesung

```
> m:=4:
  A:=hilbert(m);
  x:=vector(m,[1$m]);
  b:=evalm(A&*x);

  evalf(cond(A,2));
```

$$A := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

$$x := [1, 1, 1, 1]$$

$$b := \left[\frac{25}{12}, \frac{77}{60}, \frac{19}{20}, \frac{319}{420}\right]$$

$$15513.73874$$

```
> AA:=evalm(A):
  AA[1,1]:=1.0:   # damit numerische Rechnung
  evalm(AA);
```

$$\begin{bmatrix} 1.0 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

Maple V

```
> Digits:=10:
  z:=linsolve(AA,b);    # bei jeden neuen Aufruf sind die ungenauen
                        # Dezimalen immer wieder anders
```

$$z := [1.000000119, 0.9999987500, 1.000002897, 0.9999981611]$$

Maple 8, eine Dezimalstelle genauer

```
> Digits:=10:
  z:=linsolve(AA,b);    # bei jeden neuen Aufruf sind die ungenauen
                        # Dezimalen immer wieder anders
```

$$z := [0.9999999630, 1.000000368, 0.9999991786, 1.000000508]$$

(2) Kommandos `norm` und `cond` aus dem Paket `linalg`

Diese beiden Kommandos waren in Maple V eine Schwachstelle und sind in der neuen Version korrigiert worden. Auch hier soll die symmetrische und positiv definite Hilbert-Matrix getestet werden.

Norm und Kondition mit Hilbert-Matrix

Maple V

Falsche Berechnung von Norm, Kondition fuer die Hilbert-Matrix $H(m),\ m = 4$

```
> Digits:=10:
  A:=hilbert(m):
  evalf(norm(A,2));    # ? = 1.500214280
  evalf(Eigenvals(evalm(A&*transpose(A))));
  [seq(sqrt("[i]),i=1..m)];
  inverse(A);
  evalf(norm(inverse(A),2));
  evalf(norm(A,2)*norm(inverse(A),2));
  evalf(cond(A,2));
```

$$0.00009670230404$$

$$[0.987717778\,10^{-8}, 0.00004540435963, 0.02860875214, 2.250642886]$$

$$[\ 0.00009938399157,\ 0.006738275716,\ 0.1691412195,\ 1.500214280\ ]$$

$$\begin{bmatrix} 16 & -120 & 240 & -140 \\ -120 & 1200 & -2700 & 1680 \\ 240 & -2700 & 6480 & -4200 \\ -140 & 1680 & -4200 & 2800 \end{bmatrix}$$

$$0.6665714447$$

$$0.00006445899451$$

$$0.00006445899451$$

Ausweg
Prozedur fuer die Berechnung der Spektralnorm ueber die EW von AA^T

```
> norm2:=proc(A::matrix)
   local n,i,B,EVB,seq_EVB;

   B:=evalm(A&*transpose(A));
   n:=linalg[rowdim](B);
   EVB:=evalf(Eigenvals(B));
   seq_EVB:=seq(EVB[i],i=1..n);
   sqrt(max(seq_EVB));
  end:

> norm2(A);          # richtig
  evalf(norm(A,2));  # falsch
```

$$1.500214280$$
$$0.00009670230404$$

Falsche Berechnung von Norm, Kondition fuer die Hilbert-Matrix $H(m),\ m \geq 4$

```
> Digits:=16:
  i:='i':
  j:='j':
  printf(` n          cond(H(n,n))\n`):

  for n from 1 to 10 do
   A:=hilbert(n):
   erg:=evalf(norm(A,2)*norm(inverse(A),2));
   printf(`%2d    %.15e  \n`,n,erg);
  end do:
```

```
 n          cond(H(n,n))
 1    1.000000000000000e+00
 2    1.928147006790397e+01
 3    5.240567775860608e+02
 4    6.445899449695157e-05
 5    2.098163633664969e-06
 6    6.688489585050706e-08
 7    2.103636249958449e-09
 8    6.554121158778911e-11
 9    2.027760335156716e-12
10    6.239748533756699e-14
```

Maple 8
Richtige Berechnung von Norm, Kondition fuer die Hilbert-Matrix $H(m),\ m = 4$

```
> Digits:=10:
  A:=hilbert(4):
  evalf(norm(A,2));
  evalf(Eigenvals(evalm(A&*transpose(A))));
  [seq(sqrt(%[i]),i=1..4)];
  inverse(A):
  evalf(norm(inverse(A),2));
  evalf(norm(A,2)*norm(inverse(A),2));
  evalf(cond(A,2));
```

$$1.500214280$$
$$[0.873843588\,10^{-8}, 0.00004540427505, 0.02860875198, 2.250642886]$$
$$[\ 0.00009347960141,\ 0.006738269440,\ 0.1691412190,\ 1.500214280\ ]$$
$$10341.01524$$
$$15513.73874$$
$$15513.73874$$

Anwendung der Prozedur fuer die Berechnung der Spektralnorm ueber EW von AA^T

```
> norm2(A);
  evalf(norm(A,2));
```

$$1.500214280$$
$$1.500214280$$

Richtige Berechnung von Norm, Kondition fuer die Hilbert-Matrix $H(m),\ m \geq 4$

```
> Digits:=16:
  i:='i': j:='j':
  printf(` n          cond(H(n,n))\n`):
  for n from 1 to 10 do
   A:=hilbert(n):
   erg:=evalf(norm(A,2)*norm(inverse(A),2));
   printf(`%2d    %.15e  \n`,n,erg);
  end do:
```

```
n          cond(H(n,n))              Rundungseffekt
1    1.000000000000000e+00
2    1.928147006790397e+01
3    5.240567775860608e+02
4    1.551373873893259e+04   <- 1.551373873893258 8222...e+04
5    4.766072502425608e+05
6    1.495105864013122e+07
7    4.753673549881789e+08   <- 4.753673549881789 9255...e+08
8    1.525757574164694e+10
9    4.931549269715421e+11
10   1.602628687021688e+13
```

2.2 Lange Listen und Folgen

Es wurde schon darauf hingewiesen, dass man mit beliebig langen Listen arbeiten kann, aber die Indizierung der Listenelemente, wie dies bei Vektorkomponenten üblich ist, an der Grenze 100 andere Eigenschaften erfährt.
Die Darstellungen in Maple sind selbstkommentierend, wobei diese in einige zusätzliche Informationen über Listen und Folgen eingebettet worden sind.

Rechnungen in Maple (Datei: *list1.mws*)

Kurze Liste

```
> l1:=[seq(k,k=1..10)];
  nops(l1);              # Anzahl der Listenelemente
  op(10,l1);             # Listenelement an der Stelle ...
  op(9..10,l1);
  l1[10];
  member(10,l1);         # true, falls Element in Liste ist
  l1[11];                # fehlerhaft
  l1[10]:=-1;
  l1;
```

$$l1 := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$
$$10$$
$$10$$
$$9, 10$$
$$10$$
$$true$$

```
Error, invalid subscript selector
```

$$l1_{10} := -1$$
$$[1, 2, 3, 4, 5, 6, 7, 8, 9, -1]$$

Liste und Folge

```
> l1;

  l1[1..nops(l1)];          # Liste
  op(l1);                   # Folge
  [op(l1)];                 # Liste
  seq(l1[i],i=1..nops(l1)); # Folge
```

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, -1]$$
$$[1, 2, 3, 4, 5, 6, 7, 8, 9, -1]$$
$$1, 2, 3, 4, 5, 6, 7, 8, 9, -1$$
$$[1, 2, 3, 4, 5, 6, 7, 8, 9, -1]$$
$$1, 2, 3, 4, 5, 6, 7, 8, 9, -1$$

Unterliste bzw. Streichen von Listenelementen

```
> l1;

  lu11:=subsop(5=NULL,l1);
  lu12:=subsop(1=NULL,5=NULL,10=NULL,l1);

  lu13:=[op(1..4,l1),op(6..10,l1)];
  lu14:=[op(l1[1..4]),op(l1[6..10])];
  lu15:=l1[1..nops(l1)-1];
  lu16:=l1[2..nops(l1)];
```

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, -1]$$
$$lu11 := [1, 2, 3, 4, 6, 7, 8, 9, -1]$$
$$lu12 := [2, 3, 4, 6, 7, 8, 9]$$
$$lu13 := [1, 2, 3, 4, 6, 7, 8, 9, -1]$$
$$lu14 := [1, 2, 3, 4, 6, 7, 8, 9, -1]$$
$$lu15 := [1, 2, 3, 4, 5, 6, 7, 8, 9]$$
$$lu16 := [2, 3, 4, 5, 6, 7, 8, 9, -1]$$

Lange Liste

```
> l2:=[seq(k,k=1..100)];
  nops(l2);
  op(100,l2);
  l2[100];

  l2[100]:=-1;
  l2;
```

$$l2 := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]$$

$$100$$
$$100$$
$$100$$

$$l2_{100} := -1$$

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, -1]$$

Lange Liste, mehr als 100 Elemente

```
> l3:=[seq(k,k=1..101)];
  l3[101];       # Abfrage moeglich
  l3[1];
  # bei mehr als 100 Listenelementen damit keine Zuordnung
  l3[101]:=-101;
  l3[1]:=-1;

  # Kontrolle der Inhalte
  l3;
  l3[101];
  l3[1];
```

$$l3 := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101]$$

$$101$$

$$1$$

```
Error, assigning to a long list, please use arrays
Error, assigning to a long list, please use arrays
```

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101]$$

$$101$$

$$1$$

Verwendung des Kommandos `op` fuer die Auswahl von Elementen bzw. Teillisten, man bemerke die runden Klammern bei Auswahl mehrerer Listenelemente.

```
> l4:=[seq(k,k=1..101)]:
  `nops(l4)`=nops(l4);          # Anzahl der Listenelemente

  `op(1,l4)`=op(1,l4);
  `op(1..2,l4)`=op(1..2,l4);
  `op(2..100,l4)`=op(2..100,l4);
  `op(101,l4)`=op(101,l4);
```

$$nops(l4) = 101$$

$$op(1,l4) = 1$$

$$op(1..2,l4) = (1,2)$$

op(2..100,l4) = (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100)

op(101,l4) = 101

Moeglichkeit des Zusammenfuegens von Teillisten

```
> l41:=[-1,op(2..100,l4),-101];
  l42:=[op(l41),op(l41)]:  # Verdoppeln der Liste
  `nops(l42)`=nops(l42);
  `l42[202]`=l42[202];
```

l41 := [-1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, -101]

nops(l42) = 202

l42[202] = -101

Folge

```
> s1:=seq(k,k=1..100);
  s1[1];                # indiziertes Element
  s1[100];
  op(s1);               # Elemente?
  nops(s1);             # Anzahl?
  member(10,s1);        # Zugehoerigkeit eines Elements zur Folge?
```

s1:= 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, . . . 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

1

100

```
Error, wrong number (or type) of parameters in function op
Error, wrong number (or type) of parameters in function nops
Error, wrong number (or type) of parameters in function member
```

```
> s1[1]:=-1;  s1[100]:=-100;       # Fehler
  s1:=-1,seq(s1[i],i=2..99),-100;  # mit Auswahl von Teilfolge
```

```
Error, cannot assign to an expression sequence
Error, cannot assign to an expression sequence
```

s1:= -1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, . . . 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, -100

2.3 Matrixgenerierung

Es sollen hier verschiedene Möglichkeiten der Definition von Feldern und ihrer Umsetzung in Maple aufgezeigt werden. Dabei erweisen sich die zahlreichen Kommandos/Funktionen in den Paketen **linalg** und **LinearAlgebra**, speziell zur linearen Algebra, als sehr nützlich.
Ihr Aufruf kann erfolgen gemäß

1. ```
   > with(linalg):
     command();
   ```
2. ```
   > with(linalg,command):
     command();
   ```
3. Langform
   ```
   > linalg[command](..):
   ```

Wir notieren gebräuchliche Varianten, aber das Spektrum der Möglichkeiten ist bei Weitem nicht ausgeschöpft. Viele Konstruktionen sind selbstkommentierend. Bei einigen Matrix-Vektor-Konstrukten werden auch die zu Grunde liegenden mathematisch-technischen Modelle kurz vorgestellt.

Rechnungen in Maple (Datei: *genarr1.mws*)

Felder

Ein- und mehrdimensionale Felder werden mit dem Kommando **array** erzeugt.
Matrizen **matrix** und Vektoren **vector** sind spezielle Felder.
Der Index ist aus einem festem Integer-Bereich zu waehlen und es erfolgt eine Kontrolle der Zulaessigkeit der Indizes.
Ein Feld kann initialisiert werden, entweder mit einer Liste von Daten oder ohne.

(1) Eindimensionales Felder

array(m..n), array(m..n,list)

Damit wird ein Feld mit den Indizes $m, m+1, ..., n$ erzeugt. Oft ist $m = 1$.

```
> A1 := array(0..3);
  type(A1,array);
  type(A1,matrix);
  A2 := array(0..3,[]);
  A3 := array(0..3,[2,4,6,8]);
```

$$A1 := \text{array}(0 \,..\, 3,\ [\ ])$$
$$true$$
$$false$$
$$A2 := \text{array}(0 \,..\, 3,\ [\ ])$$

$A3 :=$ array(0 .. 3, [
(0)=2
(1)=4
(2)=6
(3)=8
])

```
> A4 := array(1..3,[2,4,6]);  # wie array/vector behandelt
  A5 := vector(3);
  A5[1] := 1;
  A5;
  A6 := vector(3,[4,5,6]);
  A7 := vector([6,7,8]);
```

$$A4 := [2, 4, 6]$$
$$A5 := \text{array}(1 .. 3, [\ ])$$
$$A5_1 := 1$$
$$A5$$
$$A6 := [4, 5, 6]$$
$$A7 := [6, 7, 8]$$

Ausgaben

```
> print(A1);  # Komponenten nicht belegt
  # array
  print(A3);  # wie eval(A3);

  evalm(A3);  # Ausgabe von Feldbezeichner
  # array mit unterer Indexgrenze = 1
  print(A4);  # wie eval(A4); und  evalm(A4);
  # vector
  print(A5);  # wie evalm(A5);
  eval(A5);   # Ausgabe der belegten Komponenten (Werte) sowie
              # nicht belegten Komponenten mit Feldbezeichner "?"
  print(A6);
```

array(0 .. 3, [
(0) = $A1_0$
(1) = $A1_1$
(2) = $A1_2$
(3) = $A1_3$
])
array(0 .. 3, [
(0) = 2
(1) = 4
(2) = 6
(3) = 8
])

$$A3$$
$$[2,\ 4,\ 6]$$
$$[1,\ A5_2,\ A5_3]$$
$$[1,\ ?_2,\ ?_3]$$
$$[4,\ 5,\ 6]$$

Eindimensionales Feld

```
> n := 4;
  v := array(1..n);
  v[1] := a;
  for i from 2 to n do v[i] := a*v[i-1]; end do;
```

$$n := 4$$
$$v := \text{array}(1\ ..\ 4,\ [\ ])$$
$$v_1 := a$$
$$v_2 := a^2$$
$$v_3 := a^3$$
$$v_4 := a^4$$

```
> a := 3;
  v[1]; v[2];
  print(v);
```

$$a := 3$$
$$3$$
$$9$$
$$[a, a^2, a^3, a^4]$$

Anwendungen

(a) Sortieren einer Zahlenfolge mit Algorithmus *bubble sort*

```
> n := 10;
  v := array(1..n);
  for i to n do v[i] := n-i; end do:
  ‘old:‘; seq(v[i],i=1..n);
  for i to n-1 do
   for j from i+1 to n do
    if v[i] > v[j] then
     temp := v[i]; v[i] := v[j]; v[j] := temp; end if;
   end do;
  end do;
  ‘new:‘; seq(v[i],i=1..n);
```

$$n := 10$$
$$v := \text{array}(1\ ..\ 10,\ [\ ])$$

old:

9, 8, 7, 6, 5, 4, 3, 2, 1, 0

new:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

(b) Produkt von 2 Polynomen

Das Polynom $p_m(x) = a_0 + a_1x + ... + a_mx^m$ generiert die Liste $[a_0, ..., a_m]$ seiner Koeffizienten. Analog sei das Polynom $q_n(x)$ vom Grad n gegeben.
Die einfache Variante der Produktbildung $r(x) = p_m(x) \cdot q_n(x)$ ist wie folgt.

```
> pm := 1+2*x+3*x^2;
  qn := 1-2*x-3*x^2-4*x^3;
  m := degree(pm,x);
  n := degree(qn,x);
  la := [seq(coeff(pm,x,i),i=0..m)];
  lb := [seq(coeff(qn,x,i),i=0..n)];
  a := array(0..m,la);
  b := array(0..n,lb);
```

$$pm := 1 + 2x + 3x^2$$
$$qn := 1 - 2x - 3x^2 - 4x^3$$
$$m := 2$$
$$n := 3$$
$$la := [1, 2, 3]$$
$$lb := [1, -2, -3, -4]$$

$a :=$ array(0 .. 2, [
(0)=1
(1)=2
(2)=3
])
$b :=$ array(0 .. 3, [
(0)=1
(1)=-2
(2)=-3
(3)=-4
])

```
> m := nops(la)-1;  # Grad von pm
  n := nops(lb)-1;  # Grad von qn
```

$$m := 2$$
$$n := 3$$

Produkt

```
> c := array(0..m+n);
  for k from 0 to m+n do c[k] := 0; end do:
   for i from 0 to m do
    for j from 0 to n do
      c[i+j] := c[i+j] + a[i]*b[j];
      # c[i+j] := c[i+j] + la[i+1]*lb[j+1];
    end do:
   end do:

  'Koeffizienten c[0..m+n] des Produkts';
  [seq(c[k],k=0..n+m)];  # Koeffizienten in eine Liste
```

$$c := \text{array}(0 \,..\, 5, [\ ])$$
$$\textit{Koeffizienten c[0..m+n] des Produkts}$$
$$[1,\ 0,\ -4,\ -16,\ -17,\ -12]$$

(2) Zweidimensionale Felder

array(c..d,m..n), ***array(c..d,m..n,list)***

Damit wird ein Feld bzw. Matrix mit den Zeilenindizes $c, c+1, ..., d$ sowie den Spaltenindizes $m, m+1, ..., n$ erzeugt. Oft sind $c, m = 1$. Fuer hoeherdimensionale Felder funktioniert es analog.

```
> A1 := array(0..3,0..1);
  A2 := array(0..3,0..1,[]);
  A3 := array(0..3,0..1,[[2,4],[6,8],[9,10],[22,33]]);
```

$$A1 := \text{array}(0 \,..\, 3,\ 0 \,..\, 1,\ [\ ])$$
$$A2 := \text{array}(0 \,..\, 3,\ 0 \,..\, 1,\ [\ ])$$

$A3 :=$ array(0 .. 3, 0 .. 1, [
(0,0)=2
(0,1)=4
(1,0)=6
(1,1)=8
(2,0)=9
(2,1)=10
(3,0)=22
(3,1)=33
])

```
> A4 := array([[2,4,6],[1,2,3]]);  # wie array/matrix behandelt
  type(A4,array);
  type(A4,matrix);
```

```
A5 := matrix(2,3);
A5[1,1] := 1;
A5;
A6 := matrix(2,2,[[4],[5,6]]);
A7 := matrix([[6,7],[8,9]]);
```

$$A4 := \begin{bmatrix} 2 & 4 & 6 \\ 1 & 2 & 3 \end{bmatrix}$$

$$true$$

$$true$$

$$A5 := \mathrm{array}(1 \,..\, 2, 1 \,..\, 3, [\ ])$$

$$A5_{1,1} := 1$$

$$A5$$

$$A6 := \begin{bmatrix} 4 & A6_{1,2} \\ 5 & 6 \end{bmatrix}$$

$$A7 := \begin{bmatrix} 6 & 7 \\ 8 & 9 \end{bmatrix}$$

Ausgaben

```
> print(A1);  # Komponenten nicht belegt

  # array
  print(A3);  # wie eval(A3);
  evalm(A3);  # Ausgabe von Feldbezeichner

  # array mit unteren Indexgrenzen = 1
  print(A4);  # wie eval(A4); und  evalm(A4);

  # matrix
  print(A5);  # wie evalm(A5);
  eval(A5);   # Ausgabe der belegten Komponenten (Werte) sowie
              # nicht belegten Komponenten mit Feldbezeichner "?"

  print(A6);
  print(A7);
```

$$\mathrm{array}(0 \,..\, 3, 0 \,..\, 1, [$$
$$(0,0) = A1_{0,0}$$
$$(0,1) = A1_{0,1}$$
$$(1,0) = A1_{1,0}$$
$$(1,1) = A1_{1,1}$$
$$(2,0) = A1_{2,0}$$
$$(2,1) = A1_{2,1}$$
$$(3,0) = A1_{3,0}$$
$$(3,1) = A1_{3,1}$$
$$])$$

array(0 .. 3, 0 .. 1, [
(0,0) = 2
(0,1) = 4
(1,0) = 6
(1,1) = 8
(2,0) = 9
(2,1) = 10
(3,0) = 22
(3,1) = 33
])

$$A3$$

$$\begin{bmatrix} 2 & 4 & 6 \\ 1 & 2 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & A5_{1,2} & A5_{1,3} \\ A5_{2,1} & A5_{2,2} & A5_{2,3} \end{bmatrix}$$

$$\begin{bmatrix} 1 & ?_{1,2} & ?_{1,3} \\ ?_{2,1} & ?_{2,2} & ?_{2,3} \end{bmatrix}$$

$$\begin{bmatrix} 4 & A6_{1,2} \\ 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 7 \\ 8 & 9 \end{bmatrix}$$

Anwendungen

(a) Austausch von i-ter und j-ter Zeile einer Matrix

```
> m := 3;
  n := 4;
  A := matrix(m,n);
  for i to m do
   for j to n do A[i,j] := 1/(i+j-1);  # Hilbert-Matrix
   end do;
  end do;
  eval(A);
```

$$m := 3$$

$$n := 4$$

$$A := \text{array}(1 \text{ .. } 3,\ 1 \text{ .. } 4,\ [\ ])$$

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \end{bmatrix}$$

```
> i := 2; j := 3;
  if i<>j then
   for k to n do
    temp := A[i,k]; A[i,k] := A[j,k]; A[j,k] := temp;
   end do;
  end if;
  print(A);
```

$$i := 2$$
$$j := 3$$
$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

Sowohl die Hilbert-Matrix als auch der Zeilentausch werden durch Kommandos aus dem Paket **linalg** unterstuetzt.
(b) Hilbert-Matrix

```
> A:=hilbert(3);  # A:=linalg[hilbert](3);  #  nicht hilbert(m,n)
```

$$A := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

(c) Zeilentausch, A aus (a)

```
> B:=swaprow(A,2,3);  # B:=linalg[swaprow](A,2,3);
```

$$B := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

(3) Feldbezeichner

Man beachte, dass der Wert von A der Bezeichner des Felds A selbst ist, weil auch oft die Feldkomponenten noch keine konkreten Werte haben. Es gibt spezielle Kommandos zur Berechnung/Verarbeitung von Feldern. Wenn man Zuweisungen, Ausgaben oder Prozedurrueckgabe von Feldern machen will, dann benutze man das Kommando `print` oder eine der Berechnungsfunktionen `eval, evalm, evalf, ....` Dabei bemerkt man Unterschiede, die schon in den obigen Beispielen angedeutet worden

sind. Insbesondere, wenn ein **array** die unteren Indexgrenzen 1 hat, dann ist es auch eine **matrix**. Diese Doppeldeutung hat Auswirkung auf die Verwendung der Berechnungsfunktionen `eval, evalm`.

- ```
  # array
  print(A3); # wie eval(A3);
  evalm(A3); # Ausgabe von Feldbezeichner
  ```

- ```
  # array mit unteren Indexgrenzen = 1
  print(A4); # wie eval(A4); evalm(A4);
  ```

- ```
  # vector, matrix
  print(A5); # wie evalm(A5);
  eval(A5);  # Ausgabe der belegten Komponenten (Werte) sowie
             # nicht belegten Komponenten mit Feldbezeichner '?'
  ```

```
> A:=hilbert(3);
  whattype(hilbert), whattype(A);
  type(A,matrix), type(A,array);
  A;
  evalm(A);  # wie eval(A);
  evalf(A);
  evalf(evalm(A));     # Digits=10
```

$$symbol, symbol$$

$$true, true$$

$$A$$

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

$$A$$

$$\begin{bmatrix} 1. & 0.5000000000 & 0.3333333333 \\ 0.5000000000 & 0.3333333333 & 0.2500000000 \\ 0.3333333333 & 0.2500000000 & 0.2000000000 \end{bmatrix}$$

Traditionelle mathematische Notation mit Bezeichner und Gleichheitszeichen

```
> A:
  % = evalm(%);
```

$$A = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$
```

(4) Moeglichkeiten der Generierung von Matrizen und Vektoren

• Explizite Angabe

```
> A0 := matrix([[1,2,3],[3,4,5],[0,1,0]]); # nur Angabe der Elemente
```

$$A0 := \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 0 & 1 & 0 \end{bmatrix}$$

```
> A1 := matrix(3,3,[[1,2,3],                    # mit Angabe der Dimensionen
                   [3,4,5],
                   [0,1,0]]);
```

$$A1 := \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 0 & 1 & 0 \end{bmatrix}$$

• Mit Funktionsdefinition

```
> A2 := x->matrix(3,3,[[0,0,x],[0,x^2$2],[x^3$3]]);
  A2(2);
```

$$A2 := x \rightarrow \text{matrix}(3,3,[[0,\ 0,\ x],\ [0,\ x^2\,\$\,2],\ [x^3\,\$\,3]])$$

$$\begin{bmatrix} 0 & 0 & 2 \\ 0 & 4 & 4 \\ 8 & 8 & 8 \end{bmatrix}$$

• Spezielle Matrizen wie hilbert, toeplitz, sylvester, vandermonde, ...

```
> A3 := matrix(3,6,(i,j)->1/(i+j-1));
  A3 := matrix(3,3,(i,j)->1/(i+j-1)); # Hilbert-Matrix
  A3 := hilbert(3):
  toeplitz([a,b,c]);   # Toeplitz-Matrix
```

$$A3 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \end{bmatrix}$$

$$A3 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ b & a & b \\ c & b & a \end{bmatrix}$$

- Einheitsmatrix

```
> A4 := diag(1,1,1);
  A4 := diag(1$3);
  A4 := diag(seq(1,i=1..3));
  A4 := array(1..3,1..3,identity);  evalm(A4);
  A4 := band([1],3);   # Bandmatrix=Diagonalmatrix, Breite=1
```

$$A4 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A4 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A4 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A4 := \mathrm{array}(identity, 1 .. 3, 1 .. 3, [\ ])$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A4 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Blockdiagonalmatrix

```
> A41 := diag(A1,A2(2));
  A42 := BlockDiagonal(A1,A3,hilbert(2));
```

$$A41 := \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 8 & 8 & 8 \end{bmatrix}$$

$$A42 := \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{3} \end{bmatrix}$$

• Elementweise Eingabe einer Matrix
Achtung: nachfolgend so viel leere Zellen lassen, wie Anzahl der Matrixelemente

```
> A5 := matrix(2,2):
  entermatrix(A5); # Eingabe von "enter element i,j >"
                   # mit ; abschliessen
enter element 1,1 > 1;
enter element 1,2 > 2;
enter element 2,1 > 3;
enter element 2,2 > 4;
```

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

```
> A5 = evalm(A5);
```

$$A5 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

• Elementweise Definition einer Matrix mit Doppelschleife

```
> A6 := matrix(3,3):
  for i to rowdim(A6) do
   for j to coldim(A6) do A6[i,j]:=(i+1)*j; end do;
  end do:

  A6[2,2] := 1: A6[3,3] := 0:
  A6;
  # Anzeige
  evalm(A6);    # eval(A6); weil alle Komponenten belegt sind
  A6:
  % = evalm(%);
```

$$A6$$
$$\begin{bmatrix} 2 & 4 & 6 \\ 3 & 1 & 9 \\ 4 & 8 & 0 \end{bmatrix}$$
$$A6 = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 1 & 9 \\ 4 & 8 & 0 \end{bmatrix}$$

• Zufallsmatrix

```
> A7 := randmatrix(3,3);   # randmatrix(3,3,dense);
                           # Zahlen gemaess random(-99,99)
  rand(); # random number generator, random 12 digit non-negative integer
  dd := rand(1..6): # 1..6
  dd();
  dd := 1+rand(6):  # 1 + 0..5
  dd();
```

$$A7 := \begin{bmatrix} 4 & -59 & 66 \\ -55 & 25 & 9 \\ 40 & 61 & 40 \end{bmatrix}$$
$$549552888716$$
$$1$$
$$5$$

- Matrixelemente aus Gleichungen

```
> Eqns := [x-2*y+3*z=1,2*x+3*y-z=0,x+y+z=2];
  A8 := genmatrix(Eqns,[x,y,z]);
```

$$Eqns := [x - 2y + 3z = 1, 2x + 3y - z = 0, x + y + z = 2]$$
$$A8 := \begin{bmatrix} 1 & -2 & 3 \\ 2 & 3 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Aneinandersetzen/Zusammenfuegen von Matrizen (und Vektoren) bei gleicher Zeilenanzahl

```
> H1 := concat(A1,A2(2));    # augment()
  H2 := concat(A8,A1);
  A9 := matrix(6,6,(i,j)->if i<=3 then H1[i,j]; else H2[i-3,j]; end if);

  H1 := transpose(concat(transpose(A1),transpose(A8)));
  H2 := transpose(concat(transpose(A2(2)),transpose(A1)));
  A91 := concat(H1,H2);                # =A9

  A92 := [[evalm(A1), A2(2)],
          [evalm(A8), evalm(A1)]];    # <>A9
```

$$H1 := \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 2 \\ 3 & 4 & 5 & 0 & 4 & 4 \\ 0 & 1 & 0 & 8 & 8 & 8 \end{bmatrix}$$
$$H2 := \begin{bmatrix} 1 & -2 & 3 & 1 & 2 & 3 \\ 2 & 3 & -1 & 3 & 4 & 5 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$
$$A9 := \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 2 \\ 3 & 4 & 5 & 0 & 4 & 4 \\ 0 & 1 & 0 & 8 & 8 & 8 \\ 1 & -2 & 3 & 1 & 2 & 3 \\ 2 & 3 & -1 & 3 & 4 & 5 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$H1 := \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 0 & 1 & 0 \\ 1 & -2 & 3 \\ 2 & 3 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$H2 := \begin{bmatrix} 0 & 0 & 2 \\ 0 & 4 & 4 \\ 8 & 8 & 8 \\ 1 & 2 & 3 \\ 3 & 4 & 5 \\ 0 & 1 & 0 \end{bmatrix}$$

$$A91 := \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 2 \\ 3 & 4 & 5 & 0 & 4 & 4 \\ 0 & 1 & 0 & 8 & 8 & 8 \\ 1 & -2 & 3 & 1 & 2 & 3 \\ 2 & 3 & -1 & 3 & 4 & 5 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$A92 := \left[\left[\begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 2 \\ 0 & 4 & 4 \\ 8 & 8 & 8 \end{bmatrix}, \begin{bmatrix} 1 & -2 & 3 \\ 2 & 3 & -1 \\ 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 0 & 1 & 0 \end{bmatrix} \right] \right]$$

• Zusammengesetzte Matrix, Matrix mit Blockstruktur

$$A_{10} = A_{10}(14,14), \quad A_{10} = \begin{pmatrix} A & B \\ C & A \end{pmatrix}, \quad A(7,7),\ B(7,7),\ C(7,7)$$

$$A = (a)_{ij}, \quad B = (b)_{ij},\ b_{ij} = a_{8-i,j}$$

$$A = \begin{pmatrix} 5 & 4 & 7 & 5 & 6 & 7 & 5 \\ 4 & 12 & 8 & 7 & 8 & 8 & 6 \\ 7 & 8 & 10 & 9 & 8 & 7 & 7 \\ 5 & 7 & 9 & 11 & 9 & 7 & 5 \\ 6 & 8 & 8 & 9 & 10 & 8 & 9 \\ 7 & 8 & 7 & 7 & 8 & 10 & 10 \\ 5 & 6 & 7 & 5 & 9 & 10 & 10 \end{pmatrix}, \quad C = \begin{pmatrix} \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 5 \\ \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 6 & \frac{1}{15} \\ \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 7 & \frac{1}{15} & \frac{1}{16} \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
> A:=matrix(7,7,[[5,4,7,5,6,7,5],
   [4,12,8,7,8,8,6],
   [7,8,10,9,8,7,7],
   [5,7,9,11,9,7,5],
   [6,8,8,9,10,8,9],
   [7,8,7,7,8,10,10],
   [5,6,7,5,9,10,10]]):
```

```
> B:=matrix(7,7,(i,j)->A[8-i,j]);
```

$$B := \begin{bmatrix} 5 & 6 & 7 & 5 & 9 & 10 & 10 \\ 7 & 8 & 7 & 7 & 8 & 10 & 10 \\ 6 & 8 & 8 & 9 & 10 & 8 & 9 \\ 5 & 7 & 9 & 11 & 9 & 7 & 5 \\ 7 & 8 & 10 & 9 & 8 & 7 & 7 \\ 4 & 12 & 8 & 7 & 8 & 8 & 6 \\ 5 & 4 & 7 & 5 & 6 & 7 & 5 \end{bmatrix}$$

```
> C:=matrix(7,7,[[1/8,1/9,1/10,1/11,1/12,1/13,5],
   [1/9,1/10,1/11,1/12,1/13,6,1/15],
   [1/10,1/11,1/12,1/13,7,1/15,1/16],
   [0,0,0,5,0,0,0],
   [0,0,6,0,0,0,0],
   [0,7,0,0,0,0,0],
   [8,0,0,0,0,0,0]]):
```

$$C := \begin{bmatrix} \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 5 \\ \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 6 & \frac{1}{15} \\ \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 7 & \frac{1}{15} & \frac{1}{16} \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> H1:=[evalm(A),evalm(B)]:   # kein Zusammenfuegen
  H1:=concat(A,B):    # augment
  H2:=concat(C,A):
  A10:=matrix(14,14,(i,j)->if i<8 then H1[i,j] else H2[i-7,j] end if);
```

$$A10 := \begin{bmatrix} 5 & 4 & 7 & 5 & 6 & 7 & 5 & 5 & 6 & 7 & 5 & 9 & 10 & 10 \\ 4 & 12 & 8 & 7 & 8 & 8 & 6 & 7 & 8 & 7 & 7 & 8 & 10 & 10 \\ 7 & 8 & 10 & 9 & 8 & 7 & 7 & 6 & 8 & 8 & 9 & 10 & 8 & 9 \\ 5 & 7 & 9 & 11 & 9 & 7 & 5 & 5 & 7 & 9 & 11 & 9 & 7 & 5 \\ 6 & 8 & 8 & 9 & 10 & 8 & 9 & 7 & 8 & 10 & 9 & 8 & 7 & 7 \\ 7 & 8 & 7 & 7 & 8 & 10 & 10 & 4 & 12 & 8 & 7 & 8 & 8 & 6 \\ 5 & 6 & 7 & 5 & 9 & 10 & 10 & 5 & 4 & 7 & 5 & 6 & 7 & 5 \\ \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 5 & 5 & 4 & 7 & 5 & 6 & 7 & 5 \\ \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 6 & \frac{1}{15} & 4 & 12 & 8 & 7 & 8 & 8 & 6 \\ \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 7 & \frac{1}{15} & \frac{1}{16} & 7 & 8 & 10 & 9 & 8 & 7 & 7 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 5 & 7 & 9 & 11 & 9 & 7 & 5 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 6 & 8 & 8 & 9 & 10 & 8 & 9 \\ 0 & 7 & 0 & 0 & 0 & 0 & 0 & 7 & 8 & 7 & 7 & 8 & 10 & 10 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 6 & 7 & 5 & 9 & 10 & 10 \end{bmatrix}$$

• QR-Faktorisierung mittels Householder-Reflexionen

Wir verwenden die Reflexionsmatrix $S = I - 2\frac{uu^T}{u^Tu}$, um einen Vektor x auf den verlängerten ersten Einheitsvektor zu transformieren. Die Erzeugung von Nullkomponenten ab der zweiten Komponente des Vektors nutzt man bei der schrittweisen Transformation einer Matrix auf die obere Dreiecksform, indem der Reihe nach alle Spalten gespiegelt werden.
Sei $A = (a_1, a_2, ..., a_n)$ mit a_i als Spalten der Matrix.

Transformation eines Vektors x

$$\begin{aligned} u &= x - ce_1, \quad c = -\text{sign1}(x_1)\ \|x\|_2,\ |c| = \|x\|_2, \\ u &= (x_1 - c, x_2, x_3, ..., x_n)^T, \quad u_1 = x_1 - c, \\ ce_1 &= Sx, \quad S = I - 2\frac{uu^T}{u^Tu},\ f = \frac{2}{u^Tu} = \frac{1}{\|x\|_2^2 - cx_1} = -\frac{1}{cu_1}. \end{aligned}$$

Transformation der Matrix A

Schritt $k = 0$
$A^{(0)} = A = (a_1, A'), \quad x = a_1, \quad Q_0 = S_0,$

$$A^{(1)} = Q_0A^{(0)} = S_0(a_1, A') = (S_0a_1, S_0A') = (c_1e_1, S_0A') = \begin{pmatrix} c_1 & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix}.$$

Schritt $k = 1, 2, 3, ..., n-2$

$$A^{(k)} = \begin{pmatrix} c_1 & \dots & * & \dots & * \\ 0 & \ddots & \vdots & & \vdots \\ . & & c_k & \dots & * \\ . & & 0 & & \\ \vdots & & \vdots & T^{(k)} & \\ 0 & \dots & 0 & & \end{pmatrix}.$$

Wir definieren die symmetrische und orthogonale Matrix

$$Q_k = \left(\begin{array}{c|c} I(k,k) & 0 \\ \hline 0 & S_k \end{array}\right)$$

mit der Reflexionsmatrix S_k, welche die Matrix $T^{(k)}$ auf

$$S_kT^{(k)} = \begin{pmatrix} c_{k+1} & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix}$$

transformiert. Ihre j-te Spalte ($j = k+2, k+3, ..., n$) ergibt sich zu

$$(S_k T^{(k)})_j = ((I - f u u^T) T^{(k)})_j = T_j^{(k)} - f(u^T T_j^{(k)})\, u.$$

Damit berechnet man nun

$$A^{(k+1)} \;=\; Q_k A^{(k)} \;=\; \begin{pmatrix} c_1 & \dots & * & * & \dots & * \\ 0 & \ddots & \vdots & \vdots & & \vdots \\ . & & c_k & * & \dots & * \\ . & & 0 & c_{k+1} & \dots & * \\ . & & 0 & 0 & & \\ \vdots & & \vdots & \vdots & T^{(k+1)} & \\ 0 & \dots & 0 & 0 & & \end{pmatrix}.$$

Bildung der Transformationsmatrix Q_k, Schritt $k = 2$

```
> n:=5;
  E:=evalm(array(identity,1..n,1..n));
  k:=2;
  Ak:=matrix(n,n,[[1,2,3,4,5],
                  [0,1,2,3,4 ],
                  [0,0,3,6,9 ],
                  [0,0,4,7,10],
                  [0,0,0,5,15]]);
```

$$n := 5$$

$$E := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$k := 2$$

$$Ak := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 3 & 6 & 9 \\ 0 & 0 & 4 & 7 & 10 \\ 0 & 0 & 0 & 5 & 15 \end{bmatrix}$$

```
> # Schritt k=2: Spiegelung x=(3,4,0)-> c(1,0,0)
  x :=submatrix(Ak,k+1..n,[k+1]);
  nx:=norm(x,2);
  if evalf(nx)>0 then
   c:=-nx;
   if evalf(x[1,1])<0 then c:=nx; end if;
   e1:=submatrix(E,k+1..n,[k+1]);
   u :=evalm(x-c*e1);
   nu:=norm(u,2)^2;
   S:=simplify(evalm(submatrix(E,k+1..n,k+1..n)-2/nu*(u&*transpose(u))));
```

```
 Qh:=evalm(concat(
  transpose(concat(submatrix(E,1..k,1..k),array(sparse,1..k,1..n-k))),
  transpose(concat(array(sparse,1..n-k,1..k),evalm(S)))
  ));

 Akp1:=evalm(Qh&*Ak);
end if;

# nicht
Qh1:=[[submatrix(E,1..k,1..k),    array(sparse,1..k,1..n-k)],
      [array(sparse,1..n-k,1..k), evalm(S)]];
```

$$x := \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}$$

$$nx := 5$$

$$c := -5$$

$$e1 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$u := \begin{bmatrix} 8 \\ 4 \\ 0 \end{bmatrix}$$

$$nu := 80$$

$$S := \begin{bmatrix} \frac{-3}{5} & \frac{-4}{5} & 0 \\ \frac{-4}{5} & \frac{3}{5} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Qh := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{-3}{5} & \frac{-4}{5} & 0 \\ 0 & 0 & \frac{-4}{5} & \frac{3}{5} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Akp1 := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & -5 & \frac{-46}{5} & \frac{-67}{5} \\ 0 & 0 & 0 & \frac{-3}{5} & \frac{-6}{5} \\ 0 & 0 & 0 & 5 & 15 \end{bmatrix}$$

$$Qh1 := \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} \frac{-3}{5} & \frac{-4}{5} & 0 \\ \frac{-4}{5} & \frac{3}{5} & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

• LGS mit Koeffizientenmatrix und rechter Seite

```
> H := matrix(3,3,(i,j)->1/(i+j-1)):   # Hilbert-Matrix
  i := 'i';
  j := 'j';
  b := vector(3,[seq(sum(1/(i+j-1),j=1..3),i=1..3)]);
  x := vector(3);
  A11 := matrix(3,4);
  for i to 3 do for j to 3 do A11[i,j] := H[i,j]; end do; end do:
  for i to 3 do A11[i,4] := b[i]; end do:
  #  A11 := concat(H,b);
  A11:
  % = evalm(%);
```

$$b := \left[\frac{11}{6}, \frac{13}{12}, \frac{47}{60}\right]$$

$$x := \text{array}(1 \,..\, 3, [\ ])$$

$$A11 := \text{array}(1 \,..\, 3,\, 1 \,..\, 4, [\ ])$$

$$A11 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{11}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{13}{12} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{47}{60} \end{bmatrix}$$

• **Bandmatrizen**

Die Matrix hat die folgende Bandstruktur.

$$A(n,n) = (a_{ij}) = \left(\begin{array}{cccccccccc} * & * & * & * & * & . & . & . & . & . \\ * & * & * & * & * & * & . & . & . & . \\ * & * & * & * & * & * & * & . & . & . \\ . & * & * & * & * & * & * & * & . & . \\ . & . & * & * & * & * & * & * & * & . \\ . & . & . & * & * & * & * & * & * & * \\ . & . & . & . & * & * & * & * & * & * \\ . & . & . & . & . & * & * & * & * & * \\ . & . & . & . & . & . & * & * & * & * \\ . & . & . & . & . & . & . & * & * & * \end{array}\right) \begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \end{array}\right\} \beta$$

$$\underbrace{\qquad\qquad}_{\alpha}$$

Das Symbol $*$ charakterisiert den Bereich der Nichtnullelemente (NNE) der Matrix und damit die Nebendiagonalen (Kodiagonalen), wo NNE auftreten.
Die Anzahl der unteren Nebendiagonalen mit NNE ist $\alpha \geq 0$, die der oberen $\beta \geq 0$.

Es gilt

$$\alpha = \begin{cases} 0, & \text{falls } a_{ij} = 0 \text{ für alle } i > j, \\ \max\limits_{\substack{i>j \\ a_{ij}\neq 0}} (i-j), & \text{sonst,} \end{cases}$$

$$\beta = \begin{cases} 0, & \text{falls } a_{ij} = 0 \text{ für alle } j > i, \\ \max\limits_{\substack{j>i \\ a_{ij}\neq 0}} (j-i) = -\min\limits_{\substack{j>i \\ a_{ij}\neq 0}} (i-j), & \text{sonst.} \end{cases}$$

Als Bandbreite bezeichnen wir die Größe

$$bw(A) = \alpha + \beta + 1.$$

Im Allgemeinen ist $\alpha+\beta+1 \ll n$. Ist $\alpha = \beta = 0$, so erhalten wir eine Diagonalmatrix. Ist einer der Werte gleich Null, so haben wir eine obere bzw. untere Dreiecksmatrix. Für $\alpha = \beta$ liegt eine symmetrische Bandstruktur vor, jedoch noch nicht die Symmetrie der NNE. Für $A = A^T$ bezeichnen wir mit $\alpha+1 = \beta+1$ die halbe Bandbreite, in der Literatur jedoch manchmal auch als Bandbreite genannt. Die Bandbreite einer Nullmatrix ist somit 1.

Als erste modifizierte Bandbreite bezeichnen wir die Größe

$$b1(A) = 1 + 2 \max_{\substack{i,\,j\,=\,1(1)n \\ j\neq i}} (|i-j| \text{ mit } a_{ij} \neq 0 \text{ oder } a_{ji} \neq 0).$$

Als zweite modifizierte Bandbreite bezeichnen wir die Größe

$$b2(A) = \min(m, \quad \text{wobei } a_{ij} = 0 \ \forall i,j \text{ mit } |i-j| > m).$$

Die Bandbreite $b2$ ist somit gleich der Anzahl der Nebendiagonalen oberhalb, resp. unterhalb der Hauptdiagonalen, welche NNE enthalten.

Jede voll besetzte Matrix hat die Bandbreiten $bw = b1 = 2n-1,\ b2 = n-1$.
Aber auch viele sparse Matrizen haben diese modifizierten Bandbreiten, z. B. die Matrix

$$\begin{pmatrix} * & & & & * \\ & * & & & \\ & & * & & \\ & & & * & \\ & & & & * \end{pmatrix}$$

mit $bw = n,\ b1 = 2n-1$ und $b2 = n-1$.

- Einige Bandmatrizen

```
> A12 := matrix(3,3,(i,j)->if i=j then 2
         elif abs(i-j)=1 then -1 else 0 end if);  # Tridiagonalmatrix
```

$$A12 := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

```
> A13 := band([1],3);   # Bandmatrix=Diagonalmatrix, Breite=1
```

$$A13 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> A14 := band([1,2,-1],4);            # Bandmatrix, Breite=3
  A141 := band([4,1,2,-1,4],4);       # Breite=5
  A142 := band([5,4,1,2,-1,4,5],4):   # Breite=7, volle Matrix
```

$$A14 := \begin{bmatrix} 2 & -1 & 0 & 0 \\ 1 & 2 & -1 & 0 \\ 0 & 1 & 2 & -1 \\ 0 & 0 & 1 & 2 \end{bmatrix}$$

$$A141 := \begin{bmatrix} 2 & -1 & 4 & 0 \\ 1 & 2 & -1 & 4 \\ 4 & 1 & 2 & -1 \\ 0 & 4 & 1 & 2 \end{bmatrix}$$

- Matrizen mit Bildungsfunktion/Optionen (built-in indexing function)
  identity, sparse, diagonal, symmetric, antisymmetric

```
> A151 := matrix(4,4,[]);
  A152 := diag(1$4);

  # array mit Matrixoptionen
  A15 := array(identity,1..4,1..4);   # array(1..4,1..4,identity);
  evalm(A15);
  A151 := evalm(A15):  # Kontrolle der Zuweisung
  A152 := evalm(A15):

  A16 := array(sparse,1..4,1..4);     # Nullmatrix
  evalm(A16);
  A16[1,1] := 1:
  evalm(A16);          # sparse Speicherstruktur nicht zu erkennen

  A17 := array(diagonal,1..4,1..4);
  A17[1,1] := 2:
  evalm(A17);
```

```
A18 := array(symmetric,1..4,1..4);
A18[1,2] := 2:
evalm(A18);

A19 := array(antisymmetric,1..4,1..4);
A19[1,2] := 2:
evalm(A19);
```

$$A151 := \mathrm{array}(1 \,..\, 4,\, 1 \,..\, 4,\, [\ ])$$

$$A152 := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A15 := \mathrm{array}(identity,\, 1 \,..\, 4,\, 1 \,..\, 4,\, [\ ])$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A16 := \mathrm{array}(sparse,\, 1 \,..\, 4,\, 1 \,..\, 4,\, [\ ])$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A18 := \mathrm{array}(diagonal,\, 1 \,..\, 4,\, 1 \,..\, 4,\, [\ ])$$

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & A17_{2,2} & 0 & 0 \\ 0 & 0 & A17_{3,3} & 0 \\ 0 & 0 & 0 & A17_{4,4} \end{bmatrix}$$

$$A18 := \mathrm{array}(symmetric,\, 1 \,..\, 4,\, 1 \,..\, 4,\, [\ ])$$

$$\begin{bmatrix} A18_{1,1} & 2 & A18_{1,3} & A18_{1,4} \\ 2 & A18_{2,2} & A18_{2,3} & A18_{2,4} \\ A18_{1,3} & A18_{2,3} & A18_{3,3} & A18_{3,4} \\ A18_{1,4} & A18_{2,4} & A18_{3,4} & A18_{4,4} \end{bmatrix}$$

$$A19 := \mathrm{array}(antisymmetric,\, 1 \,..\, 4,\, 1 \,..\, 4,\, [\ ])$$

$$\begin{bmatrix} 0 & 2 & A19_{1,3} & A19_{1,4} \\ -2 & 0 & A19_{2,3} & A19_{2,4} \\ -A19_{1,3} & -A19_{2,3} & 0 & A19_{3,4} \\ -A19_{1,4} & -A19_{2,4} & -A19_{3,4} & 0 \end{bmatrix}$$

(5) Spezielle Matrizen

Diagonalmatrizen mit speziellen Spektren (Verteilung der Diagonalelemente) unter Verwendung eines Shift

- Gleichverteilung

```
> c := 0;      # Shift
  n := 8;      # gerade
  m := n/2;    i := 'i':
  B1 := evalm(diag(seq(-1+2*(i-1)/(n-1),i=1..n))+c*diag(1$n));
  evalf(evalm(B1));    # GPZ mit Digits:=10
```

$$c := 0$$

$$n := 8$$

$$m := 4$$

$$B1 := \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{-5}{7} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-3}{7} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-1}{7} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{7} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{3}{7} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{5}{7} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1., & 0., & 0., & 0., & 0., & 0., & 0., & 0. \\ 0., & -0.7142857143, & 0., & 0., & 0., & 0., & 0., & 0. \\ 0., & 0., & -0.4285714286, & 0., & 0., & 0., & 0., & 0. \\ 0., & 0., & 0., & -0.1428571429, & 0., & 0., & 0., & 0. \\ 0., & 0., & 0., & 0., & 0.1428571429, & 0., & 0., & 0. \\ 0., & 0., & 0., & 0., & 0., & 0.4285714286, & 0., & 0. \\ 0., & 0., & 0., & 0., & 0., & 0., & 0.7142857143, & 0. \\ 0., & 0., & 0., & 0., & 0., & 0., & 0., & 1. \end{bmatrix}$$

- Verteilung, an den Raendern dichter

```
> c := 0:    # Shift
  n := 8:    # gerade
  i := 'i':
  B2 := evalm(diag(seq(evalf(sin(Pi/2*(-1+2*(i-1)/(n-1)))),i=1..n))
          +c*diag(1$n));
```

$$B2 := \begin{bmatrix} -1., & 0., & 0., & 0., & 0., & 0., & 0., & 0. \\ 0., & -0.9009688678, & 0., & 0., & 0., & 0., & 0., & 0. \\ 0., & 0., & -0.6234898020, & 0., & 0., & 0., & 0., & 0. \\ 0., & 0., & 0., & -0.2225209340, & 0., & 0., & 0., & 0. \\ 0., & 0., & 0., & 0., & 0.2225209340, & 0., & 0., & 0. \\ 0., & 0., & 0., & 0., & 0., & 0.6234898020, & 0., & 0. \\ 0., & 0., & 0., & 0., & 0., & 0., & 0.9009688678, & 0. \\ 0., & 0., & 0., & 0., & 0., & 0., & 0., & 1. \end{bmatrix}$$

• Verteilung, nahe den Raendern zwei schmale Streifen

```
> c := 0:      # Shift
  d := 1/10:   # Streifenbreite des Teilspektrums
  n := 8:      # gerade
  m := n/2:
  i := 'i':
  B3 := diag(0$n):
  for i from 1 to m do
    h := -1+d*(i-1)/(m-1);
    B3[i,i] := h+c;     # Spektrum in 2 kleinen Teilintervallen/Streifen
    B3[n+1-i,n+1-i] := -h+c;
  end do:
  B3 := evalm(B3);
```

$$B3 := \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{-29}{30} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-14}{15} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-9}{10} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{9}{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{14}{15} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{29}{30} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

• Tridiagonalmatrix

Als Modellproblem für die Erzeugung einer solchen Matrix wird die Stabdurchbiegung als einfache eindimensionale Zweipunktrandwertaufgabe (RWA) mit inhomogenen Randbedingungen gewählt.
Das zur numerischen Behandlung verwendete Diskretisierungsverfahren führt mit dem 3-Punkte-Differenzenstern auf ein LGS, dessen Eigenschaften kurz dargestellt werden. Des Weiteren wird für die Lösung des LGS ein IV vorgestellt.

- Zweipunktrandwertaufgabe mit inhomogenen Randbedingungen:

$$\begin{aligned} -U''(x) &= F(x), \quad x \in \Omega = (0,1) \subset \mathbb{R}, \\ U &= \varphi \text{ für } x \in \partial\Omega \quad \text{bzw. } U(0) = \varphi_0, \quad U(1) = \varphi_1. \end{aligned}$$

- Gitter: $\overline{\Omega}_h = \{x \mid x = ih,\ i = 0(1)N,\ h = 1/N\}$, h Maschenweite.
- Gitterfunktion: $u_h = (u_1, u_2, ..., u_{N-1})^T$ mit $u_i \approx U_i = U(ih)$.
- Analog für rechte Seite: $f_h = (f_1, f_2, ..., f_{N-1})^T$ mit $f_i = F_i = F(ih)$, d. h. auf dem Gitter wird die rechte Seite exakt dargestellt.
- Approximation der Ableitungen (Operatoren) mittels Differenzenausdrücken:

$$U''(x_i) \approx \frac{1}{h^2}(U_{i+1} - 2U_i + U_{i-1}) \text{ zentraler Differenzenquotient 2. Ordnung.}$$

- Diskretisierte Aufgabe als LGS:

$$-\frac{1}{h^2}(u_{i+1}-2u_i+u_{i-1}) \;=\; f_i, \quad i=1,2,...,N-1,$$
$$u_0 \;=\; \varphi_0, \quad u_N \;=\; \varphi_1.$$

- Matrixschreibweise des LGS:

$$A_h u_h \;=\; b_h \quad \text{bzw.} \quad Au \;=\; b$$

mit

$$A_h \;=\; \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b_h \;=\; \begin{pmatrix} f_1+\varphi_0/h^2 \\ f_2 \\ f_3 \\ \cdots \\ f_{N-2} \\ f_{N-1}+\varphi_1/h^2 \end{pmatrix},$$

$$A \;=\; \begin{pmatrix} 2 & -1 & & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b \;=\; h^2\begin{pmatrix} f_1+\varphi_0/h^2 \\ f_2 \\ f_3 \\ \cdots \\ f_{N-2} \\ f_{N-1}+\varphi_1/h^2 \end{pmatrix},$$

$$A \;=\; \text{tridiag}(-1,2,-1).$$

Die Koeffizientenmatrix $A(n,n)$ ist schwach besetzt. Sie hat etwa $3n$ nichtverschwindende Elemente ($n=N-1$) an Stelle von n^2 Elementen bei voll besetzten Matrizen. Sie ist eine Tridiagonalmatrix, d. h. ihre Bandbreite ist 3. Dazu ist sie symmetrisch und positiv definit (spd).

Das IV für die Lösung von $Au=b$ notieren wir in der Basisversion gemäß

$$u^{(m+1)} = Hu^{(m)}+c = (I-W^{-1}A)u^{(m)}+W^{-1}b = u^{(m)}+W^{-1}r^{(m)},$$

wobei

$u^{(0)}$	Startvektor,
$r^{(m)}=b-Au^{(m)}$	Residuum, manchmal auch $r^{(m)}=Au^{(m)}-b$,
$Au^{(m)}-b$	Defekt,
W	Wichtung, Vorkonditionierungsmatrix,
$W^{-1}(Au^{(m)}-b)$	Korrekturvektor,
$H=I-W^{-1}A$	Iterationsmatrix

bedeuten.

```
> n := 8:
  B4 := band([-1,2,-1],n);   # spd 1D-Laplace-Matrix
```

$$B4 := \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

• Weitere RWA

$T''(x) = -\sin(\pi x),\ 0 < x < 1,\ T(0) = T(1) = 0,\ T_{exakt}(x) = \frac{1}{\pi^2}\sin(\pi x)$

Definition der Komponenten im LGS $Ax = b$

```
> n:=8;
  h:=1/(n+1);
  i:='i':
  xii:=vector(n,[seq(i*h,i=1..n)]);
  Texakt:= evalf(evalm(sin(Pi*xii)/(Pi*Pi)));
  B41 := band([-1,2,-1],n):  # Tridiagonalmatrix
  i := 'i': j := 'j':
  b := h*h*sin(Pi*xii);
  evalm(%);
  evalf(%);

  T := vector(n):
  B41E := matrix(n,n+1):
  B41E := augment(B41,b):   # erweiterte Matrix
  # B41E := concat(B41,b);  # Synonym fuer augment
  B41E:
  % = evalm(%);
```

$$n := 8$$

$$h := \frac{1}{9}$$

$$xii := \left[\frac{1}{9}, \frac{2}{9}, \frac{1}{3}, \frac{4}{9}, \frac{5}{9}, \frac{2}{3}, \frac{7}{9}, \frac{8}{9}\right]$$

$$Texakt := [0.03465388573,\ 0.06512800142,\ 0.08774671895,\ 0.09978188716,\ 0.09978188716,\ 0.08774671895,\ 0.06512800142,\ 0.03465388573]$$

$$b := \frac{1}{81}\sin(\pi\, xii)$$

$$\left[\frac{1}{81}\sin\left(\frac{\pi}{9}\right), \frac{1}{81}\sin\left(\frac{2\pi}{9}\right), \frac{\sqrt{3}}{162}, \frac{1}{81}\sin\left(\frac{4\pi}{9}\right), \frac{1}{81}\sin\left(\frac{4\pi}{9}\right), \frac{\sqrt{3}}{162}, \frac{1}{81}\sin\left(\frac{2\pi}{9}\right), \frac{1}{81}\sin\left(\frac{\pi}{9}\right)\right]$$

$$[\ 0.004222470904,\ 0.007935649501,\ 0.01069167165,\ 0.01215812041,\ 0.01215812041,\ 0.01069167165,\ 0.007935649501,\ 0.004222470904]$$

$$B41E := \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{81}\sin(\frac{\pi}{9}) \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{81}\sin(\frac{2\pi}{9}) \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & \frac{\sqrt{3}}{162} \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & \frac{1}{81}\sin(\frac{4\pi}{9}) \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & \frac{1}{81}\sin(\frac{4\pi}{9}) \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & \frac{\sqrt{3}}{162} \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & \frac{1}{81}\sin(\frac{3\pi}{10}) \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & \frac{1}{81}\sin(\frac{2\pi}{9}) \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & \frac{1}{81}\sin(\frac{\pi}{9}) \end{bmatrix}$$

• Blocktridiagonalmatrizen

Betrachten wir den Temperaturverlauf in einer dünnen quadratischen Platte gegeben durch die partielle Differentialgleichung für eine Funktion $U(x, y)$ auf dem Einheitsquadrat.

$$-\Delta U(x,y) = -\left(\frac{\partial U}{\partial x^2} + \frac{\partial U}{\partial y^2}\right) = Q(x,y), \quad (x,y) \in \Omega = (0,1)^2.$$

Auf dem Rand des Gebietes sei $U(x, y)$ gleich Null.
Das ist eine elliptische Randwertaufgabe bzw. die Poisson-Gleichung.
Der Diskretisierungsparameter bzw. die Maschenweite des quadratischen Gitters sei $h = 1/(N + 1)$. Man diskretisiert die partiellen Ableitungen mittels zentraler Differenzenquotienten 2. Ordnung

$$\Delta U(x_i, y_j) \approx \frac{1}{h^2}(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{ij})$$

und notiert die Differenzenformel (5-Punkte-Differenzenstern) für alle inneren (zweidimensionalen) Knoten (x_i, y_j), $i, j = 1, 2, ..., N$, in linearer Reihenfolge zeilenweise gemäß $(j - 1)N + i$.
Die diskretisierte RWA schreibt man als LGS $Au = h^2 q$.
Welche Struktur und Eigenschaften hat die Matrix A? Wie groß ist ihre Bandbreite?
A besitzt die folgende Blockstruktur.

$$A = \begin{pmatrix} B & -I & & & \\ -I & B & -I & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & B \end{pmatrix}$$

mit der $(N \times N)$-Matrix

$$B = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 4 \end{pmatrix}$$

und der $(N \times N)$-Einheitsmatrix I. A ist eine dünn besetzte symmetrische Matrix mit Bandstruktur. Die Bandbreite beträgt $2N + 1$. Die Matrix ist irreduzibel diagonaldominant.

```
> n1 := 4:   # n1=N
  n1q := n1^2:
  T := band([-1,4,-1],n1);
  mI := diag(-1$n1);
  B5 := band([mI,T,mI],n1):
  evalm(B5);
  B5 := band([evalm(mI),evalm(T),evalm(mI)],n1);   # Error
```

$$T := \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{bmatrix}$$

$$mI := \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} T & mI & 0 & 0 \\ mI & T & mI & 0 \\ 0 & mI & T & mI \\ 0 & 0 & mI & T \end{bmatrix}$$

```
Error, (in evalm) unnamed vector or array with undefined entries.
```

Korrekt: Blocktridiagonalmatrizen vom 2D-Laplace-Operator auf quadratischem Gitter

Variante 1

```
> n1 := 4:
  n1q := n1^2:
  T := band([-1,4,-1],n1):   # Diagonalblock
  B6 := diag(T$n1):          # Blockdiagonalmatrix
  for i from 1 to n1q-n1 do  # Blocknebendiagonalen ergaenzen
   B6[i,i+n1] := -1;
   B6[i+n1,i] := -1;
  end do:
  B6 := evalm(B6);           # Blocktridiagonalmatrix
```

$$B6 := \begin{bmatrix} 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 \end{bmatrix}$$

Variante 2

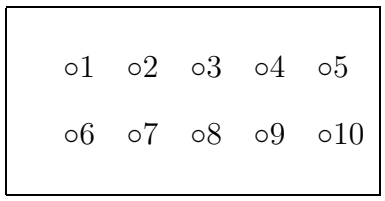
```
> n1 := 4:
  n1q := n1^2:
  B7 := band([-1,0$(n1-2),-1,4,-1,0$(n1-2),-1],n1q); # 5 Diagonalen belegt
  # evtl. noch einige -1 zu Null machen, z.B. B7[4,5], B7[5,4],...
  B71:=evalm(B7):
  for i to n1-1 do
    ii := i*n1;
    B71[ii,ii+1] := 0;
    B71[ii+1,ii] := 0;
  end do:
  B71 := evalm(B71):  # wie B6
```

$$B7 := \begin{bmatrix} 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 \end{bmatrix}$$

• Gegeben sei die partielle Differentialgleichung (Poisson-Gleichung) für eine Funktion $U(x, y)$ auf einem Rechteckgebiet.

$$-\Delta U(x,y) = -\left(\frac{\partial U}{\partial x^2} + \frac{\partial U}{\partial y^2}\right) = Q(x,y), \quad (x,y) \in \Omega = (a,b) \times (c,d).$$

Auf dem Rand des Gebietes sei die Funktion $U(x, y)$ vorgegeben (Dirichletsche Randbedingungen).
Zur Lösung verwenden wir die finite Differenzenmethode auf einem (2×5)-Gitter (x_i, y_j) mit der Maschenweite h. Wir führen die Nummerierung der Gitterpunkte im rechteckigen Gebiet zeilenweise durch.



So erhalten wir eine Matrix mit der Blockstruktur

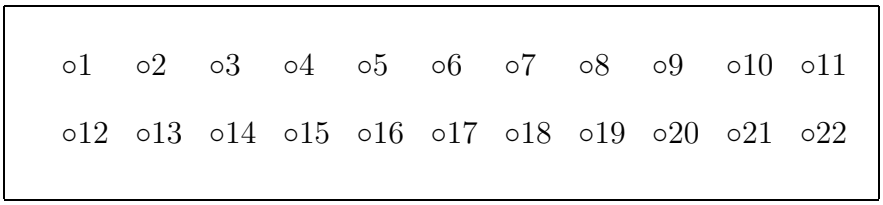
$$A = \begin{pmatrix} B & -I \\ -I & B \end{pmatrix}$$

Blocktridiagonalmatrizen vom 2D-Laplace-Operator auf Rechteckgitter 2×5, 5-Punkte-Differenzenstern

```
> m := 2:
> n := 5:
  mn := m*n:
  B8 := band([-1,0$(n-2),-1,4,-1,0$(n-2),-1],mn);
  # evtl. noch einige -1 zu Null machen, z.B. B8[5,6], B8[6,5],...
  B81 := evalm(B8):
  for i to m-1 do
    ii := i*n;
    B81[ii,ii+1] := 0;
    B81[ii+1,ii] := 0;
  end do:
  B81 := evalm(B81);
```

$$B81 := \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 \end{bmatrix}$$

Analog überlegt man sich die Lösung der Poisson-Gleichung mit der finiten Differenzenmethode auf einem (2×11)-Gitter (x_i, y_j) mit der Maschenweite h. Wir führen die Nummerierung der Gitterpunkte im rechteckigen Gebiet wiederrum zeilenweise durch.

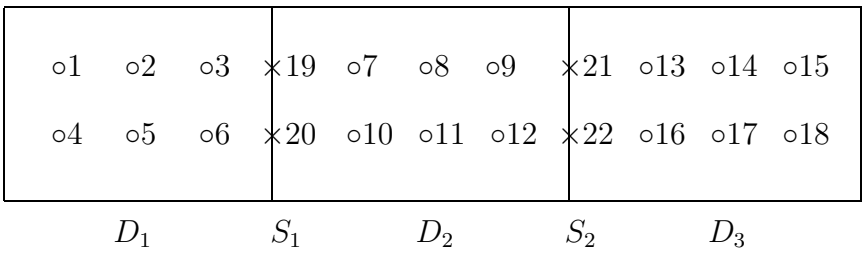


So erhalten wir eine Matrix mit der Blockstruktur

$$A = \begin{pmatrix} B & -I \\ -I & B \end{pmatrix}$$

mit (11×11)-Matrizen B und I. Die Matrix A hat die Bandbreite 23.

Jetzt nehmen wir eine Gebietszerlegung vor. Folgende Zerlegung von Ω in Gebiete D_i und "innere Ränder" S_i sei gegeben.



Man diskretisiert die partiellen Ableitungen wiederum mittels zentraler Differenzenquotienten und notiert die Differenzenformel für alle inneren 22 Knoten $(x_i, y_j),\ i = 1, 2, ..., 12,\ j = 1, 2$, in der Reihenfolge 1,2,...,18,19,...,22.
Hierbei beschreiben die inneren Teilgebiete S_i Punkte, welche bei der Diskretisierung die beiden benachbarten breiten Teilgebiete beeinflussen. Sie werden in der Nummerierung als letzte berücksichtigt.

Dies führt auf die folgende Matrixstruktur

$$A = \begin{pmatrix} B & & & C_1 \\ & B & & C_2 \\ & & B & C_3 \\ C_1^T & C_2^T & C_3^T & D \end{pmatrix}$$

mit der (6×6)-Matrix B den (6×4)-Matrizen C_i und der (4×4)-Matrix D.

• Van der Vorst Matrix mit Shift

```
> c := 0:    # Shift
  n := 100:
  i := 'i':
  C1 := evalm(diag(seq(-11+2*i,i=1..n))-c*diag(1$n)):
  seq(C1[i,i],i=1..n);
```

-9, -7, -5, -3, -1, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129, 131, 133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153, 155, 157, 159, 161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181, 183, 185, 187, 189

```
> c := 0.97:   # Shift
  n := 100:
  i := 'i':
  C2 := evalm(diag(seq(-11+2*i,i=1..n))-c*diag(1$n)):
  seq(C2[i,i],i=1..n);
```

-9.97, -7.97, -5.97, -3.97, -1.97, 0.03, 2.03, 4.03, 6.03, 8.03, 10.03, 12.03, 14.03, 16.03, 18.03, 20.03, 22.03, 24.03, 26.03, 28.03, 30.03, 32.03, 34.03, 36.03, 38.03, 40.03, 42.03, 44.03, 46.03, 48.03, 50.03, 52.03, 54.03, 56.03, 58.03, 60.03, 62.03, 64.03, 66.03, 68.03, 70.03, 72.03, 74.03, 76.03, 78.03, 80.03, 82.03, 84.03, 86.03, 88.03, 90.03, 92.03, 94.03, 96.03, 98.03, 100.03, 102.03, 104.03, 106.03, 108.03, 110.03, 112.03, 114.03, 116.03, 118.03, 120.03, 122.03, 124.03, 126.03, 128.03, 130.03, 132.03, 134.03, 136.03, 138.03, 140.03, 142.03, 144.03, 146.03, 148.03, 150.03, 152.03, 154.03, 156.03, 158.03, 160.03, 162.03, 164.03, 166.03, 168.03, 170.03, 172.03, 174.03, 176.03, 178.03, 180.03, 182.03, 184.03, 186.03, 188.03

Diese Matrizen sind Testmatrizen für die Verfahren der konjugierten Gradienten und konjugierten Residuen.

(6) Matrixzerlegung $A = D - E - F = D(I - L - U)$

in ihre Diagonal- und Dreiecksanteile.
The Matrix(..) function is the constructor for the Matrix data structure.
It is one of the principal data structures on which the **LinearAlgebra** routines operate.

```
> with(LinearAlgebra):
```

```
> n:=3:
  A:=matrix(n,n,[[12,-2,3],[-1,8,-2],[-1,3,12]]);
```

$$A := \begin{bmatrix} 12 & -2 & 3 \\ -1 & 8 & -2 \\ -1 & 3 & 12 \end{bmatrix}$$

```
> evalm(A);
  DD:=diag(seq(A[i,i],i=1..n));
           # DD:=matrix(n,n,(i,j)->if i=j then A[i,j] else 0 end if);
  L1:=Matrix(n,[A],shape=triangular[lower]);
  U1:=Matrix(n,[A],shape=triangular[upper]);
  E:=evalm(U1-A);
           # E:=matrix(n,n,(i,j)->if i>j then -A[i,j] else 0 end if);
  F:=evalm(L1-A);
           # F:=matrix(n,n,(i,j)->if i<j then -A[i,j] else 0 end if);
  evalm(DD-E-F);            # = A
```

$$\begin{bmatrix} 12 & -2 & 3 \\ -1 & 8 & -2 \\ -1 & 3 & 12 \end{bmatrix}$$

$$DD := \begin{bmatrix} 12 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 12 \end{bmatrix}$$

$$L1 := \begin{bmatrix} 12 & 0 & 0 \\ -1 & 8 & 0 \\ -1 & 3 & 12 \end{bmatrix}$$

$$U1 := \begin{bmatrix} 12 & -2 & 3 \\ 0 & 8 & -2 \\ 0 & 0 & 12 \end{bmatrix}$$

$$E := \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ -1 & 3 & 0 \end{bmatrix}$$

$$F := \begin{bmatrix} 0 & -2 & 3 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 12 & -2 & 3 \\ -1 & 8 & -2 \\ -1 & 3 & 12 \end{bmatrix}$$

```
  L:=evalm(inverse(DD)&*E);
  U:=evalm(inverse(DD)&*F);
  II:=evalm(array(identity,1..n,1..n));
  evalm(DD&*(II-L-U));      # = A
```

$$L := \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{8} & 0 & 0 \\ \frac{1}{12} & \frac{-1}{4} & 0 \end{bmatrix}$$

$$U := \begin{bmatrix} 0 & \frac{1}{6} & \frac{-1}{4} \\ 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 \end{bmatrix}$$

$$II := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 12 & -2 & 3 \\ -1 & 8 & -2 \\ -1 & 3 & 12 \end{bmatrix}$$

2.4 Felder und das Kommando `evalm`

Man beachte, dass der Wert eines Feldes A der Bezeichner des Felds A selbst ist, weil auch oft die Feldkomponenten noch keine konkreten Werte haben. Es wird also mit Adressen gearbeitet und es werden Adressen übergeben.
Es gibt spezielle Kommandos zur Berechnung/Verarbeitung von Feldern, wo es um ihren Inhalt, also um die Feldkomponenten geht. Wenn man Zuweisungen, Ausgaben oder Prozedurrückgabe von Feldern machen will, dann benutzt man das Kommando `print` oder eine der Berechnungsfunktionen `eval, evalm, evalf, ....` Dabei wird unterschieden, ob das Argument ein Array oder eine Matrix/Vektor ist.

Rechnungen in Maple (Datei: *evalm_t1.mws*)

Felder

Behandlung von Feldern als symbolische Groessen/Namen, Uebergabe von Adressen

```
> A := array(0..3,[2,4,6,8]): A;
  B := A:  # B ist identisch mit A
  B;
```

$$A$$
$$A$$

Auswertung von Feldern mit Berechnungsfunktionen

```
> eval(A);
  evalm(A), evalf(A), evalf(evalm(A));
```

array(0 .. 3, [
(0)=2
(1)=4
(2)=6
(3)=8
])

$$A,\ A,\ A$$

```
> eval(B);
  evalm(B), evalf(B), evalf(evalm(B));
```

array(0 .. 3, [
(0)=2
(1)=4
(2)=6
(3)=8
])

$$A,\ A,\ A$$

```
> # array mit unteren Indexgrenzen = 1 wie  matrix bzw. vector
  A := array(1..3,[2,4,6]): A;
  B := A: B;
```

$$A$$
$$A$$

```
> evalm(A), eval(A);
  evalf(evalm(A)), evalf(A);
```

$$[2,4,6], [2,4,6]$$
$$[2.,4.,6.], A$$

```
> evalm(B), eval(B);
  evalf(evalm(B)), evalf(B);
```

$$[2,4,6], [2,4,6]$$
$$[2.,4.,6.], A$$

Will man also Matrizen und Vektoren umspeichern und unter anderen Namen verwenden, dann benutzt man das Kommando `evalm`.

Prozeduren

Test von Feldvariablen in einer Prozedur
Variante 1: korrekt, ueberall `evalm( )` verwenden!

```
> TT1:=proc(n::posint,b::vector)
   local p,r;

   r:=evalm(b+b):
   p:=evalm(r);                  # an Stelle von p:=r;
   printf(‘r\n‘); print(r);
   printf(‘p\n‘); print(p);

   r:=evalm(2*r);
   printf(‘r\n‘); print(r);

   printf(‘Kontrolle\n‘);
   printf(‘p ohne Aenderung <>r !!!\n‘); print(p);
   p:=evalm(r+100*p);
   printf(‘p\n‘); print(p);
   p;
  end:

> n:=3;  b:=vector(n,[1,2,3]);
```

$$n := 3$$
$$b := [1, 2, 3]$$

Ergebnisse TT1

```
> erg:=TT1(n,b);
  evalm(erg);
```

r

$$[2,\ 4,\ 6]$$

p

$$[2,\ 4,\ 6]$$

r

$$[4,\ 8,\ 12]$$

Kontrolle
p ohne Aenderung <>r !!!

$$[2,\ 4,\ 6]$$

p

$$[204,\ 408,\ 612]$$

$$erg := p$$

$$[204,\ 408,\ 612]$$

Variante 2: fehlerhaft, weil einmal `evalm( )` nicht steht

```
> TT2:=proc(n::posint,b::vector)
   local p,r;

   r:=evalm(b+b):
   p:=r;                      # Fehlerquelle
                              # an Stelle von  p:=evalm(r);
   printf(‘r\n‘); print(r);
   printf(‘p\n‘); print(p);

   r:=evalm(2*r);
   printf(‘r\n‘); print(r);

   printf(‘Kontrolle\n‘);
   printf(‘p=r -> Fehler !!!\n‘); print(p);
   p:=evalm(r+100*p);
   printf(‘p\n‘); print(p);
   p;
  end:

> n:=3;  b:=vector(n,[1, 2, 3]);
```

$$n := 3$$

$$b := [1, 2, 3]$$

Ergebnisse TT2

```
> erg:=TT2(n,b);
  evalm(erg);
```

r

$$[2, 4, 6]$$

p

$$[2, 4, 6]$$

r

$$[4, 8, 12]$$

Kontrolle
p=r –> Fehler !!!

$$[4, 8, 12]$$

p

$$[404, 808, 1212]$$

$$erg := p$$

$$[404, 808, 1212]$$

Variante 3: fehlerhaft durch Umspeicherung ohne `evalm( )`

```
> TT3:=proc(n::posint,b::vector)
   local p,r,ph;

   r:=evalm(b+b):
   p:=evalm(r);                  # an Stelle von  p:=r;
   ph:=p;
   printf(‘r\n‘); print(r);
   printf(‘p\n‘); print(p);
   printf(‘ph\n‘); print(ph);

   r:=evalm(2*r);
   printf(‘r\n‘); print(r);

   printf(‘Kontrolle\n‘);
   printf(‘p <>r !!!\n‘); print(p);
   printf(‘ph<>r !!!\n‘); print(ph);
   p:=ph;                        # Fehlerquelle
                                 # korrekt waere  p:=evalm(ph);
   printf(‘ph  wird zum Symbol/Namen !!!\n‘); print(ph);
   printf(‘p   wird zum Symbol/Namen !!!\n‘); print(p);

   p:=evalm(r+100*p);  # enthaelt nun Namen p
   printf(‘p\n‘); print(p);
   p;
  end:
```

```
> n:=3;
  b:=vector(n,[1, 2, 3]);
```

$$n := 3$$
$$b := [1, 2, 3]$$

Ergebnisse TT3

```
> erg:=TT3(n,b);
  evalm(erg);
```

r

$$[2,\ 4,\ 6]$$

p

$$[2,\ 4,\ 6]$$

ph

$$[2,\ 4,\ 6]$$

r

$$[4,\ 8,\ 12]$$

Kontrolle
p <>r !!!

$$[2,\ 4,\ 6]$$

Kontrolle
ph <>r !!!

$$[2,\ 4,\ 6]$$

ph wird zum Symbol/Namen !!!

$$p$$

p wird zum Symbol/Namen !!!

$$p$$

p

$$[4+100\,p,\ 8+100\,p,\ 12+100\,p]$$
$$erg := p$$
$$[4+100\,p,\ 8+100\,p,\ 12+100\,p]$$

2.5 Produkte, Differenzen und Potenzen mit Matrizen und Vektoren

In der linearen Algebra tritt die Verwendung von Feldern in vielfältigen Situationen auf. Sind Vektoren mit einbezogen, so geht man in der Regel davon aus, dass es sich um **Spaltenvektoren** handelt.
Wir betrachten reelle Vektoren $x, y, z, ... \in \mathbb{R}^n$ und reelle Matrizen $A, B, C, ... \in \mathbb{R}^{n,n}$ bzw. $\mathbb{R}^{m,n}$ (Rechteckmatrizen).

Eigenschaften und Definitionen mit Feldern

- Transposition als $x^T,\ A^T$.
- Inverse einer quadratischen regulären Matrix als A^{-1}.
- Multiplikation von Vektoren oder Matrizen mit skalaren Größen.
- Skalarprodukt oder inneres Produkt von Vektoren aus $\mathbb{R}^n$

  $$(x, y) \ = \ x^T y \ = \ \sum_{i=1}^{n} x_i y_i,$$

  ihre Orthogonalität $x \perp y$ bedeutet $(x, y) = 0$.
- Norm eines Vektors als $\|x\|$ bzw. im Zusammenhang mit dem Skalarprodukt die euklidische Norm $\|x\|_2 = \sqrt{(x, x)}$.
- Norm und Kondition einer Matrix als $\|A\|$ bzw. $\text{cond}(A) = \|A\|\|A^{-1}\|$.
- Dyade oder dyadisches Produkt von Vektoren $x \in \mathbb{R}^n,\ y \in \mathbb{R}^m$

  $$xy^T \ = \ \begin{pmatrix} x_1y_1 & x_1y_2 & x_1y_3 & \cdots & x_1y_{m-1} & x_1y_m \\ x_2y_1 & x_2y_2 & x_2y_3 & \cdots & x_2y_{m-1} & x_2y_m \\ x_3y_1 & x_3y_2 & x_3y_3 & \cdots & x_3y_{m-1} & x_3y_m \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n-1}y_1 & x_{n-1}y_2 & x_{n-1}y_3 & \cdots & x_{n-1}y_{m-1} & x_{n-1}y_m \\ x_ny_1 & x_ny_2 & x_ny_3 & \cdots & x_ny_{m-1} & x_ny_m \end{pmatrix}.$$

- Produkte aus n-dimensionalen Matrizen und Vektoren als

  $$y \ = \ Ax, \quad y_i \ = \ \sum_{j=1}^{n} a_{ij} x_i, \ i = 1, 2, ..., n,$$

  $$y^T \ = \ x^T A, \quad y_j \ = \ \sum_{i=1}^{n} x_i a_{ij}, \ j = 1, 2, ..., n.$$

- Produkte und Potenzen von Matrizen gemäß AB, ABC, $A^0 = I$, A^k u. a.
- Orthogonalität einer Matrix gemäß $AA^T = A^TA = I$ mit I als Einheitsmatrix.
- Quadratische Formen mit n-dimensionalen Matrizen und Vektoren als

$$\begin{aligned} x^TAy &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_iy_j, \\ (Ax)^TAy &= x^TA^TAy = \sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ki}a_{kj}x_iy_j. \end{aligned}$$

Man achtet also konsequent auf die Notation der Vektoren als Spaltenvektoren sowie die Verwendung von einspaltigen oder einzeiligen Rechteckmatrizen.

Maple will sich in der Benutzung von Vektoren als Spalten- oder Zeilenvektoren nicht festlegen, sondern sich nach Bedarf und Stellung eines Vektors in der Formel entscheiden. Das hat zur Folge, dass eigentlich Verwirrungen vorprogrammiert sind. Wir fassen wichtige Erkenntnisse von unterschiedlichen Implementationen in Maple zusammen.

1. So wird das Kommando des Vektortranspostion `transpose(x)` nicht explizit ausgeführt.

2. Im Matrix-Vektor-Produkt `A&*x` ist x ein Spaltenvektor, während x in `x&*A` als Zeilenvektor genommen wird. Stehen jedoch nur Vektoren in der Formel, wie beim Skalarprodukt `transpose(x)&*x` oder dyadischen Produkt `x&*transpose(x)`, dann wird x als Spaltenvektor interpretiert. Also ist die Deutung des links stehenden Vektors x in den Produkten `x&*A` und `x&*transpose(x)` verschieden.

3. Eine Mischung der Situationen von Skalar- und Matrix-Vektor-Produkt ergibt sich bei der Betrachtung der quadratischen Form x^TAx.
   Wegen `x&*A` und `A&*x` ist einerseits die Darstellung `x&*A&*x` möglich.
   Aber genauso kann man wegen `A&*x` und `transpose(x)&*x` den Ausdruck `transpose(x)&*A&*x` notieren.

4. Sobald in größeren Formeln Ausdrücke mit Feldern stehen, z. B. Differenzen von Vektoren oder Matrizen, ist zumeist die Einbeziehung der Berechnungsfunktion `evalm` zu empfehlen, manchmal ist es zur Vermeidung von syntaktischen Fehlern sogar notwendig.

5. Bei Differenzen von Matrizen haben wir die Situation, dass `A-A` nicht die Nullmatrix ist, sondern einfach der Wert 0, so dass dann das Ergebnis auch keine Dimensionsabfragen mit `rowdim( )` bzw. `coldim( )` erlaubt.
   Ist A jedoch inhaltlich mit der Einheitsmatrix I identisch, so ist die Differenz `A-I` eine Nullmatrix mit gegebener Dimension.

6. Bei Potenzen von Matrizen haben wir die Situation, dass `A^0` nicht die Einheitsmatrix ist, sondern einfach der Wert 1, so dass dann das Ergebnis auch keine Dimensionsabfragen mit `rowdim( )` bzw. `coldim( )` erlaubt.
   Bildet man jedoch `A^0-I` mit der Einheitsmatrix I, so erinnert sich Maple daran, dass A^0 eigentlich mehr als die Eins darstellt und berechnet richtig als Differenz die Nullmatrix mit gegebener Dimension.

Zur Kontrolle von Ergebnissen werden auch Feldkomponenten ausgewertet.

Rechnungen in Maple (Datei: *matvek1.mws*)

Definition von Matrizen und Vektoren

```
> m:=5:
  n:=2:
  A:=matrix(m,m,(i,j)->i+j-1);
  A51:=matrix(m,1,[[ 1],[ 2],[ 3],[ 4],[ 5]]);
  A52:=matrix(m,n,[[ 1, 3],
                   [ 2, 2],
                   [ 3, 1],
                   [ 4, 0],
                   [ 5,-1]]);

  x:=vector(m,[1,1,1,1,1]);  # je nach Bedarf Spalten- oder Zeilenvektor
                             # Vektor an Kommas erkennbar
  b:=vector(m,[1,2,3,4,5]);
  c:=vector(m,[1,0,0,0,0]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

$$A51 := \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

$$A52 := \begin{bmatrix} 1 & 3 \\ 2 & 2 \\ 3 & 1 \\ 4 & 0 \\ 5 & -1 \end{bmatrix}$$

$$x := [1, 1, 1, 1, 1]$$

$$b := [1, 2, 3, 4, 5]$$

$$c := [1, 0, 0, 0, 0]$$

Transposition

```
> transpose(A51);
  transpose(A52);

  transpose(x);       # keine explizite Angabe des Vektors,
                      # da nicht klar ob Zeilen- oder Spaltenvektor
  evalm(transpose(x));
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 0 & -1 \end{bmatrix}$$
$$\mathrm{transpose}(x)$$
$$\mathrm{transpose}(x)$$

Produkte

Matrix∗Matrix, Matrix∗Vektor, Vektor∗Matrix, ...
Kontrolle von Feldkomponenten

```
> evalm(transpose(A52)&*A52);    # A52(1..5,1..2)
  evalm(A52&*transpose(A52));
  d:=evalm(transpose(A52)&*b);   # b(1..5) Spaltenvektor
  d[1], d[2];

  f:=evalm(b&*A52);              # b(1..5) Zeilenvektor
  f[1], f[2];

  B:=matrix(2,1,[[2],
                 [3]]);
  B[1,1];
  B[2,1];
  B[1,2];    # Fehler

  B:=evalm(transpose(A52)&*b);   # A52(1..5,1..2), b(1..5) Spaltenvektor
  B[1], B[2];
  B[1,1];    # Fehler
  B[2,1];    # Fehler
  B[1,2];    # Fehler

  C:=evalm(b&*A52);              # b(1..5) Zeilenvektor, A52(1..5,1..2)
  C[1], C[2];
  C[1,1];    # Fehler
```

$$\begin{bmatrix} 55 & 5 \\ 5 & 15 \end{bmatrix}$$
$$\begin{bmatrix} 10 & 8 & 6 & 4 & 2 \\ 8 & 8 & 8 & 8 & 8 \\ 6 & 8 & 10 & 12 & 14 \\ 4 & 8 & 12 & 16 & 20 \\ 2 & 8 & 14 & 20 & 26 \end{bmatrix}$$

$$d := [55, 5]$$

$$55,\ 5$$

$$f := [55, 5]$$

$$55,\ 5$$

$$B := \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$2$$

$$3$$

```
Error, 2nd index, 2, larger than upper array bound 1
```

$$B := [55, 5]$$

$$55,\ 5$$

```
Error, array defined with 1 indices, used with 2 indices
Error, array defined with 1 indices, used with 2 indices
Error, array defined with 1 indices, used with 2 indices
```

$$C := [55, 5]$$

$$55,\ 5$$

```
Error, array defined with 1 indices, used with 2 indices
```

... mit Rechteckmatrizen

```
> R:=evalm(transpose(A52)&*A51);   # A52(1..5,1..2), A51(1..5,1..1)
  R[1,1], R[2,1];
  R[1];

  S:=evalm(transpose(A51)&*A52);
  S[1,1], S[1,2];
  S[1];
```

$$R := \begin{bmatrix} 55 \\ 5 \end{bmatrix}$$

$$55,\ 5$$

```
Error, array defined with 2 indices, used with 1 indices
```

$$S := [55\ 5]$$

$$55,\ 5$$

```
Error, array defined with 2 indices, used with 1 indices
```

Quadratische Form $(Ax)^T Ax$

```
> qfd:=evalm(transpose(evalm(A&*b))&*A&*b);
  evalm(transpose(A&*b)&*A&*b);          # Fehler
  evalm(evalm(transpose(A&*b))&*A&*b);   # Fehler
```

$$qfd := 38375$$

```
Error, (in evalm/amperstar) &* is reserved for matrix multiplication
Error, (in evalm/amperstar) &* is reserved for matrix multiplication
```

Quadratische Form $x^T Ax$

```
> qf:=evalm(b&*A&*b);                 # A(1..5,1..5)
  qf:=evalm(transpose(b)&*A&*b);
  evalm(transpose(b)&*A&*(b-c));

  evalm(transpose(b-c)&*A&*b);        # Fehler
  evalm(transpose(b-c)&*A&*(b-c));    # Fehler, Differenz vorher berechnen

  bmc:=evalm(b-c):
  evalm(transpose(evalm(b-c))&*A&*evalm(b-c));
  evalm(transpose(bmc)&*A&*bmc);
```

$$qf := 1425$$
$$qf := 1425$$
$$1370$$

```
Error, (in linalg[multiply]) expecting a matrix or a vector
Error, (in linalg[multiply]) expecting a matrix or a vector
```

$$1316$$
$$1316$$

Skalarprodukt $x^T x$

```
> evalm(b);
  evalm(c);
  skal:=evalm(transpose(b)&*b);
  evalm(transpose(b-c)&*(b-c));       # Fehler, Differenz vorher berechnen

  bmc:=evalm(b-c);
  evalm(transpose(evalm(b-c))&*evalm(b-c));
  evalm(transpose(bmc)&*bmc);
```

$$[1,\ 2,\ 3,\ 4,\ 5]$$
$$[1,\ 0,\ 0,\ 0,\ 0]$$
$$skal := 55$$

```
Error, (in linalg[multiply]) expecting a matrix or a vector
```

$$bmc := [0, 2, 3, 4, 5]$$
$$54$$
$$54$$

Skalarprodukt (Normquadrat) $z^T z$ und dyadisches Produkt zz^T
Vektor z je nach Bedarf Spalten- oder Zeilenvektor

```
> z:=vector(m,[1$m]);
  transpose(z);
  z1:=matrix(5,1,z);
  transpose(z1);

  # Skalarprodukt
  norm2q:=evalm(transpose(z)&*z); # Normquadrat, z wie Spaltenvektor
  eval(transpose(z)&*z);          # nicht verwertbar
  evalf(transpose(z)&*z);         # nicht verwertbar

  # dyadisches Produkt
  dyad:=evalm(z1&*transpose(z1)); # einsichtige Darstellung mit Matrizen
  dyad:=evalm(z&*transpose(z)):   # auch moeglich, z wie Spaltenvektor
  dyad[1,1], dyad[5,5];

  erg1:=evalm(z&*z1);             # (1)-Vektor, z wie Zeilenvektor
  erg1[1];
  evalm(z&*transpose(z1));        # Fehler, Zeilenvektor*Zeilenvektor
  erg2:=evalm(transpose(z)&*z1);  # ? nicht verwertbar
  erg2[1];
```

$$z := [1, 1, 1, 1, 1]$$

$$\text{transpose}(z)$$

$$z1 := \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$[1 \quad 1 \quad 1 \quad 1 \quad 1]$$

$$norm2q := 5$$

$$\text{transpose}(z)\&^* z$$

$$\text{transpose}(z)\&^* z$$

$$dyad := \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$1, 1$$

$$erg1 := [5]$$

$$5$$

```
Error, (in linalg[multiply]) non matching dimensions for vector/matrix product
```

$$erg2 := \text{transpose}([5])$$

$$\text{transpose}([5])_1$$

Weitere Produkte

```
> evalm(z);                          # z(1..5)
  evalm(z1);                         # z1(1..5,1..1)

  ma11:=evalm(transpose(z1)&*z);     # (1)-Vektor, z wie Spaltenvektor
  ma11[1];
  ma12:=evalm(transpose(z1)&*z1);    # (1 x 1)-Matrix
  ma12[1,1];
  ma13:=evalm(z&*z1);                # (1)-Vektor, z wie Zeilenvektor
  ma13[1];

  # Problemsituationen
  d1:=evalm(z1&*z);            # Fehler, Spaltenanzahl(z1)<>Zeilenanz.(z)
  d2:=evalm(z1&*transpose(z)); # Fehler, muesste Dyade sein
  d3:=evalm(transpose(z1)&*transpose(z));   # Fehler
  d4:=evalm(transpose(z)&*z1);       # nicht verwertbar
```

$$[1,\ 1,\ 1,\ 1,\ 1]$$

$$:= \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$ma11 := [5]$$

$$5$$

$$ma12 := [\ 5]$$

$$5$$

$$ma13 := [5]$$

$$5$$

```
Error, (in linalg[multiply]) non matching dimensions for vector/matrix product
Error, (in linalg[multiply]) expecting a matrix or a vector
Error, (in linalg[multiply]) expecting a matrix or a vector
```

$$d4 := \mathrm{transpose}([5])$$

Matrixdifferenzen und Dimension der Matrix

```
> A1:=diag(1,1);
  rowdim(A);

  A2:=matrix(2,2,[[1,2],[3,4]]);
  rowdim(A1);

  evalm(A2-A1);
  rowdim(A2-A1);
```

$$A1 := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$2$$
$$A2 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$
$$2$$
$$\begin{bmatrix} 0 & 2 \\ 3 & 3 \end{bmatrix}$$
$$2$$

```
> n:=2;
  II:=array(identity,1..n,1..n);
  evalm(II);
  rowdim(II);
  I0:=array(sparse,1..n,1..n);
  evalm(I0);
  rowdim(I0);
```

$$n := 2$$
$$II := \mathrm{array}(identity, 1 \mathinner{..} 2, 1 \mathinner{..} 2, [\ ])$$
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$2$$
$$I0 := \mathrm{array}(sparse, 1 \mathinner{..} 2, 1 \mathinner{..} 2, [\ ])$$
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$
$$2$$

```
> evalm(II-II);       # 0-Wert-Matrix
  rowdim(II-II);      # Fehler
  evalm(A1-A1);       # 0-Wert-Matrix
  rowdim(A1-A1);      # Fehler
  evalm(A1-A1-I0);    # 0-Matrix
  rowdim(A1-A1-I0);
  evalm(A1-II);       # 0-Matrix
  rowdim(A1-II);
```

$$0$$
```
Error, (in rowdim) first argument is zero, need zero matrix
```
$$0$$
```
Error, (in rowdim) first argument is zero, need zero matrix
```
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$
$$2$$
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$
$$2$$

Potenzen

mit Vektoren

```
> evalm(x), evalm(b), evalm(c);
  evalm(x^0);

  evalm(x^0+x^1);
  evalm(x^0-b^0);
  evalm(x^0-x);    # 1-[1,1,1,1]=[0,0,0,0]
  evalm(x^0-c);
```

$$[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1, 0, 0, 0, 0]$$
$$1$$
$$[2, 2, 2, 2, 2]$$
$$0$$
$$[0, 0, 0, 0, 0]$$
$$[0, 1, 1, 1, 1]$$

mit Matrizen

```
> evalm(A2), evalm(II);
  evalm(A2^0);
  rowdim(evalm(A2^0));    # Fehler

  evalm(A2^0+A2^1);
  evalm(A2^0-A1^0);
  evalm(A2^0-A2);         # A2^0-A2=1-A2=I-A2
  evalm(A2^0-II);         # A2^0-II=1-II=0-Matrix
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$1$$

```
Error, (in rowdim) expecting a matrix
```

$$\begin{bmatrix} 2 & 2 \\ 3 & 5 \end{bmatrix}$$
$$0$$
$$\begin{bmatrix} 0 & -2 \\ -3 & -3 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

2.6 Aufwand und Zeitmessungen beim Umgang mit Matrizen

Bei der Einschätzung der Effizienz von Algorithmen werden oft drei wichtige Probleme betrachtet: Aufwand an Operationen, Rechenzeit, Speicherbedarf. Aber dazu kann man auch noch andere Aspekte einbeziehen, wie die Vorbereitung von Daten, die Auswertung der Ergebnisse, Dateiarbeit allgemein, Ergibtanweisungen, die Arbeit mit Steuerstrukturen u. ä. Aus dieser Vielfalt wird man sich jedoch auf einige wesentliche Fragen beschränken. Manchmal bleibt es dann bei der Bestimmnung der Anzahl der arithmetischen Operationen, also bei der Berechnung einer sogenannten Komplexitätsfunktion in Abhängig vom Problemumfang (Dimension), sowie bei Rechenzeitvergleichen. Die Programmiersprachen und CAS bieten oft mehrere Möglichkeiten der Unterstützung bei Aufwandsbetrachtungen.
In Maple kann man Zeitmessungen vornehmen und dann auch bei Kenntnis der Komplexitätsfunktion näherungsweise eine durchschnittliche Anzahl von GP-Operationen (*floating point operations*, $flops$) pro Sekunde für die benutzte Rechnerplattform ermitteln. Die Kommandos für die Zeit sind grob notiert

```
> ta := time():
  y := evalm(A&*x);   # Algorithmus
  te := time():
  tdiff := te-ta:
```

Matlab bietet zwei Varianten, einmal zur Bestimmung der Anzahl der durchgeführten GP-Operationen mit `flops` sowie die Zeitmessungen mit `clock, etime`.

```
t0 = clock;
flops(0);
y = A*x;      % Algorithmus
fl = flops;   % oder einfach flops
t1 = etime(clock,t0);
```

Das Kommando `flops(0)` (nicht `flops = 0`) setzt den Zähler auf Null zurück. So kann die Eingabe von `flops(0)` unmittelbar vor dem Beginn des Algorithmus und der Aufruf `flops` gleich nach seiner Beendigung die $flops$ ermitteln. Die Funktion `clock` gibt die aktuelle Zeit mit der Genauigkeit auf eine Hundertstel Sekunde an. Mit zwei solchen Zeiten kann `etime` die abgelaufene Zeit (Zeitdifferenz) in Sekunden bestimmen. Seit der Version 4.0 gibt es die bequemere Variante einer Stoppuhr mit `tic, toc`, ab der Version 6 ist das Kommando `flops` nicht mehr verfügbar.

Bei der Anzahl der GP-Operationen wird man bestrebt sein, die verschiedenen arithmetischen Operationen extra zu zählen, denn auf den Rechnern sind die Zeiten dafür durchaus nicht gleich. Inzwischen dauert auf modernen Computern die Auswertung einer Addition, Subtraktion oder Multiplikation ungefähr gleich lang, während die Division schon mehr Zeit braucht und die Berechnung von Standardfunktionen natürlich wesentlich länger dauern kann.

2.6.1 Besonderheiten von Rechnersystemen

Es steht zunächst das Problem, ob man nicht für klassische Algorithmen die Eigenschaften des Rechnersystems und/oder des Compilers ausnutzen kann, um die Laufzeit zu optimieren. Möglicherweise bringt eine solche Optimierung letztendlich mehr Gewinn als die Implementierung eines neuen Algorithmus.
In diesem Abschnitt untersuchen wir die Fragen für das Problem der Basisoperationen auf PC-Technik. Dabei werden wir vorhandene Möglichkeiten an rechnerspezifische Gegebenheiten anpassen.
Es ist klar, dass solche Betrachtungen sehr von dem verwendeten Rechnertyp abhängen. Auf skalaren Rechnern wird dies zu anderen Ergebnissen führen als auf Hochleistungsrechnern, die Vektorinstruktionen verwenden oder sogar parallele Verarbeitung ermöglichen.

Selbst auf skalaren Rechnern wird die Problemstellung von einer Vielzahl von Parametern beeinflusst, wie etwa:

- schnelle und große Rechenregister, Cache-Speicher,
- interne Parallelität des Rechenwerkes (Prozessor, CPU),
- Optimierer, die bestimmte Sprachkonstrukte sehr effektiv behandeln, andere aber nur weniger effektiv,
- Verwendung von Maschinencode.

Auch ohne die explizite Verwendung von Maschinespracheprogrammen ist für die Detailoptimierung ein weites Betätigungsfeld gegeben.
Häufig wird die Rechenzeit in direkten Zusammmenhang mit der Anzahl der arithmetischen Operationen gebracht. Das ist ein wichtiges Kriterium. Aber im Zuge der Hardwareentwicklung sollte man andere beachtliche Ergebnisse keinesfalls unberücksichtigt lassen.
So ist z. B. eine Multiplikation beim i-Pentium genauso schnell wie eine Addition.
Das Quadrieren einer Zahl ist gar doppelt so schnell wie die Addition. Zahlreiche verbesserte numerische Verfahren beruhen darauf, auf Kosten von Additionen einige Multiplikationen einzusparen, das bringt auf modernen Rechnern keinen Zeitgewinn mehr. Einsparungen an Multiplikationen sind auch dann nicht mehr unbedingt sinnvoll, wenn eine Zuweisung mehr Zeit als eine GP-Multiplikation benötigt (beim i-Pentium gar mehr als 200% der Zeit).
Die Kontrollstrukturen verschiedener Compiler für Schleifen, Vergleiche u. v. m. sind sehr unterschiedlich gestaltet. Hier kann man zahlreiche Variationen der Implementation austesten.

Diese und andere Untersuchungen sollte man jedem Optimierungsangriff auf ein numerisches Verfahren vorausschicken. Die speziellen Systemvoraussetzungen sind zunächst zu prüfen. Anschließend kann man den Hebel an den Schwachstellen ansetzen.

2.6.2 Systemtest

Zunächst sollen einige Untersuchungen zu den Basisoperationen in den höheren Programmiersprachen C und Turbo Pascal (TP) erfolgen. Damit kann man dann auch Vergleiche anstellen und sich Vorstellungen über die Leistungsfähigkeit von CAS verschaffen.
Sämtliche Tests wurden auf PCs unter dem Betriebssystem MS-DOS gemacht. Zahlreiche Versuche haben gezeigt, dass andere Betriebssysteme diese Geschwindigkeiten kaum erreichen.
Die Rechenzeit eines Programmteils lässt sich in seine elementaren Bestandteile zerlegen. Eine Bestimmung der einzelnen Geschwindigkeiten gibt oft schon einen Einblick in das Gesamtverhalten. Man kann also schon theoretisch Vorhersagen für die reale Rechenzeit machen. Dabei können jedoch auch so manche Überraschungen auftreten. Das endgültige Verständnis für viele Effekte lässt sich meist erst auf Assemblerebene klären. Hier kann man die Qualität eines Compilers genau erkennen.

Operation	AMD 486DX4-100	i-Pentium 90
Leere Schleife	0.702	0.880
Zuweisung	1.780	1.218
Addition	1.220	0.568
Subtraktion	1.221	0.568
Multiplikation	1.526	0.568
Division	7.416	4.558
Quadrat	1.220	0.232
Wurzel	8.218	7.888

Tab. 2.1 Geschwindigkeiten für 10 Millionen skalare GP-Operationen in *sec*, Borland C++ 3.1

Mittels dieser Werte lässt sich nun beispielsweise die Zeit für folgende Kompaktanweisung bestimmen.

```
for ( i=1; i<10000000; i++ ) c=a*b;
```

Die gemessene Gesamtzeit der Schleife für den 486DX-PC beträgt ca. $4sec$. Sie berechnet sich gemäß

$$\begin{aligned} t_* &= < LeereSchleife > + < Zuweisung > + < Multiplikation > \\ &= 0.702 + 1.780 + 1.526 = 4.008. \end{aligned}$$

Die angegebenen Zeiten sind jedoch stark vom verwendeten Compiler abhängig. Des Weiteren versagt diese Idee bei relativ komplexen Programmen, dort können z. B. die Steueranweisungen einiges durcheinander bringen.

Dazu kommen noch Besonderheiten bezüglich der Auswertung spezieller Formen von Ausdrücken. Erfolgt die Zeitmessung bei Ausführung der Schleifenanweisung

```
for ( i=1; i<10000000; i++ ) c=a*b+y;
```

nach obigen Muster, würde man

$$\begin{aligned} t_{*+} &= < LeereSchleife > + < Zuweisung > + < Multipl. > + < Add. > \\ &= 0.702 + 1.780 + 1.526 + 1.220 = 5.228 \; sec. \end{aligned}$$

erhalten. Eine konkrete Rechnung dauert nicht so lang wie t_{*+}. Der Grund liegt in der günstigen Auswertung von sogenannten *axpy*-Ausdrücken als besondere Einheit der Gestalt $a * x + y$ im Prozessor des Rechners.
In der Tab. 2.1 fällt die relativ hohe Arbeitszeit für eine leere Schleife auf. Wodurch entsteht diese? Das Problem ist nicht das Verwalten der Zählvariable. Der Sprung (JMP), den der Prozessor für jeden Durchlauf ausführt, ist die Lösung. Genau genommen ist es ein bedingter Sprung, der bei allen Intel-Prozessoren die Befehlswarteschlange löscht. Der i-Pentium hat damit noch größere Probleme, da die parallele Pipeline-Struktur hier eher stört als nützt. Daraus ergibt sich eine Anforderung an die Hardwarehersteller bezüglich immer schnellerer Speicher.

Die einzelnen Geschwindigkeiten werden nun auf dem PC 80486DX2-S (66MHz) und später PC Pentium III (800MHz) in der Sprache TP 7.0 getestet. Dabei sollen die Programmierung im Assemblercode und mit den üblichen Sprachkonstrukten gegenübergestellt werden.

Für die Zeitmessungen liegt die Unit UHR.PAS vor.

```
{$N+}
Unit Uhr;

Interface
Procedure _Time_Zero;
Procedure _Time_Go;
Procedure _Time_Stop;
Function  _Time_sec : Double;

Implementation
Uses dos;
Var  _time                    : Double;
     a1,a2,b1,b2,c1,c2,d1,d2 : Word;

Procedure _Time_Zero;
  Begin _time:=0.0; End;

Procedure _Time_Go;
  Begin Gettime(a1,b1,c1,d1); End;
```

```
Procedure _Time_Stop;
  Var  x,y : Double;
  Begin
    Gettime(a2,b2,c2,d2);
    x:=a1*3600+b1*60+c1+d1/100;
    y:=a2*3600+b2*60+c2+d2/100;
    _time:=_time+y-x;
  End;
Function _Time_sec : Double;
  Begin _Time_sec:=_time;  End;

Begin
  _Time_Zero;
End.
```

Den Test der schon genannten Operationen führen wir mit dem Rahmenprogramm FPUSPEE1.PAS durch.
Die Anzahl der Durchläufe in den Schleifenkonstrukten beträgt
$10^7 = 1000 * 10000 = con * con1$.

```
{$N+}
program FPUTest;

uses crt,dos,uhr;

type testproc = procedure;
var  k          : char;
     a,b,x      : double;
     i          : word;
     time,time1 : array [1..8] of double;
     test       : array [1..8] of testproc;
     con,con1   : word;
     j,jmax     : longint;

{$F+ }
procedure Test1;  assembler;
  asm
    FINIT

    MOV  DX,con
  @N:  MOV  CX,con1
  @M:
       LOOP @m
       DEC  DX
       Jnz  @N
  end;
```

```
{x:=a;}
procedure Test2;  assembler;
  asm
    FINIT

    MOV  DX,con
  @N:  MOV  CX,con1
  @M:  FLD  qword ptr a
       FSTP qword ptr x

       LOOP @m
       DEC  DX
       Jnz  @N
  end;

{analog
  x:=a+b; Test3;
          @M:  FLD  qword ptr a
               FADD qword ptr b
               FSTP qword ptr x

  x:=a-b; Test4;
          @M:  FLD  qword ptr a
               FSUB qword ptr b
               FSTP qword ptr x

  x:=a*b; Test5;
          @M:  FLD  qword ptr a
               FMUL qword ptr b
               FSTP qword ptr x

  x:=a/b; Test6;
          @M:  FLD  qword ptr a
               FDIV qword ptr b
               FSTP qword ptr x

  x:=sqr(a); Test7;
          @M:  FLD  qword ptr a
               FMUL ST,ST
               FSTP qword ptr x

  x:=sqrt(a); Test8;
          @M:  FLD  qword ptr a
               FSQRT
               FSTP qword ptr x
}
```

```
{$F- }
begin
  clrscr;
  writeln('Systemtest - Speed');
  con:=1000;  con1:=10000;
  delay(2000); clrscr;
  test[1]:=test1;
  test[2]:=test2;
  test[3]:=test3;
  test[4]:=test4;
  test[5]:=test5;
  test[6]:=test6;
  test[7]:=test7;
  test[8]:=test8;
  a:=14.25;
  b:=-1314E-2;
  x:=13.41E1;

  writeln('TP - Assembler');
  writeln;
  for i:=1 to 8 do
    begin
      _time_zero;
      _time_go;
      test[i];
      _time_stop;
      time[i]:=_time_sec;
      writeln('Zeit t[',i,']: ',_time_sec:10:2,'s');
    end;

  writeln;
  writeln('Schleife             : ',time[1]:10:2);
  writeln('Schleife + Zuw. X:=A : ',time[2]:10:2);
  writeln('Zuweisung       X:=A : ',time[2]-time[1]:10:2);
  writeln('ADD              A+B : ',time[3]-time[2]:10:2);
  writeln('SUB              A-B : ',time[4]-time[2]:10:2);
  writeln('MUL              A*B : ',time[5]-time[2]:10:2);
  writeln('DIV              A/B : ',time[6]-time[2]:10:2);
  writeln('SQR              A*A : ',time[7]-time[2]:10:2);
  writeln('SQRT             ... : ',time[8]-time[2]:10:2);
  k:=readkey;
  if k=' ' then halt;
  clrscr;
  writeln('TP');
  writeln;
  jmax:=longint(con)*longint(con1);
```

```
  _time_zero;
  _time_go;
  for j:=1 to jmax do;
  _time_stop;
  time1[1]:=_time_sec;
  writeln('Zeit  (loop)         : ',_time_sec:10:2,'s');

  _time_zero;
  _time_go;
  for j:=1 to jmax  do x:=a;
  _time_stop;
  time1[2]:=_time_sec;
  writeln('Zeit  (loop,:=)      : ',_time_sec:10:2,'s');

{analog

  for j:=1 to jmax  do x:=a+b;     --> time1[3]
  for j:=1 to jmax  do x:=a-b;     --> time1[4]
  for j:=1 to jmax  do x:=a*b;     --> time1[5]
  for j:=1 to jmax  do x:=a/b;     --> time1[6]
  for j:=1 to jmax  do x:=sqr(a);  --> time1[7]
  for j:=1 to jmax  do x:=sqrt(a); --> time1[8]
}

  writeln;
  writeln;
  writeln('Schleife             : ',time1[1]:10:2);
  writeln('Schleife + Zuw. X:=A : ',time1[2]:10:2);
  writeln('Zuweisung       X:=A : ',time1[2]-time1[1]:10:2);

  writeln('ADD              A+B : ',time1[3]-time1[2]:10:2);
  writeln('SUB              A-B : ',time1[4]-time1[2]:10:2);
  writeln('MUL              A*B : ',time1[5]-time1[2]:10:2);
  writeln('DIV              A/B : ',time1[6]-time1[2]:10:2);

  writeln('SQR              A*A : ',time1[7]-time1[2]:10:2);
  writeln('SQRT             ... : ',time1[8]-time1[2]:10:2);

  k:=readkey;
end.
```

Bei den Operationen $\{+, -, *, /\}$ ist in der Gesamtzeit noch der Zugriff auf die dritte Variable gemäß $x := a \circ b$ enthalten.

Die Ergebnisse der Geschwindigkeitsmessungen werden tabellarisch angegeben.

Operation	Assembler	Hochsprache
Leere Schleife	1.04	2.25
Zuweisung	1.97	2.58
Addition	1.48	1.53
Subtraktion	1.49	1.48
Multiplikation	2.25	2.10
Division	10.98	11.05
Quadrat	1.47	2.19
Wurzel	13.30	15.70

Tab. 2.2 Geschwindigkeiten für 10 Millionen skalare GP-Operationen in *sec* auf PC 80486DX2-S (66MHz), TP 7.0

Lässt man Steueranweisungen und die rechenintensiven Operationen unberücksichtigt, nimmt man also nur die Grundoperationen $\{+, -, *\}$, so braucht eine GP-Operation auf diesem PC durchschnittlich $2 \cdot 10^{-7}$ *sec*.
Ähnlich sind die Relationen auf dem schnelleren PC Pentium III (GenuineIntel x86, 800MHz) unter MS Windows98. Die Anzahl der Durchläufe in den Schleifenkonstrukten beträgt $10^8 = 10000 * 10000 = con * con1$.

Operation	Assembler	Hochsprache
Leere Schleife	0.71	0.87
Zuweisung	0.16	0.27
Addition	0.22	0.17
Subtraktion	0.27	0.32
Multiplikation	0.22	0.17
Division	3.68	3.45
Quadrat	0.22	0.28
Wurzel	7.52	15.05

Tab. 2.3 Geschwindigkeiten für 100 Millionen skalare GP-Operationen in *sec* auf PC Pentium III (800MHz), TP 7.0

Die rechenintensiven Operationen werden aber vergleichsweise langsamer.
Nimmt man nur die Grundoperationen $\{+, -, *\}$, so gibt es kaum noch Unterschiede zwischen beiden Varianten, außer dem, dass der PC Pentium III fast 100 Mal schneller ist. Eine GP-Grundoperation auf dem PC Pentium III dauert durchschnittlich $3 \cdot 10^{-9}$ *sec*. Dieser gute Durchschnitt wird aber durch die anderen aufwendigeren Operationen beeinträchtigt.

Ähnliche Betrachtungen kann man auch bezüglich Aufwand und Rechenzeiten in CAS Matlab anstellen, in dem die Numerik auch mit dem *double*-Format arbeitet. Wir nehmen das Matrixprodukt

$$C \;=\; AB \;\text{ mit } C=(c_{ij}),\; c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj},\; i,j=1,2,...,n,$$

bei voll besetzten Matrizen und führen einige Zeitmessungen durch. Wegen der geschachtelten Schleifen und der Indexarbeit bei den Feldern ist es nicht so einfach, die genauen Zeiten für die einzelnen Befehle zu ermitteln.

1. Matlab 4 auf PC 80486DX2-S
   Die Kommandofolge mit Zeitmessung dazu ist

```
n=30; m=n^3;
A=hilb(n); B=A; r=...; t=...; s=...; u=...;
tic;
for i=1:n, for j=1:n,
 s=0;
 for k=1:n, s=s+A(i,k)*B(k,j); end;
 C(i,j)=s;
end; end;
toc
```

   Zum Vergleich dazu rechnen wir, wenn einige Anweisungen davon entfallen bzw. verändert werden.

Version	Zeit (*sec*)
for i=1:m, end;	0.22
for i=1:n, for j=1:n, for k=1:n, end;end;end;	0.60
for i=1:n, for j=1:n, s=0; for k=1:n, end;end;end;	0.66
for i=1:n, for j=1:n, for k=1:n, end; C(i,j)=s; end;end;	0.71
for i=1:n, for j=1:n, s=0; for k=1:n, end; C(i,j)=s; end;end;	0.71
for i=1:n, for j=1:n, for k=1:n, A(i,k)*B(k,j); end;end;end;	6.92
for i=1:n, for j=1:n, for k=1:n, s+A(i,k)*B(k,j); end;end;end;	7.63
for i=1:n, for j=1:n, for k=1:n, s=s+A(i,k)*B(k,j); end;end;end;	7.74
for i=1:n, for j=1:n, s=0; for k=1:n, s=s+A(i,k)*B(k,j); end;end;end;	7.85
for i=1:n, for j=1:n, for k=1:n, s=s+A(i,k)*B(k,j); end; C(i,j)=s; end;end;	7.91
for i=1:n, for j=1:n, s=0; for k=1:n, s=s+A(i,k)*B(k,j); end; C(i,j)=s; end;end;	8.02

Tab. 2.4 Rechenzeiten zu $C=AB,\; n=30,\; m=n^3$, in verschiedenen Versionen in *sec* auf PC 80486DX2-S (66MHz), Matlab 4

Natürlich wird in der Vollversion die innerste Schleife den Zeitaufwand wesentlich bestimmen. Mit der geschachtelten Dreifachschleife, den 2 Operatoren $+, *$, 3 *double*-Größen s, A, B und 4 Indizes i, k, k, j kommt man grob auf $0.6 + 2 * 1 + 3 * 1 + 4 * 0.5 = 7.6$ *sec* und damit in die Nähe der Gesamtzeit. Um dies zu untermauern und mehr Informationen über den Zeitaufwand bei Ausführung von Basisoperationen zu erhalten, machen wir noch einige zusätzliche Testläufe.

Version	Zeit (*sec*)
for i=1:n, for j=1:n, for k=1:n, end;end;end;	0.60
for i=1:m, end;	0.22
for i=1:m, r; end;	1.26
for i=1:m, r;s; end;	2.25
for i=1:m, r;s;t; end;	3.29
for i=1:m, r;s;t;u; end;	4.23
for i=1:m, r+t; end;	4.06
for i=1:m, r*t; end;	4.06
for i=1:m, r*r; end;	4.01
for i=1:m, r/t; end;	4.12
for i=1:m, r^2; end;	5.88
for i=1:m, r*t+u; end;	4.83
for i=1:m, r*(t+u); end;	4.78
for i=1:m, r*t+s*u; end;	5.55
for i=1:m, s=r; end;	2.30
for i=1:m, s=r+t; end;	4.17
for i=1:m, s=r*t; end;	4.11
for i=1:m, s=r*r; end;	4.12
for i=1:m, s=r/t; end;	4.19
for i=1:m, s=r^2; end;	6.15
for i=1:m, s=r*t+u; end;	4.99
for i=1:m, s=r*(t+u); end;	4.89
for i=1:m, s=r*t+s*u; end;	5.72

Tab. 2.5 Rechenzeiten zu Basisoperationen in verschiedenen Versionen in *sec* auf PC 80486DX2-S (66MHz), $m = n^3 = 27000$, Matlab 4

Die 4 Grundoperationen $\{+, -, *, /\}$ führen ungefähr auf gleiche Rechenzeiten, die Division ist also nicht schlechter. Nimmt man die reine Operationszeit, so kommt man dabei auf ca. 30000 Operationen pro Sekunde. Ähnlich lange dauert der Zugriff auf eine einfache Variable vom Typ *double*, so dass sich die Matlab-Anweisungen `for i=1:m, s=r; end;` und `for i=1:m, r;s; end;` in der Rechenzeit kaum unterscheiden.

Die Auswertung eines Ausdrucks oder die Übergabe seines Wertes an die Variable s macht kaum einen Unterschied in der Rechenzeit aus. Das ist damit zu erklären, dass für die erste Variante intern eine Hilfsvariable bereitgestellt wird, mit der wie mit der "Ergebnisgröße" s umgegangen wird.

2. Matlab 6 auf PC Pentium III

   Wir ermitteln wieder die Zeit zum Matrixprodukt $C = AB$ in der Vollversion sowie zum Vergleich dazu Varianten, wenn einige Anweisungen davon entfallen bzw. verändert werden. Dabei nehmen wir die Dimension $n = 90$.

```
n=90; m=n^3;
A=hilb(n); B=A; r=...; t=...; s=...; u=...;
tic;
for i=1:n, for j=1:n,
 s=0;
 for k=1:n, s=s+A(i,k)*B(k,j); end;
 C(i,j)=s;
end; end;
toc
```

Version	Zeit (*sec*)
for i=1:m, end;	0.22
for i=1:n, for j=1:n, for k=1:n, end;end;end;	0.33
for i=1:n, for j=1:n, s=0; for k=1:n, end;end;end;	0.33
for i=1:n, for j=1:n, for k=1:n, end; C(i,j)=s; end;end;	0.33
for i=1:n, for j=1:n, s=0; for k=1:n, end; C(i,j)=s; end;end;	0.33
for i=1:n, for j=1:n, for k=1:n, A(i,k)*B(k,j); end;end;end;	5.38
for i=1:n, for j=1:n, for k=1:n, s+A(i,k)*B(k,j); end;end;end;	6.10
for i=1:n, for j=1:n, for k=1:n, s=s+A(i,k)*B(k,j); end;end;end;	4.33
for i=1:n, for j=1:n, s=0; for k=1:n, s=s+A(i,k)*B(k,j); end;end;end;	4.34
for i=1:n, for j=1:n, for k=1:n, s=s+A(i,k)*B(k,j); end; C(i,j)=s; end;end;	4.34
for i=1:n, for j=1:n, s=0; for k=1:n, s=s+A(i,k)*B(k,j); end; C(i,j)=s; end;end;	4.34

Tab. 2.6 Rechenzeiten zu $C = AB,\ n = 90,\ m = n^3$, in verschiedenen Versionen in *sec* auf PC Pentium III (800MHz), Matlab 6

Natürlich wird in der Vollversion die innerste Schleife den Zeitaufwand wieder wesentlich bestimmen. Mit der geschachtelten Dreifachschleife, den 2 Operatoren $+, *$, 3 *double*-Größen s, A, B und 4 Indizes i, k, k, j kommt man grob auf $0.33 + 2 * 0.6 + 3 * 0.6 + 4 * 0.25 = 4.3$ *sec* und damit in die Nähe der Gesamtzeit.

Um mehr Informationen über den Zeitaufwand bei der Ausführung von GP-Operationen zu erhalten, machen wir auch hier einige zusätzliche Testläufe.

Version	Zeit (*sec*)
for i=1:n, for j=1:n, for k=1:n, end;end;end;	0.33
for i=1:m, end;	0.22
for i=1:m, r; end;	0.93
for i=1:m, r;s; end;	1.59
for i=1:m, r;s;t; end;	2.20
for i=1:m, r;s;t;u; end;	2.86
for i=1:m, r+t; end;	3.51
for i=1:m, r*t; end;	3.57
for i=1:m, r*r; end;	3.57
for i=1:m, r/t; end;	3.51
for i=1:m, r^2; end;	3.79
for i=1:m, r*t+u; end;	4.34
for i=1:m, r*(t+u); end;	4.23
for i=1:m, r*t+s*u; end;	4.95
for i=1:m, s=r; end;	1.15
for i=1:m, s=r+t; end;	1.86
for i=1:m, s=r*t; end;	2.08
for i=1:m, s=r*r; end;	2.04
for i=1:m, s=r/t; end;	1.92
for i=1:m, s=r^2; end;	2.30
for i=1:m, s=r*t+u; end;	2.69
for i=1:m, s=r*(t+u); end;	2.63
for i=1:m, s=r*t+s*u; end;	3.62

Tab. 2.7 Rechenzeiten zu Basisoperationen in verschiedenen Versionen in *sec* auf PC Pentium III (800MHz), $m = n^3 = 729000$, Matlab 6

Die 4 Grundoperationen $\{+, -, *, /\}$ führen ungefähr auf gleiche Rechenzeiten. Nimmt man die reine Operationszeit, so kommt man dabei auf ca. 1000000 Operationen pro Sekunde. Geringfügig länger dauert der Zugriff auf eine einfache Variable vom Typ *double*.

Auffällig ist, dass die reine Auswertung eines Ausdrucks ohne die Zuweisung zu einer Variablen zeitmäßig viel schlechter ist als die Darstellung mit einer Ergibtanweisung. Hat etwa Matlab 6 Probleme, mit dem Wert eines solchen Ausdrucks geeignet umzugehen? Es ist zu vermuten, dass dieses Frage in die Richtung der Betrachtung der erweiterten Syntax wie in der Programmiersprache TP gehen wird. Aber darauf soll hier nicht weiter eingegangen werden.
Genauso aufwendig würden diesbezügliche Untersuchungen in Maple wegen der genauen Arithmetik und der Vielfalt der GPA mit `Digits:=...;` sein.

2.6.3 Generierung von Matrizen sowie Matrix-Vektor-Multiplikation

Die klassische Matrix-Vektor-Multiplikation n-dimensionaler Matrizen und Vektoren $y = Ax$ braucht n^2 Multiplikationen und $n^2 - n$ Additionen.
Diesen Aufwand drücken wir durch die Komplexitätsfunktion

$$\mathcal{K}(n) \;=\; 2n^2 - n$$

aus. Wir notieren dies auch in der Form $\mathcal{K}(n) = \mathcal{O}(2n^2)$, wo nur der führende Ausdruck (Potenz) steht. Im Operationsmix $\{+, *\}$ ist der Aufwand $\mathcal{O}(n^2)$ und er muss also verdoppelt werden.
Für eine Bandmatrix mit der Bandbreite $bw = \alpha + \beta + 1 \ll n$ erhält man die Komplexität $\mathcal{O}(2n(\alpha + \beta + 1))$, im Fall einer Diagonalmatrix die Größe $\mathcal{O}(n)$.
Somit wird schon deutlich, dass man einfache Matrixstrukturen auch adäquat verarbeiten sollte, um den Aufwand klein zu halten.

Mit dem CAS Maple untersuchen wir nun den Aufwand und machen dazu Zeitmessungen für folgende Aufgaben:

- Generierung/Erzeugung von Matrizen,
- Umspeichern von Matrizen,
- Matrix-Vektor-Multiplikation.

Dabei nehmen wir verschiedene Matrixstrukturen. Außerdem sind exakte Rechnungen (mit Rationalarithmetik) wie solche in einer GPA möglich. Um Vergleiche zu höheren Programmiersprachen und Matlab anzustellen, wo das *double*-Format gebräuchlich ist, nehmen wir in Maple die Rechengenauigkeit `Digits:=16:`.
Die Betrachtungen beziehen sich zwar nur auf einige Situationen, sind aber durchaus hinreichend aussagekräftig und können zu Vergleichen herangezogen werden.

Rechnungen in Matlab

Matlab hat bis zur Version 5 die Möglichkeit geboten, die Anzahl der durchgeführten GP-Operationen mit `flops` zu messen. Da die Algorithmen diesbezüglich nicht optimal ausgelegt waren, mag das wohl der Grund für die Streichnug diese Kommandos in den weiteren Versionen gewesen sein.
Die Befehlsfolge zur Matrix-Vektor-Multiplikation mit der Problemdimension $n = 4$

```
flops(0);
y = A*x;      % Algorithmus
flops
```

liefert das Ergebnis $flops = 32 = 2n^2$.

Das entspricht der Darstellung und dem Ergebnis von

```
flops(0);
for i=1:4,     % Algorithmus
 z=0;
 for j=1:4,
  z=z+A(i,j)*x(j);
 end;
 y(i)=z;
end;
flops
```

Aber mit der Kommandofolge

```
flops(0);
for i=1:4,      % Algorithmus
 z=A(i,1)*x(1);
 for j=2:4,
  z=z+A(i,j)*x(j);
 end;
 y(i)=z;
end;
flops;
```

beträgt der Aufwand nur $n^2 - n = 28$ Operationen, was zwar nicht die seine Größenordnung beeinflusst und damit nicht zu sichtbarer Zeitersparnis führt, aber die Art und Weise der Abarbeitung von Befehlen doch näher beleuchtet.
Nimmt man die Kommandofolge

```
flops(0);
for i=1:4,         % Algorithmus, allgemein 4->n
 z=A(i,4)*x(4);
 for j=4-1:-1:1,   % for j=n-1:-1:1
  z=z+A(i,j)*x(j);
 end;
 y(i)=z;
end;
flops
```

so ist der Aufwand wiederum $n^2 = 32$ Operationen durch die zusätzliche Subtraktion im Anfangswert der j-Schleife.
Unter solchen Gesichtspunkten sind entsprechende Kommmados in den CAS, nicht nur in Matlab oder Maple, stets kritisch zu sehen und zu beurteilen.

Rechnungen in Maple (Datei: *timearr1.mws*)

Erzeugung von GP-Zahlen in Matrizen, die exakt (symbolisch) definiert sind

Berechnungsfunktionen `evalm, evalf`

```
> H := hilbert(4);
  x := vector(4,[seq(i,i=1..4)]);
  y := evalm(H&*x);
  Digits := 16:
  Hf := evalf(H);          # noch keine GPZ erzeugt
  Hf := evalf(evalm(H));   # GPZ
  Hf := evalf(hilbert(4));

  xf := evalf(x);          # noch keine GPZ erzeugt
  xf := evalf(evalm(x));   # GPZ
  xf := evalf(vector(4,[seq(i,i=1..4)]));

  y := Hf&*xf;
  y;
  y := evalm(Hf&*xf);      # Berechnung
```

$$H := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

$$x := [1, 2, 3, 4]$$

$$y := \left[4, \frac{163}{60}, \frac{21}{10}, \frac{241}{140}\right]$$

$$Hf := H$$

$$Hf :=$$

$$\begin{bmatrix} 1. & 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 \\ 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 \\ 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 \\ 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 & 0.1428571428571429 \end{bmatrix}$$

$$Hf :=$$

$$\begin{bmatrix} 1. & 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 \\ 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 \\ 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 \\ 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 & 0.1428571428571429 \end{bmatrix}$$

$$xf := x$$

$$xf := [1., 2., 3., 4.]$$

$$xf := [1., 2., 3., 4.]$$

$$y := Hf \ \&^* \ xf$$

$$Hf \ \&^* \ xf$$

$$y := [4.000000000000000, 2.716666666666667, 2.100000000000000, 1.721428571428572]$$

Generierung und Umspeichern von Matrizen

Die Rechnungen wurden auf dem PC Pentium III (800MHz) durchgeführt.

- Blocktridiagonalmatrizen vom 2D-Laplace-Operator auf quadratischem Gitter mit 5-Punkte-Differenzenstern (`BTDM`)
- (voll besetzte) Hilbert-Matrix (`H-M`)

Matrizen der Dimension 9,16,25,...,625 im *double*-Format

```
> kmax := 25:
  time1 := vector(kmax,[0$kmax]):
  time2:=evalm(time1): time3:=:evalm(time1) time4:=:evalm(time1)

  Digits := 16:
  for n1 from 3 to kmax do
   n1q := n1^2:

   sta := time():
   T := band([-1.0,4.0,-1.0],n1):
   B9 := diag(T$n1):
   for i from 1 to n1q-n1 do
    B9[i,i+n1] := -1.0;
    B9[i+n1,i] := -1.0;
   end do:
   time1[n1] := time()-sta;

   sta := time():
   B9m := evalm(B9):
   time2[n1] := time()-sta;

   sta := time():
   B9n := evalf(evalm(hilbert(n1q))):
   time3[n1] := time()-sta;

   sta := time():
   for i from 1 to n1q do
    for j from 1 to n1q do
     B9o[i,j] := 1.0/(i+j-1);
    end do:
   end do:
  time4[n1] := time()-sta;
  end do:

  printf(‘Zeiten in Sekunden\n‘):
  printf(‘      Gen(BTDM)      Umsp(vM)       Gen(H-M)
                                              Gen(H-M) p.H.\n‘):
  printf(‘n1^2  time1[3..kmax] time2[3..kmax] time3[3..kmax]
                                              time4[3..kmax]\n‘):
  for k from 3 to kmax do
   printf(‘%3d  %7.3f        %7.3f        %7.3f        %7.3f\n‘,
          k^2,time1[k],time2[k],time3[k],time4[k]):
  end do:
```

```
Zeiten in Sekunden
      Gen(BTDM)       Umsp(vM)        Gen(H-M)        Gen(H-M) p.H.
n1^2  time1[3..kmax] time2[3..kmax] time3[3..kmax] time4[3..kmax]
  9     .003            .005            .005            .020
 16     .005            .010            .010            .005
 25     .010            .010            .024            .021
 36     .010            .025            .045            .040
 49     .010            .045            .895            .075
 64     .010            .080            .285            .130
 81     .015            .125            .380            .325
100     .160            .334            .521            .435
121     .024            .451            .870            .640
144     .029            .571           1.151            .969
169     .309            .986           1.635           1.254
196     .046           1.130           2.535           1.780
225     .050           1.610           3.215           2.420
256     .225           2.195           4.305           3.520
289     .085           2.885           6.085           4.895
324     .500           4.100           8.425           6.705
361     .600           5.820          12.020           8.930
400     .125           7.535          15.695          12.155
441     .435           9.815          19.830          16.150
484     .170          13.560          26.160          21.650
529     .200          17.160          33.835          28.650
576     .810          22.755          44.930          37.990
625     .940          29.789          57.825          50.770
```

Bemerkenswert ist die schnelle Generierung von sparsen Matrizen, wie es die Blocktridiagonalmatrix ist. Lange dauert die Umspeicherung und Generierung großer voll besetzter Matrizen. Dabei ist es günstiger, auf vordefinierte Versionen (in exakter Form) zu verzichten und selbst die Matrixelemente als GP-Zahlen entsprechender Genauigkeit einzutragen.

Zeitmessungen zur Matrix-Vektor-Multiplikation
(1) Voll besetzte Matrix mit verschiedenen Formaten und Strategien

```
> kmax := 10:
  Digits := 16:
  stime:=vector(kmax,[0$kmax]): ftime:=evalm(stime): etime:=evalm(stime):

  for k from 1 to kmax do
   m := 20*k:
   H := hilbert(m):
   x := vector(m,[seq(i,i=1..m)]):
   Hf := evalf(hilbert(m)):
   xf := evalf(vector(m,[seq(i,i=1..m)])):
```

```
    # symbolisch
    sta := time():
    y := evalm(H&*x);
    stime[k] := time()-sta:

    # float
    sta := time():
    yf := evalm(Hf&*xf);
    ftime[k] := time()-sta:

    # float+loop
    sta := time():
    for i from 1 to m do
      z := 0.0;
      for j from 1 to m do z := z+Hf[i,j]*xf[j]; end do;
      yf[i] := evalf(z);
    end do;
    etime[k] := time()-sta:
  end do:

  printf(‘Zeiten in Sekunden fuer y=Ax, A(m,m) voll besetzt\n‘):
  printf(‘      symbolisch       float       float+loop\n‘):
  printf(‘m   stime[1..kmax] ftime[1..kmax] etime[1..kmax]\n‘):
  for k from 1 to kmax do
   printf(‘%3d  %7.3f        %7.3f        %7.3f\n‘,
          k*20,stime[k],ftime[k],etime[k]):
  end do:
```

```
Zeiten in Sekunden fuer y=Ax, A(m,m) voll besetzt
      symbolisch       float       float+loop
m   stime[1..kmax] ftime[1..kmax] etime[1..kmax]
 20     .015           .020           .010
 40     .060           .060           .045
 60     .140           .135           .165
 80     .375           .310           .190
100     .865           .450           .380
120    2.330           .755           .520
140    6.508          1.215           .705
160    8.330          1.605           .915
180   33.555          2.140          1.275
200   20.467          2.693          1.690
```

Die exakte Multiplikation dauert natürlich am längsten, denn die dabei zu speichernden rationalen Zahlen haben immer mehr Dezimalstellen. Warum bei der GPA (`Digits:=16:`, Format *double*) die nutzerdefinierten Schleifen zu einer etwas kürzeren Rechenzeit führen, ist nicht nur damit zu erklären, dass eine einfache Hilfsvariable zur Erzeugung der Skalarprodukte genommen wurde. Eine innere Schleife der Form

`yf[i]:=0.0; for j from 1 to m do yf[i]:=yf[i]+Hf[i,j]*xf[j]; end do;`

vergrößert nur geringfügig die Zeiten in der Spalte `etime`.

Damit bringt es Maple mit dem GP-Format *double* auf ca. 40 000 $flops$ + Aufwand für Steuerung pro Sekunde.
Mit Matlab und den Anweisungen `tic; A*b; toc` mit Zeitmessung ist man um Größenordnungen schneller und erhält ca. 70 000 000 $flops$ per *sec*.

(2) Bandstrukturen im Vergleich zur voll besetzten Matrix
Matrizen der Dimension $n1q = 625$

```
> B9d := evalm(diag(seq(1.0/i,i=1..n1q))): # Diagonalmatrix
  B9m:   # Blocktridiagonalmatrix
  B9n:   # (voll besetzte) Hilbert-Matrix

> n1q;
  x := evalf(evalm(vector(n1q,[1.0$n1q]))):

  sta := time():
  y := evalm(B9d&*x):
  print(time()-sta);

  sta := time():
  y := evalm(B9m&*x):
  print(time()-sta);

  sta := time():
  y := evalm(B9n&*x):
  print(time()-sta);
```

625
54.665
56.689
83.585

Die Matrix-Vektor-Multiplikation für eine Diagonalmatrix bzw. Blocktridiagonalmatrix dauert in der GPA ca. 50 *sec*, dabei hat man etwas Zeiteinsparung durch viele Nullen in der Matrix. Die Zeitmessung für eine volle Matrix liefert ca. 80 *sec*.
Somit entstehen in numerischen Algorithmen mit vielen Matrix-Vektor- oder Matrix-Matrix-Produkten bei großen Dimensionen enorme Rechenzeiten.

(3) Diagonalmatrizen
Erhebliche Zeiteinsparungen hat man durch die Berücksichtigung der Diagonalstruktur der Matrizen. Wir vergleichen 4 Varianten von $y = Ax$ und der Behandlung der Matrix $A(m, m)$ als voll besetzte bis zur optimalen Berechnungsformel $y_i = a_{ii}x_i,\ i = 1, 2, ..., m$.

```
> kmax := 5:
  stime := vector(kmax,[0$kmax]):   # zum Speichern der Zeiten
  ftime := evalm(stime): etime :=evalm(stime): dtime :=evalm(stime):
  Digits:=16:
```

```
for k from 1 to kmax do
 m := 100*k: i := 'i':
 C1 := evalm(diag(seq(-11+2*i,i=1..m))): # Van der Vorst Matrix
 x := vector(m,[seq(i,i=1..m)]):
 C1f := evalf(evalm(C1)):
 xf := evalf(evalm(x)):

 # symbolisch
 sta := time():
 y := evalm(C1&*x);
 stime[k] := time()-sta:
 # float
 sta := time():
 yf := evalm(C1f&*xf);
 ftime[k] := time()-sta:
 # loop+float
 sta := time():
 for i from 1 to m do
  z := 0.0;
  for j from 1 to m do z:=z+C1f[i,j]*xf[j]; end do;
  yf[i] := evalf(z);
 end do;
 etime[k] := time()-sta:
 # loop+float+diagonal
 sta := time():
 for i from 1 to m do
  yf[i] := C1f[i,i]*xf[i];
 end do;
 dtime[k] := time()-sta:
end do:
printf(`Zeiten in Sekunden fuer y=Ax, A(m,m) diagonal\n`):
printf(`      symbolisch       float       float+loop
                                          float+loop+diagonal\n`):
printf(` m   stime[1..kmax] ftime[1..kmax] etime[1..kmax] dtime[1..kmax]\n`):
for k from 1 to kmax do
 printf(`%3d %7.3f        %7.3f        %7.3f        %7.3f\n`,
         k*100,stime[k],ftime[k],etime[k],dtime[k]):
end do:
```

```
Zeiten in Sekunden fuer y=Ax, A(m,m) diagonal
      symbolisch       float       float+loop    float+loop+diagonal
 m   stime[1..kmax] ftime[1..kmax] etime[1..kmax] dtime[1..kmax]
100   0.265          0.450          0.160          0.000
200   1.958          2.207          1.005          0.005
300   6.540          6.575          2.865          0.005
400  15.030         16.105          6.365          0.015
500  31.525         34.598         11.440          0.020
```

Bemerkenswert ist, dass bei nur wenigen symbolischen Rechnungen (wegen der Diagonalform) die Rechenzeit dafür mit der der GPA konkurrieren kann. Die nutzerdefinierten Schleifen liefern etwas kürzere Rechenzeit.

2.7 LU-Faktorisierung

In der linearen Algebra treten sehr häufig unterschiedliche Faktorisierungen von Matrizen auf, die LU-Faktorisierung (auch LU-Zerlegung genannt) ist eine davon. Sie ist direkt verbunden mit dem Gaußschen Eliminationsverfahren (Gauß-Algorithmus, GA) zur Lösung von LGS.

Durchführung der **vollständigen LU-Faktorisierung** ohne Pivotisierung von $A = A(n,n) = (a_{ij})$ in eine untere bzw. obere Dreiecksmatrix.

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{pmatrix}.$$

Die Berechnungsvorschrift für L, U auf dem Platz der Matrix A erfolgt entweder mit dem Resttableau-Algorithmus (Tableau für Tableau) oder mittels verkettetem GA (pro Schritt werden nur eine Zeile und Spalte berechnet).

(1) Resttableau-Algorithmus

$A = LU,\ l_{ii} = 1.$

$$\begin{aligned} k &= 1, 2, ..., n \\ p &= a_{kk}, \\ &\quad \text{vorzeitiger Abbruch bei } p = 0 \\ i &= k+1, k+2, ..., n \\ s &= a_{ik}/p \\ a_{ik} &= s \\ a_{ij} &= a_{ij} - s\, a_{kj}, \quad j = k+1, k+2, ..., n \end{aligned}$$

(2) Verketteter Gauß-Algorithmus (VGA)

$A = -CB = LU,\ c_{ii} = -1, \quad L = -C,\ U = B.$

$$\begin{aligned} k &= 1, 2, ..., n \\ p &= a_{kk} + \sum_{i=1}^{k-1} a_{ki} a_{ik}, \\ &\quad \text{vorzeitiger Abbruch bei } p = 0 \\ a_{kk} &= p \\ a_{kj} &= a_{kj} + \sum_{i=1}^{k-1} a_{ki} a_{ij}, \quad j = k+1, k+2, ..., n \\ a_{ik} &= -\left(a_{ik} + \sum_{j=1}^{k-1} a_{ij} a_{jk} \right) / p, \quad i = k+1, k+2, ..., n \end{aligned}$$

Die Durchfürbarkeit ohne Pivotisierung entspricht der Diagonalstrategie und ist möglich, wenn in allen Schritten das Diagonalelement p nicht Null wird. Natürlich ist es günstiger eine andere Pivotstrategie zu verfolgen, denn auch Divisionen durch betragskleine Werte $p \neq 0$ beeinträchtigen die numerische Stabilität der Faktorisierung bzw. des GA.
Als erste Verbesserung betrachten wir nur die Spaltenpivotisierung mit Zeilenvertauschung. Dabei gibt es einige kleine Modifikationen der Wahl des sogenannten Pivotelements (PE) in der jeweiligen Spalte. Zum Pivotelement gehört seine Pivotzeile.

- Wenn ein Diagonalelement verschwindet, dann sucht man darunter in derselben Spalte
  - das erste nicht verschwindende Element oder
  - das betragsgrößte Element
  und vertauscht diese Zeile mit der Pivotzeile.
- Generell sucht man von Anfang an in der ersten Spalte eines jeden Teiltableaus das betragsgrößte Element und tauscht gegebenfalls die zugehörige Zeile mit der Diagonalzeile.

Dabei merkt man sich die Zeilenvertauschungen in einem Permutationsvektor $p = (p_1, p_2, ..., p_n)$, dessen Initialisierung $p = (1, 2, ..., n)$ ist.
Damit ergeben sich die Einträge in der orthogonalen Permutationsmatrix P.
p_i bedeutet in der i-ten Zeile und p_i-ten Spalte eine Eins, sonst Nullen in der Zeile.

Für $n = 5$ und $p = (4, 3, 1, 5, 2)$, ist das die Matrix

$$P = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Praktisch heißt das die Bildung der zeilenvertauschten Matrix $\tilde{A} = PA$ und ihre Faktorisierung $\tilde{A} = PA = -CB = LU$.
So kann man auch die Formel $A = P^T LU = \widetilde{L}U,\ \widetilde{L} = P^T L$ notieren.
Mit den angegebenen Permutationsmatrix P erhält man

$$P^T = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad \widetilde{L} = \begin{pmatrix} l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{11} & 0 & 0 & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \end{pmatrix}.$$

Der Leser sollte genau hinschauen, welche Form der Faktorisierung, ob $PA = LU$, d. h. $A = P^T LU$, bzw. $A = PLU$, d. h. $P^T A = LU$, in den Publikationen oder Softwarelösungen gemeint sind.

Beispiel 2.1 LU-Faktorisierung der symmetrischen und positiv definiten (spd) Hilbert-Matrix

$$A \;=\; A(3,3) \;=\; \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$$

Vorgehensweise

- Verwendung des Resttableau-Algorithmus.
- Die Pivotstrategie ist die Spaltenpivotisierung mit Zeilenvertauschung (Kolonnenmaximumstrategie), die hier eine Diagonalstrategie ist.
- Die Rechnungen sollen exakt mit Rationalarithmetik erfolgen.
- Die Matrix $A = A^{(0)}$ wird durch ihre Faktorisierungkomponenten L ($l_{ii} = 1$), U überschrieben.
- Nach zwei Schritten $A^{(k)},\ k = 1, .., n-1 = 2$, erhält man die transformierte Matrix in der Form $L\backslash U$.

$A^{(0)}$

1	$\frac{1}{2}$	$\frac{1}{3}$
$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$
$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$

$A^{(1)}$

1	$\frac{1}{2}$	$\frac{1}{3}$
$\frac{1}{2}$	**$\frac{1}{12}$**	$\frac{1}{12}$
$\frac{1}{3}$	$\frac{1}{12}$	$\frac{4}{45}$

$A^{(2)}$

1	$\frac{1}{2}$	$\frac{1}{3}$
$\frac{1}{2}$	$\frac{1}{12}$	$\frac{1}{12}$
$\frac{1}{3}$	1	$\frac{1}{180}$

Daraus erkennt man die Dreiecksmatrizen

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & 1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{1}{12} \\ 0 & 0 & \frac{1}{180} \end{pmatrix}.$$

Beim VGA ist das Endtableau $C\backslash B$ mit $A = -CB,\ C = -L,\ B = U$.

Die Hilbert-Matrix wird auch in die weiteren Betrachtungen mit einbezogen.

Wir wenden uns nun der *LU*-Faktorisierung mit den CAS Maple und Matlab zu.
Dabei soll vorausgeschickt werden, das diese Systeme bezüglich der dazu verwendeten Algorithmen und Strategien einem ständigen Wandel unterliegen. Das heißt nicht unbedingt, dass die neuere Version in jedem Fall besser ist, angefangen von der Verständlichkeit hin bis zu algorithmischen Aspekten und letztendlich bei der Lösungsdarstellung. Manchmal wird der Nutzer von der Implementation nicht nur in Erstaunen versetzt, sondern fast schockiert.
Einige Situationen, auch mit Vergleichen von "älteren" Lösungen mit Ergebnissen aktueller Versionen, illustrieren dies.

Rechnungen in Maple (Datei: *lu1.mws*)

Da hier Rechnungen mit verschieden GP-Formaten (Genauigkeiten) durchgeführt und die entsprechenden Ergebnisse angezeigt werden, soll vorab eine kurze Information über die Zahlendarstellung und die dabei erkennbaren Rundungsregime in der letzten Dezimale erfolgen.

Rechnen und Anzeigen der signifikanten Dezimalstellen bei GP-Zahlen

```
> # 2 signifikante Stellen
  Digits:=2: evalf(Pi/10^3);  # 0.0031
  Digits:=2: evalf(Pi/10^4);
  Digits:=2: evalf(Pi/10^5);
  Digits:=2: evalf(Pi/10^6);  # 0.31e-5

  # bei Dezimalzahlen vom Wert 0.0... mit mehr als 4 Anfangsnullen
  # nach dem Dezimalpunkt erfolgt ein Wechsel
  # von Festpunkt- auf GP-Darstellung
  Digits:=3: evalf(1/10^5);
  Digits:=3: evalf(1/10^6);

  # Beobachtung des Rundungseffekts in der letzten angezeigten Position
  Digits:=2:  evalf(Pi);     # 3.1415926535897932384...
  Digits:=10: evalf(Pi);
  Digits:=16: evalf(Pi);
  Digits:=2:  evalf(1.549);
  Digits:=2:  evalf(1.550);
```

$$0.0031$$
$$0.00031$$
$$0.000031$$
$$0.31\,10^{-5}$$
$$0.0000100$$
$$0.100\,10^{-5}$$
$$3.1$$
$$3.141592654$$
$$3.141592653589793$$
$$1.5$$
$$1.6$$

2.7.1 *LU*-Faktorisierung mit Maple

Dazu gibt es das Kommando `LUdecomp` im Paket `linalg`.
In der Online-Hilfe wird dazu u. a. folgendes beschrieben.

```
linalg[LUdecomp] - LU decomposition of a matrix
- The basic decomposition generates a square unit lower triangular L
  factor and an upper triangular U factor with the same dimensions as A
  so that A = L*U.
- For matrices of floating-point entries, a partial (row) pivoting method
  is used.
  For symbolic computation, pivoting is done only when a leading entry
  is zero. The permutation matrix is returned as P. Then A = P*L*U.
```

Dies ist eine relativ diffuse Erläuterung. Genauer ist:

```
- Symbolische Rechnung
  Falls alle PE nicht Null sind, dann keine Pivotstrategie und A=LU, P=I.
  Falls PE=0, wird in der Spalte das naechste NNE gesucht und die beiden
  Zeilen vertauscht, dabei Bildung der Permutationsmatrix P, so dass A=PLU.
- Numerische Rechnung
  Generell Anwendung der Spaltenpivotstrategie mit Suche des
  betragsgroessten Elements in der Spalte und dann gegebenenfalls
  Zeilenvertauschung.
```

Wir rechnen einige Beispiele.

(1) Beispiel

Diagonaldominante spd Matrix
LU-Faktorisierung ohne Pivotstrategie
Symbolische Rechnung

```
> A1:=evalm(hilbert(3)+diag(1$3));
```

$$A1 := \begin{bmatrix} 2 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{4}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{6}{5} \end{bmatrix}$$

Kurzform des Kommandos

```
> LUdecomp(A1);  # Ergebnis ist U
```

$$\begin{bmatrix} 2 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{29}{24} & \frac{1}{6} \\ 0 & 0 & \frac{2927}{2610} \end{bmatrix}$$

Eine Langform des Kommandos

```
> LUdecomp(A1,L='l',U='u',P='p'):  # Ergebnisse sind U,L,P
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p);
```

$$U := \begin{bmatrix} 2 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{29}{24} & \frac{1}{6} \\ 0 & 0 & \frac{2927}{2610} \end{bmatrix}$$

$$L := \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{6} & \frac{4}{29} & 1 \end{bmatrix}$$

$$P := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Numerische Rechnung

```
> A1[1,1]:=2.0:
> Digits:=10:

> LUdecomp(A1,L='l',U='u',P='p'):   # A1=PLU
  U:=evalm(u);   # U(2,3)=0.166666666666...
  L:=evalm(l);   # L(3,1)=0.166666666666...
  P:=evalm(p);
```

$$U := \begin{bmatrix} 2. & 0.5000000000 & 0.3333333333 \\ 0. & 1.208333333 & 0.1666666667 \\ 0. & 0. & 1.121455939 \end{bmatrix}$$

$$L := \begin{bmatrix} 1. & 0. & 0. \\ 0.2500000000 & 1. & 0. \\ 0.1666666666 & 0.1379310345 & 1. \end{bmatrix}$$

$$P := \begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 0. \\ 0. & 0. & 1. \end{bmatrix}$$

Man bemerke den Unterschied der Rechnungen/Rundungen bei den Matrixelementen U_{23} und L_{31}.

(2) Beispiel

spd Hilbert-Matrix
Symbolische Rechnung, *LU*-Faktorisierung ohne Pivotstrategie

```
> A2:=hilbert(3);
```

$$A2 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

Kurzform

```
> LUdecomp(A2); # Ergebnis ist U
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{1}{12} \\ 0 & 0 & \frac{1}{180} \end{bmatrix}$$

Langform

```
> LUdecomp(A2,L='l',U='u',P='p'):  # Ergebnisse sind U,L,P
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p):   # P=I
```

$$U := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{1}{12} \\ 0 & 0 & \frac{1}{180} \end{bmatrix}$$

$$L := \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & 1 & 1 \end{bmatrix}$$

Dreiecksmatrizen L,U mit GP-Zahlen

```
> Digits:=10:
  evalf(evalm(U));
  evalf(evalm(L));
```

$$\begin{bmatrix} 1. & 0.5000000000 & 0.3333333333 \\ 0. & 0.08333333333 & 0.08333333333 \\ 0. & 0. & 0.005555555556 \end{bmatrix}$$

$$\begin{bmatrix} 1. & 0. & 0. \\ 0.5000000000 & 1. & 0. \\ 0.3333333333 & 1. & 1. \end{bmatrix}$$

```
> # 1. Zwischenschritt der Faktorisierung, PE=1/12
  # PE=A2[2,2]<>0, nicht A2[3,2]
  `A(1)`=matrix(3,3,[[1,1/2,1/3],[0,1/12,1/12],[0,1/12,4/45]]);
```

$$A(1)= \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{1}{12} \\ 0 & \frac{1}{12} & \frac{4}{45} \end{bmatrix}$$

```
> # andere Faktorisierung mit Zeilenvertauschung 2<->3, PA=LU
  U1:=matrix(3,3,[[1,1/2,1/3],[0,1/12,4/45],[0,0,-1/180]]);
  evalf(evalm(U1));
  L1:=matrix(3,3,[[1,0,0],[1/3,1,0],[1/2,1,1]]);
  P1:=matrix(3,3,[[1,0,0],[0,0,1],[0,1,0]]);
  evalm(P1&*A2-L1&*U1);
```

$$U1 := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{4}{45} \\ 0 & 0 & \frac{-1}{180} \end{bmatrix}$$

$$\begin{bmatrix} 1. & 0.5000000000 & 0.3333333333 \\ 0. & 0.08333333333 & 0.08888888889 \\ 0. & 0. & -0.005555555556 \end{bmatrix}$$

$$L1 := \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{bmatrix}$$

$$P1 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Numerische Rechnung (GP-Zahlen), mit Pivotstrategie

Wenn wir den obigen ersten Zwischenschritt bei der LU-Faktorisierung mit dem Resttableau $A^{(1)}$ nehmen, so erkennen wir in der 2. Spalte an der Diagonalstelle und eine Position darunter die gleichen Werte $a_{22}^{(1)} = a_{32}^{(1)} = \frac{1}{12}$. Bei numerischen Rechnungen stehen dort die GP-Zahlen $\tilde{a}_{i,2}^{(1)}$ der Berechnungen für $A^{(1)}$, die bei Berücksichtigung von zusätzlichen Rundungen in der GP-Arithmetik nahe dem exakten Wert 0.083333333333333... liegen. Aber es kann dann natürlich passieren, dass $|\tilde{a}_{32}^{(1)}| > |\tilde{a}_{22}^{(1)}|$ ist, und somit eine Vertauschung von 2. und 3. Zeile stattfindet.
Damit gibt es ein abweichendes Verhalten von der symbolischen Rechnung und das Ergebnis ist die dann notierte Faktorisierung mit der Zeilenvertauschung $2 \leftrightarrow 3$.

An dieser Stelle machen wir Vergleichsrechnungen mit Maple V und
2 Varianten von Maple 8.

Der "alte" numerische Algorithmus in `LUdecomp` von Maple V wurde zwar in Maple 7/8 ausgetauscht (Ersetzung durch eine bessere FORTRAN-Routine), aber es ist schon eigenartig, dass in Maple 8 die Matrixdefinitionen `A:=hilbert(3):` und `A:=evalm(hilbert(3)):` zu unterschiedlicher Behandlung führen.

Maple V

```
> # A:=hilbert(3):     oder
  A:=matrix(3,3,[[1,  1/2,1/3],
                 [1/2,1/3,1/4],
                 [1/3,1/4,1/5]] );

> A[1,1]:=1.0:  # numerische Rechnung
> Digits:=10:

> LUdecomp(A,L='l',U='u',P='p');  # Ausgabe von U
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p);
```

Maple 8, D1

```
> # 1. Definition
  A:=hilbert(3):

> A[1,1]:=1.0:
> Digits:=10:

> LUdecomp(A,L='l',U='u',P='p');  # Ausgabe von U
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p);
```

Maple 8, D2

```
> # 2. Definition
  A:=evalm(hilbert(3)):
  # oder
  # A:=matrix(3,3,[[1,1/2,1/3],[1/2,1/3,1/4],[1/3,1/4,1/5]]);

> A[1,1]:=1.0:
> Digits:=10:

> LUdecomp(A,L='l',U='u',P='p');  # Ausgabe von U
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p);
```

	Maple V	Maple 8, D1
U	$\begin{bmatrix} 1.0 & .5000000000 & .3333333333 \\ 0 & .0833333333 & .0833333333 \\ 0 & 0 & .00555555560 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.5000000000 & 0.3333333333 \\ 0 & 0.0833333334 & 0.0888888889 \\ 0 & 0 & -0.00555555539 \end{bmatrix}$
	$\begin{bmatrix} 1.0 & .5000000000 & .3333333333 \\ 0 & .0833333333 & .0833333333 \\ 0 & 0 & .00555555560 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.5000000000 & 0.3333333333 \\ 0 & 0.0833333334 & 0.0888888889 \\ 0 & 0 & -0.00555555539 \end{bmatrix}$
L	$\begin{bmatrix} 1 & 0 & 0 \\ .5000000000 & 1 & 0 \\ .3333333333 & 1.000000000 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0.3333333333 & 1 & 0 \\ 0.5000000000 & 0.9999999988 & 1 \end{bmatrix}$
P	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$
		Maple 8, D2
U		$\begin{bmatrix} 1. & 0.500000000000000000 & 0.333333333299999979 \\ 0. & 0.0833333333500000107 & 0.0888888889111111304 \\ 0. & 0. & -0.00555555550777775410 \end{bmatrix}$
		$\begin{bmatrix} 1. & 0.5000000000 & 0.3333333333 \\ 0. & 0.08333333335 & 0.08888888891 \\ 0. & 0. & -0.005555555508 \end{bmatrix}$
L		$\begin{bmatrix} 1. & 0. & 0. \\ 0.3333333333 & 1. & 0. \\ 0.5000000000 & 0.9999999994 & 1. \end{bmatrix}$
P		$\begin{bmatrix} 1. & 0. & 0. \\ 0. & 0. & 1. \\ 0. & 1. & 0. \end{bmatrix}$

Tab. 2.8 Ergebnisse der 3 Maple-Versionen bez. LU-Faktorisierung in der GPA `Digits:=10:`

Bemerkungen

- Man sieht, dass sich Maple V und Maple 8 in der Pivotstrategie nach dem 1. Zwischenschritt anders verhalten. Wie ist das bei anderen Genauigkeiten?
- Erkennbar sind Unterschiede in den letzten Dezimalen der Elemente der Matrizen.
- Unverständlich ist die Ausgabe der oberen Dreiecksmatrix U bei `LUdecomp` in der 2. Matrixdefinition `A:=evalm(hilbert(3)):`.

- Warum stehen solche langen Dezimalzahlen, obwohl wegen der eingestellten Genauigkeit nur die ersten Stellen nach dem Punkt richtig sind und man die weiteren Stellen gar nicht verwerten kann?
- Zusätzlich stellt sich die Frage, wie die Darstellung bei anderen Genauigkeiten aussieht. Man sollte auf alles gefasst sein.

Wir bemühen uns, einiges davon aufzuklären.
Dazu wurden die numerischen Rechnungen in den drei Varianten mit verschiedenen Genauigkeiten `Digits:=1,2,3,4,...,9,10,11,...,14,15,16,...,20,25` durchgeführt sowie der Resttableau-Algorithmus für einige davon per Hand nachvollzogen. Für numerische Rechnungen sind natürlich nur gute Genauigkeiten relevant.

Digits	Maple V	Maple 8, D1	Maple 8, D2
1	ZV	-	ZV
2	-	ZV	ZV
3	-	ZV	ZV
4	-	ZV	ZV
10	-	ZV	ZV
16	-	ZV	ZV

Tab. 2.9 Ergebnisse bez. Zeilenvertauschung bei der LU-Faktorisierung in der GPA `Digits:=1,2,3,4,10,16:`, ZV = mit Zeilenvertauschung $2 \leftrightarrow 3$

Digits	$U(3,3)$	NKS	sDS	gDS
1	-0.0599999999999999424	19	18	0
2	-0.000741176470588272851	21	18	0
3	-0.00507740119760481668	20	18	1
4	-0.00507774001199741 70	20	18	2
9	-0.00555555507777780178	20	18	7
10	-0.00555555550777775410	20	18	8
11	-0.00555555555077776042	20	18	9
14	-0.00555555555555076186	20	18	12
15	-0.0055555555555539	16	14	12
16	-0.00555555555555539	17	15	13
20	-0.00555555555555555539	21	19	17
25	-0.00555555555555555555555539	26	24	22

Tab. 2.10 Maple 8, D2, bez. der Anzeige von $U(3,3)$ in `LUdecomp(A)` in der GPA `Digits:=1,2,3,4,...,9,10,11,...,14,15,16,...,20,25:`
exaktes Matrixelement ist $u_{33} = -\frac{1}{180} = -0.00555555...$
NKS = Nachkommastellen mit Sprung bei `Digits:=14,15`
sDS = signifikante Dezimalstellen (ab 1. Dezimale $\neq 0$ nach Komma)
gDS = gültige Dezimalstellen von sDS

Exakte Rechnung der LU-Faktorisierung mit Angabe des Permutationsvektors p
(o) ohne Zeilenvertauschung (m) mit Zeilenvertauschung

$\boxed{\mathbf{1}}$	$\frac{1}{2}$	$\frac{1}{3}$	$(1,2,3)$
$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	
$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	

$\boxed{\mathbf{1}}$	$\frac{1}{2}$	$\frac{1}{3}$	$(1,2,3)$
$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	
$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	

1	$\frac{1}{2}$	$\frac{1}{3}$	$(1,2,3)$
$\frac{1}{2}$	$\boxed{\mathbf{\frac{1}{12}}}$	$\frac{1}{12}$	
$\frac{1}{3}$	$\frac{1}{12}$	$\frac{4}{45}$	

1	$\frac{1}{2}$	$\frac{1}{3}$	
$\frac{1}{2}$	$\frac{1}{12}$	$\frac{1}{12}$	
$\frac{1}{3}$	$\boxed{\mathbf{\frac{1}{12}}}$	$\frac{4}{45}$	

1	$\frac{1}{2}$	$\frac{1}{3}$	
$\frac{1}{2}$	$\frac{1}{12}$	$\frac{1}{12}$	
$\frac{1}{3}$	1	$\frac{1}{180}$	

1	$\frac{1}{2}$	$\frac{1}{3}$	$(1,3,2)$
$\frac{1}{3}$	$\boxed{\mathbf{\frac{1}{12}}}$	$\frac{4}{45}$	
$\frac{1}{2}$	$\frac{1}{12}$	$\frac{1}{12}$	

1	$\frac{1}{2}$	$\frac{1}{3}$	
$\frac{1}{3}$	$\frac{1}{12}$	$\frac{4}{45}$	
$\frac{1}{2}$	1	$\frac{-1}{180}$	

$$A = L_o U_o,$$

$$L_o = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & 1 & 1 \end{pmatrix}, \quad U_o = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{1}{12} \\ 0 & 0 & \frac{1}{180} \end{pmatrix}, \quad P_o = I,$$

$$P_m A = L_m U_m, \quad A = P_m^T L_m U_m = \widetilde{L}_m U_m,$$

$$L_m = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{pmatrix}, \quad U_m = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{4}{45} \\ 0 & 0 & \frac{-1}{180} \end{pmatrix}, \quad P_m = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

$$\widetilde{L}_m = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 1 \\ \frac{1}{3} & 1 & 0 \end{pmatrix}.$$

Numerische Rechnungen

Resttableau-Algorithmus `Digits:=1;`

Die Notationen der Matrixelemente sind entsprechend der Ergebnisdarstellung in den einzelnen Maple-Versionen.

Maple V mit ZV

1.	.5	.3	(1, 2, 3)
.5	.3	.2	
.3	.2	.2	

1.	.5	.3	
.5	0	.1	
.3	**.1**	.1	

1.	.5	.3	(1, 3, 2)
.3	**.1**	.1	
.5	0	.1	

1.	.5	.3	
.3	.1	.1	
.5	0	.1	

Maple 8, D1 ohne ZV

1.	0.5	0.3	(1, 2, 3)
0.5	0.3	0.2	
0.3	0.2	0.2	

1.	0.5	0.3	(1, 2, 3)
0.5	**0.1**	0	
0.3	0	0.1	

1.	0.5	0.3	
0.5	0.1	0	
0.3	0	0.1	

Maple 8, D2 mit ZV

1.	0.5	0.3	(1, 2, 3)
0.5	0.3	0.2	
0.3	0.2	0.2	

1.	0.5	0.3	
0.5	0.05	0.05	
0.3	**0.05**	0.1	

1.	0.5	0.3	(1, 3, 2)
0.3	**0.05**	0.1	
0.5	0.05	0.05	

1.	0.5	0.3	
0.3	0.05	0.1	
0.5	1.	-0.06	

Zwischenrechnungen für $A^{(1)}$ (spaltenweise)

$0.3 - 0.5 \cdot 0.5 \Rightarrow 0$	$0.3 - 0.5 \cdot 0.5 \Rightarrow 0.1$	$0.3 - 0.5 \cdot 0.5 \Rightarrow 0.05$
$0.2 - 0.3 \cdot 0.5 \Rightarrow 0.1$	$0.2 - 0.3 \cdot 0.5 \Rightarrow 0$	$0.2 - 0.3 \cdot 0.5 \Rightarrow 0.05$
$0.2 - 0.5 \cdot 0.3 \Rightarrow 0.1$	$0.2 - 0.5 \cdot 0.3 \Rightarrow 0$	$0.2 - 0.5 \cdot 0.3 \Rightarrow 0.05$
$0.2 - 0.3 \cdot 0.3 \Rightarrow 0.1$	$0.2 - 0.3 \cdot 0.3 \Rightarrow 0.1$	$0.2 - 0.3 \cdot 0.3 \Rightarrow 0.1$

Die Zeilenvertauschung ist bei den einzelnen Versionen nicht das eigentliche Problem, auch wenn sie von der Computerrechnung dem Nutzer einfach aufgezwungen wird.

Auffälliger ist, welche Auswirkung die Genauigkeitseinstellung `Digits:=1:` hat. Während diese bei den Versionen Maple V und Maple 8, D1, dazu führt, dass auch alle Zwischenrechnungen jeweils auf eine gültige Stelle reduziert/gerundet werden, scheint es so, dass bei Maple 8, D2, in den Rechnungen eine zusätzliche Stelle mitgeführt wird und das letzte Ergebnis dann mit **einer** signifikanten Stelle erscheint. Aber in der Matrix U sind trotzdem noch Abweichungen zu erkennen.

Weiterhin beachte man, dass bei Maple 8, D2, das Kommando `LUdecomp(A)` die unnötig lange Darstellung

$$\begin{bmatrix} 1. & 0.500000000000000000 & 0.299999999999999988 \\ 0. & 0.050000000000000168 & 0.110000000000000014 \\ 0. & 0. & -0.059999999999999424 \end{bmatrix}$$

liefert.

Resttableau-Algorithmus `Digits:=2`

Maple V ohne ZV

1.0	.50	.33	(1,2,3)
.50	.33	.25	
.33	.25	.20	
1.0	.50	.33	(1,2,3)
.50	**.08**	0.08	
.33	.08	.09	
1.0	.50	.33	
.50	.08	.08	
.33	1.0	.010	

Maple 8, D1 mit ZV

1.0	0.50	0.33	(1,2,3)
0.50	0.33	0.25	
0.33	0.25	0.20	
1.0	0.50	0.33	
0.50	0.08	0.09	
0.33	**0.09**	0.09	
1.0	0.50	0.33	(1,3,2)
0.33	**0.09**	0.09	
0.50	0.08	0.09	
1.0	0.50	0.33	
0.33	0.09	0.09	
0.50	0.89	0.010	

Maple 8, D2 mit ZV

1.	0.50	0.33
0.50	0.33	0.25
0.33	0.25	0.20
1.	0.50	0.33
0.50	0.080	0.085
0.33	**0.085**	0.091
1.	0.50	0.33
0.33	**0.085**	0.091
0.50	0.080	0.085
1.	0.50	0.33
0.33	0.085	0.091
0.50	0.94	-0.00074

Zwischenrechnungen für $A^{(1)}$

$0.33 - 0.50 \cdot 0.50 \Rightarrow 0.08$	$0.33 - 0.50 \cdot 0.50 \Rightarrow 0.08$	$0.33 - 0.50 \cdot 0.50 \Rightarrow 0.080$
$0.25 - 0.33 \cdot 0.50 \Rightarrow 0.08$	$0.25 - 0.33 \cdot 0.50 \Rightarrow 0.09$	$0.25 - 0.33 \cdot 0.50 \Rightarrow 0.085$
$0.25 - 0.50 \cdot 0.33 \Rightarrow 0.08$	$0.25 - 0.50 \cdot 0.33 \Rightarrow 0.09$	$0.25 - 0.50 \cdot 0.33 \Rightarrow 0.085$
$0.20 - 0.33 \cdot 0.33 \Rightarrow 0.09$	$0.20 - 0.33 \cdot 0.33 \Rightarrow 0.09$	$0.20 - 0.33 \cdot 0.33 \Rightarrow 0.091$

Resttableau-Algorithmus `Digits:=3`

Maple V ohne ZV				Maple 8, D1 mit ZV				Maple 8, D2 mit ZV			
1.0	.500	.333	(1,2,3)	**1.0**	0.500	0.333	(1,2,3)	**1.**	0.500	0.333	
0.500	.333	.250		0.500	0.333	0.250		0.500	0.330	0.250	
0.333	.250	.200		0.333	0.250	0.200		0.333	0.250	0.200	
1.0	.500	.333	(1,2,3)	1.0	0.500	0.333		1.	0.500	0.333	
.500	**.083**	.083		0.500	0.083	0.084		0.500	0.0830	0.0835	
.333	.083	.089		0.333	**0.084**	0.089		0.333	**0.0835**	0.0891	
1.0	.500	.333		1.0	0.500	0.333	(1,3,2)	1.	0.500	0.333	
.500	.083	.083		0.333	**0.084**	0.089		0.333	**0.0835**	0.0891	
.333	1.00	.0060		0.500	0.083	0.084		0.500	0.0830	0.0835	
				1.0	0.500	0.333		1.	0.500	0.333	
				0.333	0.084	0.089		0.333	0.0835	0.0891	
				0.500	0.988	-0.0039		0.500	0.994	-0.00508	

Zwischenrechnungen für $A^{(1)}$

$0.333 - 0.500 \cdot 0.500 \Rightarrow 0.083$	$0.333 - 0.500 \cdot 0.500 \Rightarrow 0.083$	$0.333 - 0.500 \cdot 0.500 \Rightarrow 0.0830$
$0.250 - 0.333 \cdot 0.500 \Rightarrow 0.083$	$0.250 - 0.333 \cdot 0.500 \Rightarrow 0.084$	$0.250 - 0.333 \cdot 0.500 \Rightarrow 0.0835$
$0.250 - 0.500 \cdot 0.333 \Rightarrow 0.083$	$0.250 - 0.500 \cdot 0.333 \Rightarrow 0.084$	$0.250 - 0.500 \cdot 0.333 \Rightarrow 0.0835$
$0.200 - 0.333 \cdot 0.333 \Rightarrow 0.089$	$0.200 - 0.333 \cdot 0.333 \Rightarrow 0.089$	$0.200 - 0.333 \cdot 0.333 \Rightarrow 0.0891$

LU-Faktorisierung `Digits:=4`

Maple V ohne ZV	Maple 8, D1 mit ZV	Maple 8, D2 mit ZV
U $\begin{bmatrix} 1.0 & .5000 & .3333 \\ 0 & .0833 & .0833 \\ 0 & 0 & .00560 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.5000 & 0.3333 \\ 0 & 0.0834 & 0.0889 \\ 0 & 0 & -0.00539 \end{bmatrix}$	$\begin{bmatrix} 1. & 0.5000 & 0.3333 \\ 0. & 0.08335 & 0.08891 \\ 0. & 0. & -0.005508 \end{bmatrix}$
L $\begin{bmatrix} 1 & 0 & 0 \\ .5000 & 1 & 0 \\ .3333 & 1.000 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0.3333 & 1 & 0 \\ 0.5000 & 0.9988 & 1 \end{bmatrix}$	$\begin{bmatrix} 1. & 0. & 0. \\ 0.3333 & 1. & 0. \\ 0.5000 & 0.9994 & 1. \end{bmatrix}$
p (1,2,3)	(1,3,2)	(1,3,2)

Entscheidend sind die Werte der Matrix $A^{(1)} = (a_{ij}^{(1)})$ des 1. Zwischenschritts, insbesondere die Matrixelemente $a_{22}^{(1)}$ und $a_{32}^{(1)}$, für die Zeilenvertauschung und den nächsten Schritt. Die Art und Weise der Berechnung dieser Größen bei den verschiedenen Genauigkeiten fassen wir noch einmal tabellarisch zusammen.

Exakt		$a_{22}^{(1)} = a_{22}^{(0)} - a_{21}^{(1)} a_{12}^{(1)} = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}$ $= 0.3333... - 0.25 = 0.8333...$	$a_{32}^{(1)} = a_{32}^{(0)} - a_{31}^{(1)} a_{12}^{(1)} = \frac{1}{4} - \frac{1}{6} = \frac{1}{12}$ $= 0.25 - 0.1666... = 0.8333...$
Digits=1			
Maple V	ZV	... $\Rightarrow 0$	$< 0.1 \Leftarrow$ ...
Maple 8, D1		0.3–0.5·0.5 $\Rightarrow 0.1$	$> 0 \Leftarrow$ 0.2–0.3·0.5
Maple 8, D2	ZV	... $\Rightarrow 0.05$	$< 0.05 \Leftarrow$ ...
Digits=2			
Maple V		... $\Rightarrow 0.08$	$> 0.08 \Leftarrow$ ...
Maple 8, D1	ZV	0.33–0.50·0.5 $\Rightarrow 0.08$	$< 0.09 \Leftarrow$ 0.25–0.33·0.5
Maple 8, D2	ZV	... $\Rightarrow 0.080$	$< 0.085 \Leftarrow$ ...
Digits=3			
Maple V		... $\Rightarrow 0.083$	$> 0.083 \Leftarrow$ ...
Maple 8, D1	ZV	0.333–0.5·0.5 $\Rightarrow 0.083$	$< 0.084 \Leftarrow$ 0.25–0.333·0.5
Maple 8, D2	ZV	... $\Rightarrow 0.0830$	$< 0.0835 \Leftarrow$ ...
Digits=4			
Maple V		... $\Rightarrow 0.0833$	$> 0.0833 \Leftarrow$ ...
Maple 8, D1	ZV	0.3333–0.5·0.5 $\Rightarrow 0.0833$	$< 0.0834 \Leftarrow$ 0.25–0.3333·0.5
Maple 8, D2	ZV	... $\Rightarrow 0.08333$	$< 0.08335 \Leftarrow$ ...
Digits=10			
Maple V		... $\Rightarrow 0.0833333333$	$> 0.0833333333 \Leftarrow$ ...
Maple 8, D1	ZV	0.33..3–0.5·0.5 $\Rightarrow 0.0833333333$	$< 0.0833333334 \Leftarrow$ 0.25–0.33..3·0.5
Maple 8, D2	ZV	... $\Rightarrow 0.08333333333$	$< 0.08333333335 \Leftarrow$ ...

Tab. 2.11 Matrixelemente $a_{22}^{(1)}$ und $a_{32}^{(1)}$ des 1. Zwischenschritts für ZV

(3) Beispiel

Symmetrische Matrix
Symbolische Rechnung, *LU*-Faktorisierung mit Zeilenvertauschung $2 \leftrightarrow 3$
im 2. Schritt wegen PE=0

```
> A3:=matrix(3,3,[[1,2,0],
                  [2,4,1],
                  [0,1,2]]:
```

$$A3 := \begin{bmatrix} 1 & 2 & 0 \\ 2 & 4 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

```
> LUdecomp(A3,L='l',U='u',P='p'):
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p):   # p=(1,3,2), A=PLU
```

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$U := \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$L := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

Numerische Rechnung, Spaltenpivotstrategie von Beginn an
1. Schritt: Zeilenvertauschung $1 \leftrightarrow 2$
2. Schritt: Zeilenvertauschung $2 \leftrightarrow 3$

```
> A3[1,1]:=1.0:
> Digits:=10:

> LUdecomp(A3,L='l',U='u',P='p');
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p):   # p=(3,1,2), A=PLU
```

$$\begin{bmatrix} 2. & 4. & 1. \\ 0. & 1. & 2. \\ 0. & 0. & -0.500000000000000000 \end{bmatrix}$$

$$U := \begin{bmatrix} 2. & 4. & 1. \\ 0. & 1. & 2. \\ 0. & 0. & -0.5000000000 \end{bmatrix}$$

$$L := \begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 0. \\ 0.5000000000 & 0. & 1. \end{bmatrix}$$

$$P := \begin{bmatrix} 0. & 0. & 1. \\ 1. & 0. & 0. \\ 0. & 1. & 0. \end{bmatrix}, \quad P^T = \begin{bmatrix} 0. & 1. & 0. \\ 0. & 0. & 1. \\ 1. & 0. & 0. \end{bmatrix}, \quad P^T A = LU$$

(4) Beispiel

Symmetrische Matrix
Symbolische Rechnung, LU-Faktorisierung mit Zeilenvertauschung $1 \leftrightarrow 2$
im 1. Schritt wegen PE=0

```
> A4:=matrix(3,3,[[0,1,2],
                  [1,2,3],
                  [2,3,5]]:

> LUdecomp(A4,L='l',U='u',P='p'):
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p):   # p=(2,1,3), A=PLU
```

$$U := \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$L := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & -1 & 1 \end{bmatrix}$$

Numerische Rechnung, Spaltenpivotstrategie von Beginn an
1. Schritt: Zeilenvertauschung $1 \leftrightarrow 3$
2. Schritt: Zeilenvertauschung $2 \leftrightarrow 3$

```
> A4[1,1]:=0.0:
> Digits:=10:

> LUdecomp(A4,L='l',U='u',P='p');
  U:=evalm(u);
  L:=evalm(l);
  P:=evalm(p):    # p=(2,3,1), A=PLU
```

$$\begin{bmatrix} 2. & 3. & 5. \\ 0. & 1. & 2. \\ 0. & 0. & -0.500000000000000000 \end{bmatrix}$$

$$U := \begin{bmatrix} 2. & 3. & 5. \\ 0. & 1. & 2. \\ 0. & 0. & -0.5000000000 \end{bmatrix}$$

$$L := \begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 0. \\ 0.5000000000 & 0.5000000000 & 1. \end{bmatrix}$$

2.7.2 LU-Faktorisierung mit Matlab

Die LU-Faktorisierung ohne Zeilenvertauschung liefert $A = LU$ mit den entsprechenden Dreiecksmatrizen L, $l_{ii} = 1$, und U. Zeilenvertauschungen werden entweder in die untere Dreiecksmatrix L einbezogen (psychologically lower triangular matrix) oder als Permutationsmatrix in Form eines zusätzlichen Ergebnisparameters angegeben.

`[L,U] = lu(A)`	$A = LU$,	in L stehen die untere Dreiecksmatrix und Permutationsmatrix, U obere Dreiecksmatrix
`[L,U,P] = lu(A)`	$PA = LU$,	L, U untere bzw. obere Dreiecksmatrix, P Permutationsmatrix

Wir erinnern uns, die Faktorisierung mit dem Maple-Kommando `LUdecomp` ergab bei Pivotisierung $A = PLU$.

`lu(A)` alleine ergibt ein Tableau (eine Ergebnismatrix) $A1$ mit 2 Dreiecksmatrizen.

Werden keine Zeilenvertauschungen vorgenommen, so gilt folgendes.
In der älteren Version Matlab 4.2 hat $A1$ die Form $C\backslash B$, welche beim verketteten Gauß-Algorithmus entsteht, so dass $A = -CB = LU$ ist (output from LINPACK'S ZGEFA routine).
Es ist dann

```
U = triu(lu(A))
L = eye(n)-tril(lu(A),-1)  % n ist Dimension
```

In den neuen Versionen Matlab 6.$*$ erhält man eine Tableau mit L und U (output from LAPACK'S DGETRF or ZGETRF routine), so dass

```
U = triu(lu(A))
L = eye(n)+tril(lu(A),-1)
```

folgen.

Aber die LU-Faktorisierung verwendet generell eine Pivotstrategie, und zwar die Spaltenpivotisierung mit Zeilenvertauschung. Dabei wird in der jeweiligen Spalte nach dem betragsgrößten Element gesucht.
Die Zeilenvertauschung ist aus der Kurzform `lu(A)` nicht ohne Weiteres erkennbar. Das obere Dreieck von $A1$ mit Diagonale bildet die Matrix U, die Zeilen von L ergeben sich aus den Zeilen unterhalb der Diagonalen von $A1$, aber eventuell in getauschter Reihenfolge, sowie mit $l_{ii} = 1$. Deshalb ist die Verwendung der Kommandos `[L,U] = lu(A)` oder `[L,U,P] = lu(A)` sinnvoll.

Rechnungen in Matlab (Datei: *luf1.m*)
(1) Nehmen wir zunächst den einfachen Fall einer spd streng diagonaldominanten Matrix.

```
A1 = [ 10   -1    2    0
       -1   11   -1    3
        2   -1   10   -1
        0    3   -1    8 ];
```

```
% Matlab 4.2
% Tableau wie beim verketteten Gaussalgorithmus, A=-CB
lu(A1)
ans =
     10.0000   -1.0000    2.0000         0
      0.1000   10.9000   -0.8000    3.0000
     -0.2000    0.0734    9.5413   -0.7798
           0   -0.2752    0.0817    7.1106

% Matlab 6.*
% Tableau wie beim Resttableau-Algorithmus, A=LU
lu(A1)
ans =
     10.0000   -1.0000    2.0000         0
     -0.1000   10.9000   -0.8000    3.0000
      0.2000   -0.0734    9.5413   -0.7798
           0    0.2752   -0.0817    7.1106


[L U P] = lu(A1)
L =
      1.0000         0         0         0
     -0.1000    1.0000         0         0
      0.2000   -0.0734    1.0000         0
           0    0.2752   -0.0817    1.0000

U =
     10.0000   -1.0000    2.0000         0
           0   10.9000   -0.8000    3.0000
           0         0    9.5413   -0.7798
           0         0         0    7.1106

P =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

[L U] = lu(A1)  % analog ohne Ausgabe von P
```

Natürlich kann man versuchen, die Faktorisierung ohne Zeilenvertauschung durchzuführen. Falls dabei ein PE Null wird, bricht man einfach ab. Man gibt des letzte Tableau und die Anzahl der durchgeführten Schritte aus.

Die folgende Prozedur (Datei: *lu1.m*) geht so vor, wobei die Matrix A allgemein eine Rechteckmatrix sein kann.

```
% function [B,k] = lu1(A)
% lu1 returns the transformed rectangular matrix
% as upper triangular matrix U an lower triangular
% matrix L in one table B (without row exchanges,
% without pivot strategy), so far if it is possible.
% k is the number of performed stages. If k=min(n,m),
% then the minor A(1:k,1:k) is regular.
% Use for solving Ax=b or transformation of A.
%
[n m] = size(A);  % row and column number of A
B = A;
k = 0;
r = 1;  % row index
c = 1;  % column index

while (c<=m)&(r<=n)
 if (B(r,r)==0)
  c=m+1; r=n+1;
 else
  % transform rest table
  k = k+1;
  for i = r+1:n
   if (B(i,c)~=0)
    t = B(i,c)/B(r,c);
    for j = c+1:m, B(i,j) = B(i,j)-t*B(r,j); end;
    B(i,c) = t;
   end;
  end;
  r = r+1;
  c = c+1;
 end;
end;
% end of function lu1
```

(2) Betrachten wir eine reguläre Matrix, die man mit der Funktion `lu1` ohne Pivotisierung LU-faktorisieren kann.

```
A2 = [ 1  2  3
       2  3  4
       2  5  7 ];
```

```
[B,k] = lu1(A2)
B =
     1     2     3
     2    -1    -2
     2    -1    -1

k =
   3

U = triu(B)
U =
     1     2     3
     0    -1    -2
     0     0    -1

L = eye(3)+tril(B,-1)
L =
     1     0     0
     2     1     0
     2    -1     1
```

Nun verwenden wir das `lu`-Kommando. In der Kurzform erhält man

```
format
lu(A2)
ans =
    2.0000    3.0000    4.0000
    1.0000    2.0000    3.0000
    0.5000    0.2500    0.2500
```

Die obere Dreiecksmatrix U kann man ablesen, und man erkennt, dass wegen $(u_{11}, u_{12}, u_{13}) \neq (a2_{11}, a2_{12}, a2_{13})$ mindestens eine Zeilenvertauschung stattgefunden hat.
Die Zeilen von L entstehen aus den Vektoren

(1,0,0), (1.0000,1,0), (0.5000,0.2500,1),

aber nicht zwingend in dieser Reihenfolge.
Es gilt nicht $A2 = \bar{L}U$ mit

$$A2 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 2 & 5 & 7 \end{pmatrix}, \quad \bar{L}U = \begin{pmatrix} 1 & 0 & 0 \\ 1.0000 & 1 & 0 \\ 0.5000 & 0.2500 & 1 \end{pmatrix} \begin{pmatrix} 2.0000 & 3.0000 & 4.0000 \\ 0 & 2.0000 & 3.0000 \\ 0 & 0 & 0.2500 \end{pmatrix}.$$

Nehmen wir also Langformen des `lu`-Kommandos bei verschiedenen Formaten der Ergebnisdarstellung.

```
format
[L U P] = lu(A2)   % PA2=LU
L =
    1.0000         0         0
    1.0000    1.0000         0
    0.5000    0.2500    1.0000
U =
    2.0000    3.0000    4.0000
         0    2.0000    3.0000
         0         0    0.2500

P =
     0     1     0
     0     0     1
     1     0     0

P*A2-L*U
ans =
     0     0     0
     0     0     0
     0     0     0

format rat   % display rational approximations
lu(A2)
ans =
      2      3      4
      1      2      3
    1/2    1/4    1/4

[L U P] = lu(A2)   % PA2=LU
L =
      1      0      0
      1      1      0
    1/2    1/4      1
U =
      2      3      4
      0      2      3
      0      0    1/4

P =
      0      1      0
      0      0      1
      1      0      0
```

```
[Ls U] = lu(A2)   % A2=Ls*U=P'LU <-> PA2=LU
Ls =
     1/2    1/4     1
      1      0      0
      1      1      0
U =
      2      3      4
      0      2      3
      0      0     1/4
A2-Ls*U
ans =
      0      0      0
      0      0      0
      0      0      0
P*Ls-L
ans =
      0      0      0
      0      0      0
      0      0      0
```

(3) Das dritte Beispiel sei die spd Hilbert-Matrix $H(3)$.

```
A3 = [ 1.0000    0.5000    0.3333
       0.5000    0.3333    0.2500
       0.3333    0.2500    0.2000 ];
A3 = hilb(3);
[B,k] = lu1(A3)   % LU-F. ohne Zeilenvertauschung
B =
    1.0000    0.5000    0.3333
    0.5000    0.0833    0.0833
    0.3333    1.0000    0.0056
k =
     3

U = triu(B)
U =
    1.0000    0.5000    0.3333
         0    0.0833    0.0833
         0         0    0.0056
L = eye(3)+tril(B,-1)
L =
    1.0000         0         0
    0.5000    1.0000         0
    0.3333    1.0000    1.0000
```

Die symbolische Rechnung und Faktorisierung in Maple läuft auch ohne Zeilenvertauschung ab und ergibt die erwartete Lösung.

```
A3 = [1    0  0                  # Maple
      1/2  1  0
      1/3  1  1]*[1  1/2   1/3
                     0  1/12  1/12
                     0  0     1/180]
```

Das Matlab-Kommando `lu` hingegen führt im 2. Schritt eine Vertauschung von 2. und 3. Zeile durch.

```
% Matlab 4.2
% Tableau wie beim verketteten Gauss-Algorithmus, A3=-CB

format rat      % display rational approximations
lu(A3)
ans =
       1      1/2      1/3
    -1/2      1/12     4/45
    -1/3      -1      -1/180
```

Daraus ergibt sich die obere Dreiecksmatrix

$$U = \begin{pmatrix} 1 & 1/2 & 1/3 \\ 0 & 1/12 & 4/45 \\ 0 & 0 & -1/180 \end{pmatrix}.$$

Aber bez. der unteren Dreiecksmatrix ist daraus keine verwertbare Information zu entnehmen, denn die möglichen Zeilen für L würden $(1,0,0),\ (1/2,1,0),\ (1/3,1,1)$ sein, und das ist falsch.
Matlab 6.$*$ hat diese irritierende Darstellung durch den Resttableau-Algorithmus korrigiert.

```
% Matlab 6.*
% Tableau wie beim Resttableau-Algorithmus, A3=LU
format rat
lu(A3)
ans =
       1      1/2      1/3
      1/3     1/12     4/45
      1/2      1      -1/180
```

So sind zumindest die Zeilen der Matrix L als $(1,0,0),\ (1/3,1,0),\ (1/2,1,1)$ ablesbar, auch wenn wegen Zeilenvertauschungen ihre Reihenfolge noch nicht klar ist.

Die Pivotstrategie bewirkt, dass im 2. Schritt beim Vergleich von 1/12=0.08333... in der 2. Zeile mit 1/12 darunter die 3. Zeile bevorzugt wird.

Langformen zur Kontrolle mit der Permutationsmatrix

```
[L,U,P] = lu(A3)
L =
      1      0      0
     1/3     1      0
     1/2     1      1
U =
      1     1/2    1/3
      0     1/12   4/45
      0      0    -1/180
P =
      1      0      0
      0      0      1
      0      1      0
P*A3-L*U;  % 0

[Ls,U] = lu(A3)  % Zeilenvertauschung in L zu erkennen
Ls =
      1      0      0
     1/2     1      1
     1/3     1      0
U =
      1     1/2    1/3
      0     1/12   4/45
      0      0    -1/180
```

Um die Zeilenvertauschung bei der Faktorisierung in Matlab zu unterbinden, vergrößern wir das Element $A3(2,2)$ etwas gemäß $A3(2,2) = A3(2,2) + 1e-16$.
Wir erhalten dann ein mit der symbolischen Rechnung in Maple vergleichbares Ergebnis.

```
A3(2,2) = A3(2,2)+1e-16;
format rat
lu(A3)
ans =
      1     1/2    1/3
     1/2    1/12   1/12
     1/3     1     1/180
```

2.8 Orthogonalisierung und QR-Faktorisierung

Die Faktorisierung einer (allgemein rechteckigen) Matrix in eine orthogonale Matrix Q mit $QQ^T = Q^TQ = I$ und obere Dreiecksmatrix R der Form $A = QR$ ist eine grundlegende Aufgabe der linearen Algebra. Sie wird im Zusammenhang mit der Lösung von LGS und linearer Ausgleichsprobleme sowie mit einigen Verfahren zum Eigenwertproblem (EWP) benötigt.
Die QR-Faktorisierung ist numerisch günstiger als die LU-Faktorisierung des Gaußschen Eliminationsverfahrens. Ihre Überlegenheit bezüglich der numerischen Stabilität hängen eng mit den spezifischen Eigenschaften orthogonaler Matrizen zusammen. So lässt die Multiplikation mit einer Orthogonalmatrix in Verbindung mit dem Skalarprodukt die (euklidische) Länge von Vektoren und somit die Winkel zwischen Vektoren invariant.
Zudem gilt, dass man bei gegebener QR-Faktorisierung von A die Lösung des LGS $Ax = b$ auf die Lösung eines gestaffelten Systems zurückführen kann. Es gilt nämlich

$$Ax = b \quad \Rightarrow \quad Rx = Q^Tb.$$

Es gibt drei grundlegende Methoden (siehe [28], [32]).
Mit der QR-Faktorisierung der Matrix ist die Gram-Schmidt-Orthogonalisierung ihrer Spalten (Spaltenvektoren) verbunden. Für die QR-Faktorisierung bieten sich aber auch Rotationen und Reflexionen an.
Wir beschränken uns auf den Fall quadratischer Matrizen.

Zur Erläuterung von einigen Sachverhalten notieren wir die QR-Faktorisierung in der Form mit Spaltenvektoren

$$A = \begin{pmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{pmatrix} = \begin{pmatrix} | & | & & | \\ q_1 & q_2 & \dots & q_n \\ | & | & & | \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ & r_{22} & r_{23} & \dots & r_{2n} \\ & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ 0 & & & & r_{nn} \end{pmatrix} = QR.$$

Bezüglich der Vorzeichen der Spaltenvektoren q_j der Orthogonalmatrix Q hat man gewisse Freiheiten. Würde man bei gegebener Faktorisierung $A = QR$ nun an Stelle von q_j den Vektor $-q_j$ setzen, dann hätte dies folgende Auswirkung auf die j-te Zeile von R.

$$\begin{aligned} a_k &= \sum_{i=1}^{k} r_{ik}q_i, \quad k = j, j+1, ..., n, \\ &= r_{1,k}q_1 + r_{2,k}q_2 + ... + r_{j-1,k}q_{j-1} + (-r_{jk})(-q_j) + r_{j+1,k}q_{j+1} + ... + r_{kk}q_k, \end{aligned}$$

d. h. die Matrixelemente $r_{jk},\ k = j, j+1, ..., n$, würden ihr Vorzeichen ändern. Damit kann man das Vorzeichen von q_j auch davon abhängig machen, um das Diagonalelement r_{kk} positiv zu skalieren.

In den verschiedenen Verfahren, auf die wir später eingehen werden, stellt sich die Vorzeichensituation unterschiedlich dar. Man beachte weiterhin, dass gewisse Strukturen der Ausgangsmatrix A sich auf die Faktorisierungskomponenten übertragen.

Zunächst verwenden wir entsprechende Kommandos in den CAS Maple und Matlab. Folgende Beispiele sind unter Verwendung der Maple-Kommandos

```
R:=QRdecomp(A,Q='q',rank='r');      # R:=QRdecomp(A,Q='q');
Q:=evalm(q);
```

sowie Matlab-Befehls

```
[Q,R]=qr(A)
```

gerechnet worden.

Rechnungen in Maple (Datei: *qrf1.mws*)

Bei der QR-Faktorisierung mit `QRdecomp` werden in der symbolischen Version die Diagonalelemente von R positiv skaliert.

Man beachte dabei den Hinweis in Maple:

For symbolic computation, the Gram-Schmidt process is applied. For matricies of floating point entries, the numerically stable Householder-transformations are used.

Bei vielen Schritten (große Dimension) werden die symbolischen Rechnungen immer aufwendiger und "länger", damit langsamer. Die Householder-Transformation ist geringfügig genauer als die Gram-Schmidt-Orthogonalisierung.

Beispiel 2.2 Matrix A quadratisch

(1) Symmetrische Matrix

```
> n:=4:
  A:=matrix(n,n,[[ 1,-1, 1, 2],
                 [-1, 2, 0, 1],
                 [ 1, 0,-1, 1],
                 [ 2, 1, 1, 0]]):

> # For symbolic computation the low speed Gram-Schmidt process is applied.
  R:=QRdecomp(A,Q='q',rank='r');
  r;
  Q:=evalm(q);
  evalm(A-Q&*R):   # Nullmatrix
```

$$R := \begin{bmatrix} \sqrt{7} & -\frac{\sqrt{7}}{7} & \frac{2\sqrt{7}}{7} & \frac{2\sqrt{7}}{7} \\ 0 & \frac{\sqrt{287}}{7} & \frac{2\sqrt{287}}{287} & \frac{2\sqrt{287}}{287} \\ 0 & 0 & \frac{3\sqrt{451}}{41} & \frac{17\sqrt{451}}{1353} \\ 0 & 0 & 0 & \frac{23\sqrt{11}}{33} \end{bmatrix}$$

$$Q := \begin{bmatrix} \sqrt{7} & -\frac{6\sqrt{287}}{287} & \frac{31\sqrt{451}}{1353} & \frac{7\sqrt{11}}{33} \\ -\frac{\sqrt{7}}{7} & \frac{13\sqrt{287}}{287} & \frac{8\sqrt{451}}{1353} & \frac{5\sqrt{11}}{33} \\ \frac{\sqrt{7}}{7} & \frac{\sqrt{287}}{287} & -\frac{53\sqrt{451}}{1353} & \frac{4\sqrt{11}}{33} \\ \frac{2\sqrt{7}}{7} & \frac{9\sqrt{287}}{287} & \frac{5\sqrt{451}}{1353} & -\frac{\sqrt{11}}{11} \end{bmatrix}$$

```
> Digits:=16:
  evalf(evalm(R));
  evalf(evalm(Q));
```

$$\begin{bmatrix} 2.645751311064591 & -0.3779644730092274 & 0.7559289460184545 & 0.7559289460184545 \\ 0. & 2.420153478013918 & 0.1180562672201911 & 0.1180562672201911 \\ 0. & 0. & 1.553909310848437 & 0.2668329119638730 \\ 0. & 0. & 0. & 2.311586975096188 \end{bmatrix}$$

$$\begin{bmatrix} 0.3779644730092274 & -0.3541688016605733 & 0.4865776629929447 & 0.7035264706814484 \\ -0.3779644730092274 & 0.7673657369312420 & 0.1255684291594696 & 0.5025189076296060 \\ 0.3779644730092274 & 0.05902813361009554 & -0.8318908431814862 & 0.4020151261036848 \\ 0.7559289460184545 & 0.5312532024908598 & 0.2354408046740056 & -0.3015113445777636 \end{bmatrix}$$

Die Umrechnungen in die GP-Zahlen sind wegen der auftretenden Wurzeln in der letzten Dezimale ungenau, es treten also nicht nur Rundungsfehler auf.

```
> # For matricies of floating point entries,
  # high speed numerically stable Householder-transformations are used.
  Digits:=16:
  AA:=evalm(A): AA[1,1]:=1.0*AA[1,1]: evalm(AA);

  R:=QRdecomp(AA,Q='q',rank='r');
  r;
  Q:=evalm(q);
  evalm(AA-Q&*R);
```

$$\begin{bmatrix} 1.0 & -1 & 1 & 2 \\ -1 & 2 & 0 & 1 \\ 1 & 0 & -1 & 1 \\ 2 & 1 & 1 & 0 \end{bmatrix}$$

$$R := \begin{bmatrix} -2.64575131106459059 & 0.37796447300922723 & -0.75592894601845445 & -0.75592894601845446 \\ 0. & -2.42015347801391661 & -0.118056267220191058 & -0.11805626722019106 \\ 0. & 0. & 1.55390931084843669 & 0.266832911963872968 \\ 0. & 0. & 0. & 2.31158697509618778 \end{bmatrix}$$

$$Q := \begin{bmatrix} -0.3779644730092272 & 0.3541688016605732 & 0.4865776629929448 & 0.7035264706814485 \\ 0.3779644730092272 & -0.7673657369312419 & 0.1255684291594696 & 0.5025189076296060 \\ -0.3779644730092272 & -0.05902813361009553 & -0.8318908431814863 & 0.4020151261036848 \\ -0.7559289460184545 & -0.5312532024908597 & 0.2354408046740056 & -0.3015113445777636 \end{bmatrix}$$

$$\begin{bmatrix} 0. & 0. & 0. & \text{-0.1}\,10^{-14} \\ 0. & \text{-0.1}\,10^{-14} & \text{-0.1}\,10^{-15} & 0. \\ 0. & \text{-0.1}\,10^{-15} & 0.1\,10^{-14} & 0. \\ 0. & 0. & 0. & \text{-0.1}\,10^{-15} \end{bmatrix}$$

Bemerkungen

- Es fällt auf, dass das Kommando `QRdecomp` die obere Dreiecksmatrix R mit mehr Dezimalstellen zurückgibt als in `Digits:=...:` gefordert sind.
  Das erinnert an die Situation mit `LUdecomp`, die im Kapitel 2.7.1 beschrieben wurde. Solche zusätzlich ausgegebenen Stellen sind mit Vorsicht zu genießen.
- Es ist zu erwarten, dass es zwischen Maple V und Maple 8 bezüglich der angezeigten Dezimalstellen wieder Unterschiede gibt.
- Wir sehen Vorzeichenwechsel in den ersten beiden Spalten von Q, entsprechend dann in den Zeilen 1 und 2 von R.

Auf die ersten Aspekte gehen wir später kurz ein.

(2) Dreiecksmatrix

Die Dreiecksform bleibt bis auf evtl. Vorzeichenwechsel in den Zeilen erhalten.

```
> n:=4:
  A:=matrix(n,n,[[ 1, 1, 1, 1],
                 [ 0, 1, 1, 1],
                 [ 0, 0, 1, 1],
                 [ 0, 0, 0, 1]]):

> R:=QRdecomp(A,Q='q');
  Q:=evalm(q);
  evalm(A-Q&*R):  # Nullmatrix
```

$$R := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> Digits:=16:
  AA:=evalm(A): AA[1,1]:=1.0*AA[1,1]: evalm(AA);
  R:=QRdecomp(AA,Q='q');
  Q:=evalm(q);
  evalm(AA-Q&*R);
```

$$\begin{bmatrix} 1.0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R := \begin{bmatrix} -1.00 & -1.00000000000000000 & -1.00000000000000000 & -1.00000000000000000 \\ 0 & -1. & -1.00000000000000000 & -1.00000000000000000 \\ 0 & 0 & -1. & -1.00000000000000000 \\ 0 & 0 & 0 & -1. \end{bmatrix}$$

$$Q := \begin{bmatrix} -1.000000000000000 & 0. & 0. & \\ 0. & -1.000000000000000 & 0. & 0. \\ 0. & 0. & -1.000000000000000 & 0. \\ 0. & 0. & 0 & -1.000000000000000 \end{bmatrix}$$

$$\begin{bmatrix} 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \end{bmatrix}$$

(3) Tridiagonalmatrix

Die Bandstruktur von A wirkt sich entsprechend auf Q und R aus.

```
> n:=4:
> A:=matrix(n,n,[[ 2,-1, 0, 0],
                 [-1, 2,-1, 0],
                 [ 0,-1, 2,-1],
                 [ 0, 0,-1, 2]]);
```

```
> R:=QRdecomp(A,Q='q');
  Q:=evalm(q);
  evalm(A-Q&*R):   # Nullmatrix
```

$$R := \begin{bmatrix} \sqrt{5} & -\frac{4\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & \frac{\sqrt{70}}{5} & -\frac{8\sqrt{70}}{35} & \frac{\sqrt{70}}{14} \\ 0 & 0 & \frac{\sqrt{105}}{7} & -\frac{4\sqrt{105}}{21} \\ 0 & 0 & 0 & \frac{\sqrt{30}}{6} \end{bmatrix}$$

$$Q := \begin{bmatrix} \frac{2\sqrt{5}}{5} & \frac{3\sqrt{70}}{70} & \frac{2\sqrt{105}}{105} & \frac{\sqrt{30}}{30} \\ -\frac{\sqrt{5}}{5} & \frac{3\sqrt{70}}{35} & \frac{4\sqrt{105}}{105} & \frac{\sqrt{30}}{15} \\ 0 & -\frac{\sqrt{70}}{14} & \frac{2\sqrt{105}}{35} & -\frac{\sqrt{30}}{10} \\ 0 & 0 & -\frac{\sqrt{105}}{15} & \frac{2\sqrt{30}}{15} \end{bmatrix}$$

```
> Digits:=16:
  evalf(evalm(R));  evalf(evalm(Q));
```

$$\begin{bmatrix} 2.236067977499790 & -1.788854381999832 & 0.4472135954999580 & 0. \\ 0. & 1.673320053068151 & -1.912365774935030 & 0.5976143046671968 \\ 0. & 0. & 1.463850109422800 & -1.951800145897067 \\ 0. & 0. & 0. & 0.9128709291752770] \end{bmatrix}$$

$$\begin{bmatrix} 0.8944271909999160 & 0.3585685828003181 & 0.1951800145897067 & 0.1825741858350553 \\ -0.4472135954999580 & 0.7171371656006361 & 0.3903600291794134 & 0.3651483716701108 \\ 0. & -0.5976143046671968 & 0.5855400437691200 & 0.5477225575051661 \\ 0. & 0. & -0.6831300510639734 & 0.7302967433402213 \end{bmatrix}$$

```
> Digits:=16:
  AA:=evalm(A): AA[1,1]:=1.0*AA[1,1]: evalm(AA);

  R:=QRdecomp(AA,Q='q');
  Q:=evalm(q);
  evalm(AA-Q&*R);
```

$$\begin{bmatrix} 2.0 & -1 & 0 & 0 \\ -1 & 2 & -1 & 1 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

$$R := \begin{bmatrix} -2.23606797749978970 & 1.78885438199983176 & -0.447213595499957940 & 0. \\ 0. & -1.67332005306815110 & 1.91236577493502980 & -0.597614304667196816 \\ 0. & 0. & -1.46385010942279978 & 1.95180014589706636 \\ 0. & 0. & 0. & -0.912870929175276860 \end{bmatrix}$$

$$Q := \begin{bmatrix} -0.8944271909999159 & -0.3585685828003181 & -0.1951800145897066 & -0.1825741858350554 \\ 0.4472135954999579 & -0.7171371656006362 & -0.3903600291794133 & -0.3651483716701107 \\ 0. & 0.5976143046671968 & -0.5855400437691199 & -0.5477225575051661 \\ 0. & 0. & 0.6831300510639732 & -0.7302967433402215 \end{bmatrix}$$

$$\begin{bmatrix} 0. & 0. & 0.1\,10^{-15} & -0.2\,10^{-15} \\ 0. & 0. & 0. & -0.1\,10^{-15} \\ 0. & -0.1\,10^{-15} & 0. & 0. \\ 0. & 0. & 0. & 0. \end{bmatrix}$$

(4) Matrix, symmetrisch und schlecht konditioniert

Eine schlechte Kondition von A schlägt sich auch in R nieder.

```
> n:=4:
  A:=hilbert(n):

> R:=QRdecomp(A,Q='q');
  Q:=evalm(q);
  evalm(A-Q&*R):  # Nullmatrix
```

$$R := \begin{bmatrix} \frac{\sqrt{205}}{12} & \frac{48\sqrt{205}}{1025} & \frac{34\sqrt{205}}{1025} & \frac{556\sqrt{205}}{21525} \\ 0 & \frac{\sqrt{2125645}}{12300} & \frac{916\sqrt{2125645}}{10628225} & \frac{5998\sqrt{2125645}}{74397575} \\ 0 & 0 & \frac{17\sqrt{51845}}{622140} & \frac{1296\sqrt{51845}}{30847775} \\ 0 & 0 & 0 & \frac{\sqrt{5}}{11900} \end{bmatrix}$$

$$Q := \begin{bmatrix} \frac{12\sqrt{205}}{205} & -\frac{762\sqrt{2125645}}{2125645} & \frac{596\sqrt{51845}}{881365} & -\frac{\sqrt{5}}{85} \\ \frac{6\sqrt{205}}{205} & \frac{644\sqrt{2125645}}{2125645} & -\frac{2817\sqrt{51845}}{881365} & \frac{12\sqrt{5}}{85} \\ \frac{4\sqrt{205}}{205} & \frac{771\sqrt{2125645}}{2125645} & \frac{108\sqrt{51845}}{176273} & -\frac{6\sqrt{5}}{17} \\ \frac{3\sqrt{205}}{205} & \frac{732\sqrt{2125645}}{2125645} & \frac{506\sqrt{51845}}{176273} & \frac{4\sqrt{5}}{17} \end{bmatrix}$$

```
> Digits:=16:
  evalf(evalm(R));
  evalf(evalm(Q));
```

$$\begin{bmatrix} 1.193151755273029 & 0.6704930839387950 & 0.4749326011233130 & 0.3698354709027480 \\ 0. & 0.1185332674878872 & 0.1256550946308087 & 0.1175419927628806 \\ 0. & 0. & 0.006221774060128569 & 0.009566092949393922 \\ 0. & 0. & 0. & 0.0001879048720588059 \end{bmatrix}$$

$$\begin{bmatrix} 0.8381163549234937 & -0.5226483739556563 & 0.1539727615157077 & -0.02630668208823282 \\ 0.4190581774617468 & 0.4417133239205284 & -0.7277538073653502 & 0.3156801850587939 \\ 0.2793721183078312 & 0.5288213862464712 & 0.1395055221786614 & -0.7892004626469846 \\ 0.2095290887308733 & 0.5020716663196068 & 0.6536092057629874 & 0.5261336417646564 \end{bmatrix}$$

```
> Digits:=16:
  AA:=evalm(A): AA[1,1]:=1.0*AA[1,1]: evalm(AA);

  R:=QRdecomp(AA,Q='q');
  Q:=evalm(q);
  evalm(AA-Q&*R);
```

$$\begin{bmatrix} 1.0 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

$$R := \begin{bmatrix} -1.19315175527302942 & -0.67049308393879506 & -0.474932601123313152 & -0.369835470902748092 \\ 0. & -0.118533267487887182 & -0.125655094630808797 & -0.117541992762880725 \\ 0. & 0. & -0.00622177406012855824 & -0.0095660929493939279 \\ 0. & 0. & 0. & -0.00018790487205876631 0 \end{bmatrix}$$

$$Q := \begin{bmatrix} -0.8381163549234938 & 0.5226483739556564 & -0.1539727615157077 & 0.02630668208823379 \\ -0.4190581774617469 & -0.4417133239205284 & 0.7277538073653483 & -0.3156801850587979 \\ -0.2793721183078313 & -0.5288213862464715 & -0.1395055221786569 & 0.7892004626469853 \\ -0.2095290887308735 & -0.5020716663196071 & -0.6536092057629901 & -0.5261336417646529 \end{bmatrix}$$

$$\begin{bmatrix} 0.4\,10^{-15} & 0. & -0.1\,10^{-15} & 0. \\ 0.2\,10^{-15} & -0.1\,10^{-15} & 0. & 0. \\ 0.1\,10^{-15} & 0. & -0.1\,10^{-15} & 0. \\ 0. & 0. & 0. & 0. \end{bmatrix}$$

Wir kehren zur ersten Matrix mit den numerischen Einträgen

$$AA = \begin{pmatrix} 1.0 & -1.0 & 1.0 & 2.0 \\ -1.0 & 2.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & -1.0 & 1.0 \\ 2.0 & 1.0 & 1.0 & 0.0 \end{pmatrix}$$

zurück und betrachten die Anzeige der Dezimalstellen (DS) von Matrixelementen bei Ausführung von `QRdecomp` in Maple V bzw. Maple 8 mit der Wahl von `Digits:=...`. Man beachte die kleinen Unterschiede zwischen den Maple-Versionen sowie auch den Vorzeichenwechsel in r_{11} bei `Digits:=13,14`.

```
> # For matricies of floating point entries,
  # high speed numerically stable Householder-transformations are used.

  Digits:=1:  # 1,2,...,13,14,...,21
  R:=QRdecomp(AA,Q='q');
  Q:=evalm(q);
```

	Maple V		Maple 8	
Digits	r_{11} = 2.6...64590590501615...	DS	r_{11} = 2.6...64590590501615...	DS
1	2.645751311064591	16	2.64575131106459072	18
2	2.645751311064591	16	2.64575131106459072	18
...				
13	2.645751311064591	16	2.64575131106459072	18
14	-2.645751311064591	16	-2.645751311064591	16
15	-2.6457513110645906	17	-2.6457513110645906	17
16	-2.64575131106459059	18	-2.64575131106459059	18
17	-2.645751311064590591	19	-2.645751311064590591	19
18	-2.6457513110645905905	20	-2.6457513110645905905	20
19	-2.64575131106459059050	21	-2.64575131106459059050	21
20	-2.645751311064590590502	22	-2.645751311064590590502	22
21	-2.6457513110645905905016	23	-2.6457513110645905905016	23

Tab. 2.12 Maple V und 8 bez. der Anzeige von r_{11} in `QRdecomp(A)` in der GPA `Digits:=1,2,...,21:`
exaktes Matrixelement ist $r_{11} = \sqrt{7} = 2.64575131106459059050161575...$

Die Situation ist nicht ganz so verwirrend, wie beim Kommando `LUdecomp(A)` mit "voller Anzeige" bei nur wenigen gültigen Stellen (siehe Tab. 2.10).

Rechnungen in Matlab (Datei: *qrf1.m*)

Matlab führt die QR-Faktorisierung numerisch in der GPA *double* durch. Deshalb haben wir in Maple auch die Genauigkeit `Digits:=16:` mit einbezogen.
Aus der Online-Hilfe erfährt man u. a. folgendes.

```
qr  Orthogonal-triangular decomposition.

   [Q,R] = qr(A) produces an upper triangular matrix R of the same
   dimension as A and a unitary matrix Q so that A = Q*R.

   By itself, qr(A) is the output of LAPACK'S DGEQRF or ZGEQRF
   routine.
   It is a full matrix. The upper triangular part is R, so that
   triu(qr(A)) is R.

   [Q,R,P] = qr(A) produces a permutation matrix P, an upper
   triangular R and a unitary Q so that A*P = Q*R.  The column
   permutation P is chosen so that abs(diag(R)) is decreasing.

   [Q,R] = qr(A,0) produces the "economy size" decomposition.
   If A is m-by-n with m > n, then only the first n columns of Q
   are computed.
```

Wir faktorisieren dieselben Beispielmatrizen.

Beispiel 2.3 Matrix A quadratisch

(1) Symmetrische Matrix

```
disp('(1) Symmetrische Matrix')
clc
n = 4;
A = [ 1 -1  1  2
     -1  2  0  1
      1  0 -1  1
      2  1  1  0 ]
format long e
disp('MATLAB-Funktion qr fuer QR-Faktorisierung')
qr(A)
R = qr(A)
triu(qr(A))

[Q,R] = qr(A)
A-Q*R
```

```
MATLAB-Funktion qr fuer QR-Faktorisierung
qr(A)
ans =
  Columns 1 through 3
   -2.645751311064591e+000    3.779644730092273e-001   -7.559289460184544e-001
   -2.742918851774318e-001   -2.420153478013916e+000   -1.180562672201910e-001
    2.742918851774318e-001    9.350489859362447e-002    1.553909310848437e+000
    5.485837703548635e-001    4.344005052322925e-001    7.238313905834130e-002
  Column 4
   -7.559289460184546e-001
   -1.180562672201906e-001
    2.668329119638730e-001
   -2.311586975096188e+000

R = qr(A)
R =
  Columns 1 through 3
   -2.645751311064591e+000    3.779644730092273e-001   -7.559289460184544e-001
   -2.742918851774318e-001   -2.420153478013916e+000   -1.180562672201910e-001
    2.742918851774318e-001    9.350489859362447e-002    1.553909310848437e+000
    5.485837703548635e-001    4.344005052322925e-001    7.238313905834130e-002
  Column 4
   -7.559289460184546e-001
   -1.180562672201906e-001
    2.668329119638730e-001
   -2.311586975096188e+000

triu(qr(A))
ans =
  Columns 1 through 3
   -2.645751311064591e+000    3.779644730092273e-001   -7.559289460184544e-001
                         0   -2.420153478013916e+000   -1.180562672201910e-001
                         0                         0    1.553909310848437e+000
                         0                         0                         0
  Column 4
   -7.559289460184546e-001
   -1.180562672201906e-001
    2.668329119638730e-001
   -2.311586975096188e+000

[Q,R] = qr(A)
Q =
  Columns 1 through 3
   -3.779644730092273e-001    3.541688016605733e-001    4.865776629929448e-001
    3.779644730092273e-001   -7.673657369312417e-001    1.255684291594696e-001
   -3.779644730092273e-001   -5.902813361009550e-002   -8.318908431814864e-001
   -7.559289460184545e-001   -5.312532024908597e-001    2.354408046740056e-001
  Column 4
   -7.035264706814485e-001
   -5.025189076296062e-001
   -4.020151261036848e-001
    3.015113445777636e-001
```

```
R =
  Columns 1 through 3
   -2.645751311064591e+000    3.779644730092273e-001   -7.559289460184544e-001
                         0   -2.420153478013916e+000   -1.180562672201910e-001
                         0                         0    1.553909310848437e+000
                         0                         0                         0
  Column 4
   -7.559289460184546e-001
   -1.180562672201906e-001
    2.668329119638730e-001
   -2.311586975096188e+000

A-Q*R
ans =
  Columns 1 through 3
   -2.220446049250313e-016    4.440892098500626e-016                         0
    2.220446049250313e-016    4.440892098500626e-016    2.775557561562891e-017
   -2.220446049250313e-016    1.387778780781446e-016                         0
   -4.440892098500626e-016    2.220446049250313e-016                         0
  Column 4
   -4.440892098500626e-016
    1.110223024625157e-016
                         0
    1.110223024625157e-016
```

Bezüglich der Genauigkeit (maximale Abweichungen) ist das *double*-Format von Matlab günstiger als die Einstellung `Digits:=16:` in der numerischen Rechnung von Maple, welche die Fehlermatrix

$$A - QR = \begin{pmatrix} 0 & 0 & 0 & -10^{-15} \\ 0 & -10^{-15} & -10^{-16} & 0 \\ 0 & -10^{-16} & 10^{-15} & 0 \\ 0 & 0 & 0 & -10^{-16} \end{pmatrix}$$

lieferte. Es gibt auch kleine Abweichungen in der Vorzeichensituation in den Spalten von Q und entsprechenden Zeilen von R.

(2) Dreiecksmatrix

```
n = 4;
A = [ 1  1  1  1
      0  1  1  1
      0  0  1  1
      0  0  0  1 ]

disp('MATLAB-Funktion qr fuer QR-Faktorisierung')
[Q,R] = qr(A)
A-Q*R
```

```
MATLAB-Funktion qr fuer QR-Faktorisierung
[Q,R] = qr(A)
Q =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

R =
     1     1     1     1
     0     1     1     1
     0     0     1     1
     0     0     0     1

A-Q*R
ans =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

(3) Tridiagonalmatrix

```
n = 4;
A = [ 2  -1   0   0
     -1   2  -1   0
      0  -1   2  -1
      0   0  -1   2 ]

disp('MATLAB-Funktion qr fuer QR-Faktorisierung')
[Q,R] = qr(A)
A-Q*R
```

```
MATLAB-Funktion qr fuer QR-Faktorisierung
[Q,R] = qr(A)
Q =
  Columns 1 through 3
   -8.944271909999157e-001   -3.585685828003181e-001   -1.951800145897067e-001
    4.472135954999579e-001   -7.171371656006363e-001   -3.903600291794134e-001
                         0    5.976143046671969e-001   -5.855400437691201e-001
                         0                         0    6.831300510639732e-001
  Column 4
    1.825741858350554e-001
    3.651483716701107e-001
    5.477225575051661e-001
    7.302967433402215e-001
```

```
R =
  Columns 1 through 3
   -2.236067977499790e+000    1.788854381999832e+000   -4.472135954999579e-001
                         0   -1.673320053068151e+000    1.912365774935030e+000
                         0                         0   -1.463850109422800e+000
                         0                         0                         0
  Column 4
                         0
   -5.976143046671969e-001
    1.951800145897066e+000
    9.128709291752772e-001

A-Q*R
ans =
  Columns 1 through 3
    2.220446049250313e-016   -5.551115123125783e-016                         0
                         0                         0                         0
                         0                         0   -4.440892098500626e-016
                         0                         0                         0
  Column 4
                         0
   -5.551115123125783e-017
                         0
                         0
```

(4) Matrix, symmetrisch und schlecht konditioniert

```
n = 4;
A = [  1   1/2  1/3  1/4
      1/2  1/3  1/4  1/5
      1/3  1/4  1/5  1/6
      1/4  1/5  1/6  1/7 ]  % A=hilb(4)

disp('MATLAB-Funktion qr fuer QR-Faktorisierung')
[Q,R] = qr(A)
A-Q*R
```

```
MATLAB-Funktion qr fuer QR-Faktorisierung
[Q,R] = qr(A)
Q =
  Columns 1 through 3
   -8.381163549234936e-001    5.226483739556564e-001   -1.539727615157079e-001
   -4.190581774617469e-001   -4.417133239205285e-001    7.277538073653515e-001
   -2.793721183078313e-001   -5.288213862464714e-001   -1.395055221786651e-001
   -2.095290887308735e-001   -5.020716663196071e-001   -6.536092057629848e-001
  Column 4
   -2.630668208823198e-002
    3.156801850587902e-001
   -7.892004626469840e-001
    5.261336417646596e-001
```

```
R =
  Columns 1 through 3
   -1.193151755273030e+000   -6.704930839387948e-001   -4.749326011233132e-001
                         0   -1.185332674878872e-001   -1.256550946308087e-001
                         0                         0   -6.221774060128572e-003
                         0                         0                         0
  Column 4
   -3.698354709027479e-001
   -1.175419927628807e-001
   -9.566092949393871e-003
    1.879048720588566e-004

A-Q*R
ans =
  Columns 1 through 3
    2.220446049250313e-016    3.885780586188048e-016                         0
                         0    1.110223024625157e-016    5.551115123125783e-017
                         0    5.551115123125783e-017   -2.775557561562891e-017
                         0    5.551115123125783e-017    2.775557561562891e-017
  Column 4
    2.498001805406602e-016
    5.551115123125783e-017
                         0
    2.775557561562891e-017
```

2.8.1 Gram-Schmidt-Verfahren

Die Herleitung des Gram-Schmidt-Verfahrens bzw. des Gram-Schmidtschen-Orthogonalisierungsverfahrens ergibt sich unmittelbar aus dem konstruktiven Existenznachweis der QR-Faktorisierung.
Bei einer regulären Matrix A besitzt die Matrix Q also paarweise orthogonale Spaltenvektoren $q_1, q_2, \ldots, q_n$, die die (euklidische) Länge Eins haben, und R ist eine reguläre obere Dreiecksmatrix.
Aus dem Ansatz $A = QR$ lässt sich unmittelbar ableiten, dass man die j-te Spalte a_j von A als Linearkombination der ersten j Spalten von Q schreiben kann.

$$a_j \;=\; \sum_{i=1}^{j} r_{ij} q_i.$$

Wegen der Orthogonalität der q_j ist

$$r_{ij} \;=\; (a_j, q_i).$$

Da $r_{ii} \neq 0$ ist, kann zuerst q_1 durch a_1 ausgedrückt werden, dann wegen $a_2 = r_{12}q_1 + r_{22}q_2$ der Vektor q_2 durch a_2, q_1, somit auch a_1, a_2, und allgemein lässt sich dann auch

die j-te Spalte q_j von Q als Linearkombination

$$q_j \;=\; \sum_{i=1}^{j} s_{ij} a_i$$

mit geeigneten Koeffizienten s_{ij} schreiben.
Somit spannen die Vektoren $a_1, \ldots, a_j$ und $q_1, \ldots, q_j$ denselben Vektorraum auf, d. h. $\text{span}\{a_1, a_2, ..., a_j\} = \text{span}\{q_1, q_2, ..., q_j\}$.
Damit kommen wir zum Gram-Schmidt-Verfahren mit der Konstruktion der q_j. Zunächst erkennt man, dass man im ersten Schritt einfach

$$q_1 \;=\; \pm\frac{a_1}{\|a_1\|_2}, \quad a_1 \neq 0,$$

(das Vorzeichen kann nach Belieben festgelegt werden) sowie

$$r_{11} \;=\; \pm\|a_1\|_2$$

wählt. Sind die orthonormalen Vektoren $q_1, q_2, \ldots, q_{k-1}$ bestimmt, so werden q_k und die k-te Spalte r_k von R durch die folgenden Schritte ermittelt.

(1) Man wählt als Ansatz für den nicht normierten Vektor

$$\tilde{q}_k \;=\; a_k + \sum_{i=1}^{k-1} c_{ik} q_i$$

und bestimmt die c_{ik} aus der Forderung der Orthogonalität von $\tilde{q}_k$ zu $q_j,\ j = 1, 2, \ldots, k-1$. Dies führt auf die $k-1$ linearen Gleichungen für c_{ik}

$$(a_k, q_j) + \sum_{i=1}^{k-1} c_{ik}\ (q_i, q_j) \;=\; 0, \quad j = 1, 2, ..., k-1.$$

Aus der Orthonormalität der q_i folgt

$$c_{jk} \;=\; -(a_k, q_j) \;=\; -r_{jk}, \quad j = 1, 2, ..., k-1.$$

(2) Damit erhält man

$$\tilde{q}_k \;=\; a_k - \sum_{i=1}^{k-1} r_{ik} q_i.$$

(3) Man berechnet $\gamma = \|\tilde{q}_k\|_2$. Somit hat $q_k = \tilde{q}_k/\gamma$ die euklidische Norm Eins.

(4) Damit ist $r_{kk} = (a_k, q_k) = \left(\tilde{q}_k + \sum_{i=1}^{k-1} r_{ik} q_i, \tilde{q}_k/\gamma\right) = (\tilde{q}_k, \tilde{q}_k)/\gamma = \gamma.$

Jedoch treten in dieser Basisversion Rundungsfehler und damit Stellenauslöschungen auf, so dass es Probleme mit der Spaltenorthogonalität in Q gibt. Unter anderem wird eine Modifikation der Berechnung $r_{jk} = (a_k, q_j)$ vorgenommen. Für den praktischen Einsatz werden verbesserte Versionen verwendet [28].

Die Umsetzung dieser insgesamt fünf Schritte ist als folgender Blockalgorithmus beschrieben. Dabei wird der anfängliche kurze Schritt in die Schleife mit einbezogen. Er könnte auch vorab stehen bleiben, so dass sich dann die Schleife um einen Schritt verkürzt.

k = 1,2,...,n

(1) Berechnung der $r_{jk},\ j = 1, 2, ..., k-1$

for $j = 1, 2, ..., k-1$ do

$r_{jk} = 0$

for $i = 1, 2, ..., n$ do $r_{jk} = r_{jk} + a_{ik} q_{ij}$

(2) Berechnung von $\tilde{q}_k$

for $j = 1, 2, ..., n$ do

$q_{jk} = a_{jk}$

for $i = 1, 2, ..., k-1$ do $q_{jk} = q_{jk} - r_{ik} q_{ji}$

(3) Berechnung von γ und q_k

$\gamma = 0$

for $j = 1, 2, ..., n$ do $\gamma = \gamma + q_{jk}^2$

$\gamma = \sqrt{\gamma}$

for $j = 1, 2, ..., n$ do $q_{jk} = q_{jk}/\gamma$

(4) Berechnung von r_{kk}

$r_{kk} = \gamma$

end k

Die Schritte (1) und (2) können in einer Schleife $1, 2, ..., k-1$ zusammengefasst werden. Für die k-te Spalte von A und Q ist die Einführung von Hilfsvektoren sinnvoll. Die Durchführbarkeit haben wir bei einer regulären Matrix A, in jedem Fall kann die Untersuchung der Regularität mit dem Test der Größe γ auf Null erfolgen.

Bei der Komplexitätsanalyse der QR-Faktorisierung nach Gram-Schmidt erhalten wir $n^3 + \mathcal{O}(n^2)$ Multiplikationen, so dass für große n der Aufwand ungefähr dreimal so hoch ist wie beim Gauß-Algorithmus/LU-Faktorisierung.

Es folgen die Prozeduren für die Orthogonalisierung von linear unabhängigen Vektoren nach Gram-Schmidt, Vektoren als Spaltenvektoren in Matrix $A(n, n)$,
Test auf Durchführbarkeit mit Toleranz.

Maple-Version

```
> Orth_GS:=proc(n::posint,A::matrix,etol::numeric)
  local Q,R,p,ps,y,r,i,j;

  # Initialisierungen
  R:=array(sparse,1..n,1..n);
  Q:=matrix(n,0,[]);
  y:=vector(n,[]);
  p:=evalm(y);

  # Schritt j=1
  y:=evalm(col(A,1));
  # r:=evalf(norm(y,2));  # fuer GPA
  r:=norm(y,2);

  R[1,1]:=r;
  if evalf(r)<etol then RETURN(1,Q,R); end if; # Kontrolle
  p:=evalm(y/r);
  Q:=concat(Q,p);

  # Schritte j=2,3,...,n
  for j from 2 by 1 to n do
   y:=evalm(col(A,j));
   ps:=evalm(y);
   for i from 1 by 1 to j-1 do
    p:=evalm(col(Q,i));
    r:=evalm(transpose(y)&*p);
    R[i,j]:=r;
    ps:=evalm(ps-r*p);
   end do;
   # r:=evalf(norm(ps,2));
   r:=norm(ps,2);
   R[j,j]:=r;
   if evalf(r)<etol then break end if;
   p:=evalm(ps/r);
   Q:=concat(Q,p);
  end do:
  if j>n then j:=j-1; end if;
  [j,Q,R];
  end:

> # Aufruf, Beispiel (1)
  n:=4:
  A:=matrix(n,n,[[1,-1,1,2],[-1,2,0,1],[1,0,-1,1],[2,1,1,0]]);
  gen:=1E-6:
  erg:=Orth_GS(n,A,gen);
  erg[1];
  Q:=evalm(erg[2]);
  R:=evalm(erg[3]);

  evalm(transpose(Q)&*Q);   # Orthonormalitaet
  evalf(evalm(Q));
  evalf(evalm(R));
```

Matlab-Version, Prozedur als *m*-File gram_sch.m

```
% gram_sch.m
% QR-Faktorisierung und Gram-Schmidt-Orthogonalisierung

function [Q,R] = gram_sch(n,A,epsi)

% Eingangsparameter
%  n        Dimension der Matrix
%  A        Matrix
%  epsi     Toleranz fuer Abbruch bei Nicht-Durchfuehrbarkeit
%
% Ergebnisse
%  Q        Orthogonalmatrix
%  R        Rechte Dreiecksmatrix

Q = eye(n,n);
R = zeros(n,n);

for k=1:n
 ak = A(:,k);
 for j=1:k-1
  qj = Q(:,j);
  R(j,k) = ak'*qj;
 end;
 rk = R(:,k);
 qk = ak;
 for i=1:k-1
  qk = qk-rk(i)*Q(:,i);
 end;
 Q(:,k) = qk;
 gamma = norm(qk,2);
 if (gamma<epsi)
  disp('QR-Faktorisierung nicht durchfuehrbar');
  disp('Abbruch im Schritt k:');
  disp(k);
  break;
 end;
 qk = qk/gamma;
 Q(:,k) = qk;
 R(k,k) = gamma;
end;
% Ende Funktion gram_sch
```

```
% Beispiel (1)
n = 4;
A = [ 1 -1  1  2
     -1  2  0  1
      1  0 -1  1
      2  1  1  0 ]
epsi = 1e-9

% m-File gram_sch  fuer QR-Faktorisierung
disp('m-File gram_sch fuer QR-Faktorisierung')
disp('gram_sch');
[Q,R] = gram_sch(n,A,epsi)
A-Q*R
```

2.8.2 Givens-Rotationen

Die Rotationsmatrix

$$R_{qp} = \begin{pmatrix} 1 & & & & & & & & 0 \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & \cos(\varphi) & & & \sin(\varphi) & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & -\sin(\varphi) & & & \cos(\varphi) & & \\ & & & & & & & 1 & \\ & & & & & & & & \ddots \\ 0 & & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \rightarrow\ p\text{-te Zeile}, \\ \\ \\ \\ \rightarrow\ q\text{-te Zeile}, \\ \\ \\ \\ \end{matrix}$$

führt eine Rotation in der pq-Ebene aus. Mit ihr wird das Matrixelement a_{qp} verändert. Es gilt $R_{qp}^T R_{qp} = I,\ R_{qp}^T = R_{qp}^{-1}$. Das Produkt solcher Rotationsmatrizen erfüllt ebenfalls die Bedingung der Orthogonalität.
Die Matrix R_{qp} wird dazu verwendet, die Subdiagonalelemente $a_{21}, a_{31}, ..., a_{n1},\ a_{32}, ..., a_{n2}, ..., a_{n,n-1}$ der Matrix A der Reihe nach jeweils zu Null zu machen, falls sie es noch nicht sind. Bisherige "Eliminationen" bleiben dabei erhalten.
Die Folge der Rotationen gemäß

$$R \ = \ R_{n,n-1} \cdot ... \cdot R_{32} R_{n1} \cdot ... \cdot R_{31} R_{21} A$$

führt somit auf eine obere Dreiecksmatrix R und weiter auf die gewünschte Faktorisierung

$$R \ = \ Q^{-1} A \ \ \Rightarrow \ \ A \ = \ QR, \ \ Q \ = \ R_{21}^T R_{31}^T \cdot ... \cdot R_{n,n-1}^T.$$

Bestimmung der Rotationsmatrix R_{qp}

Die Matrix A enthält auch die Zwischenergebnisse, d. h. die Umformungen der Matrix finden am Platz statt. Wir führen einen Schritt allgemein durch.

$R_{qp}A \;=$

$$\begin{pmatrix} 1 & & & & & & 0 \\ & \ddots & & & & & \\ & & \cos(\varphi) & & \sin(\varphi) & & \\ & & & 1 & & & \\ & & & & \ddots & & \\ & & & & & 1 & \\ & & -\sin(\varphi) & & \cos(\varphi) & & \\ & & & & & \ddots & \\ 0 & & & & & & 1 \end{pmatrix} \begin{pmatrix} a_{11} & \dots & a_{1p} & a_{1,p+1} & \dots & a_{1n} \\ & \ddots & \vdots & \vdots & \dots & \vdots \\ & & a_{pp} & a_{p,p+1} & \dots & a_{pn} \\ & & 0 & a_{p+1,p+1} & \dots & a_{p+1,n} \\ & & \vdots & \vdots & \vdots & \vdots \\ & & 0 & a_{q-1,p+1} & \dots & a_{q-1,n} \\ & & a_{qp} & a_{q,p+1} & \dots & a_{qn} \\ & & \vdots & \vdots & \vdots & \vdots \\ 0 & & a_{np} & a_{n,p+1} & \dots & a_{nn} \end{pmatrix},$$

c, s seien Abkürzungen für $\cos(\varphi)$ bzw. $\sin(\varphi)$,

$$R_{qp}A \;= \begin{pmatrix} a_{11} & \dots & a_{1p} & a_{1,p+1} & \dots & a_{1n} \\ & \ddots & \vdots & \vdots & \dots & \vdots \\ & & ca_{pp} + sa_{qp} & ca_{p,p+1} + sa_{q,p+1} & \dots & ca_{pn} + sa_{qn} \\ & & 0 & a_{p+1,p+1} & \dots & a_{p+1,n} \\ & & \vdots & \vdots & \vdots & \vdots \\ & & 0 & a_{q-1,p+1} & \dots & a_{q-1,n} \\ & & -sa_{pp} + ca_{qp} & -sa_{p,p+1} + ca_{q,p+1} & \dots & -sa_{pn} + ca_{qn} \\ & & \vdots & \vdots & \vdots & \vdots \\ 0 & & a_{np} & a_{n,p+1} & \dots & a_{nn} \end{pmatrix}.$$

Nun wählen wir c, s so, dass das nächste Matrixelement in der p-ten Spalte, also $-sa_{pp} + ca_{qp}$ Null wird. In ausgeschriebener Form heißt das

$$\begin{aligned} 0 &= -a_{pp}\sin(\varphi) + a_{qp}\cos(\varphi), \quad a_{qp} \neq 0, \\ \cot(\varphi) &= \frac{\cos(\varphi)}{\sin(\varphi)} = \frac{a_{pp}}{a_{qp}} = \theta, \\ \tan(\varphi) &= \frac{a_{qp}}{a_{pp}} = \frac{1}{\theta} = t, \\ c = \cos(\varphi) &= \frac{1}{\sqrt{1+\tan^2(\varphi)}} = \frac{1}{\sqrt{1+t^2}} = \frac{\theta}{\sqrt{1+\theta^2}} = \frac{a_{pp}}{\sqrt{a_{pp}^2 + a_{qp}^2}}, \\ s = \sin(\varphi) &= \frac{\tan(\varphi)}{\sqrt{1+\tan^2(\varphi)}} = \frac{t}{\sqrt{1+t^2}} = \frac{1}{\sqrt{1+\theta^2}} = \frac{a_{qp}}{\sqrt{a_{pp}^2 + a_{qp}^2}}. \end{aligned}$$

Bei einer Implementierung wählt man die Formeln mit t bzw. θ in Abhängigkeit davon, welche der Größen betragsmäßig ≤ 1 ist.

Aufwand

Der Rechenaufwand für die LU-Faktorisierung einer Matrix, die wesentlicher Teil des verketteten Gauß-Algorithmus ist, beträgt größenordnungsmäßig

$\frac{n^3}{3}$ Multiplikationen und $\frac{n^3}{3}$ Additionen.

Der Aufwand bei der QR-Faktorisierung wird dadurch, dass jeweils zwei Zeilen der Matrix verändert werden, größer sein.

Ordnungsschätzung der Komplexität $T(n)$

1. Spalte:	$n-1$ Elemente zu 0,	je Element $4n$ “$*$“ und $2n$ “$+$“,
2. Spalte:	$n-2$ Elemente zu 0,	je Element $4(n-1)$ “$*$“ und $2(n-1)$ “$+$“,
.....		
$(n-1)$-te Spalte:	1 Element zu 0,	je Element $4*2$ “$*$“ und $2*2$ “$+$“.

$$\begin{aligned} T(n) &\approx \sum_{k=1}^{n-1} \{4(n-k)(n-k+1) \text{ “}*\text{“ } + \; 2(n-k)(n-k+1) \text{ “}+\text{“}\} \\ &\approx 4\sum_{k=1}^{n} k^2 \text{ “}*\text{“ } + \; 2\sum_{k=1}^{n} k^2 \text{ “}+\text{“} \\ &\approx \frac{4n^3}{3} \text{ “}*\text{“ } + \; \frac{2n^3}{3} \text{ “}+\text{“}. \end{aligned}$$

Prozedur in TP (Umbenennung $p \to j,\ q \to k$)

```
procedure QR_Zerlegung_einfach(n:integer; var a,q,r:matrix);
{ QR-Faktorisierung von A(n,n) mittels Rotationsmatrizen
  zyklischer Durchlauf fuer Elemente a[2,1], a[3,1],...,a[n,n-1]
  A=Q*R, Q(n,n) Orthonormalmatrix, Q'*Q=I(n,n)
         R(n,n) obere Dreiecksmatrix
  Speicherung von R auf dem Platz von A, dann Umspeicherung     }
var i,j,k,l:integer;
    s,theta,t,c,ajl,akl,qij,qik:float;
begin
  fillchar(r,sizeof(r),0);     { Initialisierung von Q,R }
  q:=r;
  for i:=1 to n do q[i,i]:=1;  { Q=I Einheitsmatrix }

  for j:=1 to n-1 do           { spaltenweise Elimination }
   for k:=j+1 to n do
     if a[k,j]+1.0<>1.0 then
      begin
```

```
      if abs(a[k,j])>abs(a[j,j]) then  { Fallunterscheidung }
       begin                           { numerisch guenstiger }
        theta:=a[j,j]/a[k,j];
        s:=1.0/sqrt(1.0+sqr(theta));  c:=s*theta;
       end else
       begin
        t:=a[k,j]/a[j,j];
        c:=1.0/sqrt(1.0+sqr(t));  s:=c*t;
       end;
      for l:=j to n do        { Linksmultiplikation von A }
       begin
        ajl:=a[j,l]; akl:=a[k,l];
        a[j,l]:= c*ajl+s*akl;
        a[k,l]:=-s*ajl+c*akl;
       end;
      for i:=1 to n do        { Rechtsmultiplikation von Q }
       begin
        qij:=q[i,j]; qik:=q[i,k];
        q[i,j]:= c*qij+s*qik;
        q[i,k]:=-s*qij+c*qik;
       end;
     end;
  { Umspeichern A -> R spaltenweise }
  for j:=1 to n do for i:=1 to j do r[i,j]:=a[i,j];
end;
```

Maple-Version

```
> givens_r:=proc(n::posint,A::matrix)
  local Q,R,theta,c,s,i,j,k,l,ajl,akl,qij,qik;

  # Eingangsparameter
  #  n        Dimension der Matrix
  #  A        Matrix
  #
  # Ergebnisse
  #  Q        Orthogonalmatrix
  #  R        Rechte Dreiecksmatrix

  # Initialisierungen
  Q:=evalm(array(identity,1..n,1..n));
  R:=evalm(A);

  for j from 1 by 1 to n-1 do
   for k from j+1 by 1 to n do
    if evalf(R[k,j]+1)<>1 then
```

```
    if evalf(abs(R[k,j]))>evalf(abs(R[j,j])) then
               # Fallunterscheidung numerisch guenstiger
     theta:=R[j,j]/R[k,j];
     s:=1/sqrt(1+theta^2);
     c:=s*theta;
    else
     theta:=R[k,j]/R[j,j];
     c:=1/sqrt(1+theta^2);
     s:=c*theta;
    end if;

    # Linksmultiplikation von R
    for l from j by 1 to n do
     ajl:=R[j,l];
     akl:=R[k,l];
     R[j,l]:= c*ajl+s*akl;
     R[k,l]:=-s*ajl+c*akl;
    end do;
    # Rechtsmultiplikation von Q
    for i from 1 by 1 to n do
     qij:=Q[i,j];
     qik:=Q[i,k];
     Q[i,j]:= c*qij+s*qik;
     Q[i,k]:=-s*qij+c*qik;
    end do;
   end if;

  end do;
 end do;

 # R zu Null im unteren Dreieck
 for j from 1 by 1 to n-1 do
  for i from j+1 by 1 to n do
   R[i,j]:=0;
  end do;
 end do;

 [Q,R];
 end:


> # Aufruf
 # Initialisierungen: n ,A

 erg:=givens_r(n,A);
 Q:=simplify(evalm(erg[1]));
 R:=simplify(evalm(erg[2]));

 evalm(transpose(Q)&*Q);   # Orthonormalitaet
 evalf(evalm(Q));
 evalf(evalm(R));
```

Matlab-Version, Prozedur als *m*-File givens_r.m

```
% givens_r.m
% QR-Faktorisierung mit Givens-Rotationen

function [Q,R] = givens_r(n,A)

% Eingangsparameter
%  n        Dimension der Matrix
%  A        Matrix
%
% Ergebnisse
%  Q        Orthogonalmatrix
%  R        Rechte Dreiecksmatrix

Q = eye(n,n);
R = A;

for j=1:n-1
 for k=j+1:n
  if (R(k,j)+1.0 ~= 1.0)
   if (abs(R(k,j))>abs(R(j,j)))
              % Fallunterscheidung numerisch guenstiger
    theta = R(j,j)/R(k,j);
    s = 1.0/sqrt(1.0+theta^2);
    c = s*theta;
   else
    theta = R(k,j)/R(j,j);
    c = 1.0/sqrt(1.0+theta^2);
    s = c*theta;
   end;

   % Linksmultiplikation von R
   for l=j:n
    ajl = R(j,l);
    akl = R(k,l);
    R(j,l) =  c*ajl+s*akl;
    R(k,l) = -s*ajl+c*akl;
   end;

   % Rechtsmultiplikation von Q
   for i=1:n
    qij = Q(i,j);
```

```
    qik = Q(i,k);
    Q(i,j) =  c*qij+s*qik;
    Q(i,k) = -s*qij+c*qik;
   end;
  end;
% disp(R);
 end;
end;

% R zu Null im unteren Dreieck
for j=1:n-1
 for i=j+1:n
  R(i,j) = 0;
 end;
end;
% Ende Funktion givens_r

% Aufruf
% Initialisierungen: n, A
% m-File givens_r  fuer QR-Faktorisierung

disp('m-File givens_r  fuer QR-Faktorisierung')
disp('givens_r');
[Q,R] = givens_r(n,A)
A-Q*R
```

2.8.3 Householder-Reflexionen

Wir verwenden die Reflexionsmatrix $S = I - 2\frac{uu^T}{u^Tu}$, um einen Vektor x auf den verlängerten ersten Einheitsvektor zu transformieren. Die Erzeugung von Nullkomponenten ab der zweiten Komponente des Vektors nutzt man bei der schrittweisen Transformation einer Matrix auf die obere Dreiecksform, indem der Reihe nach alle Spalten gespiegelt werden.
Sei $A = (a_1, a_2, ..., a_n)$ mit a_i als Spalten der Matrix.

Transformation eines Vektors x

$$\begin{aligned} u &= x - ce_1, \quad c = -\text{sign1}(x_1)\ \|x\|_2,\ |c| = \|x\|_2, \\ u &= (x_1 - c, x_2, x_3, ..., x_n)^T, \quad u_1 = x_1 - c, \\ ce_1 &= Sx, \quad S = I - 2\frac{uu^T}{u^Tu},\ f = \frac{2}{u^Tu} = \frac{1}{\|x\|_2^2 - cx_1} = -\frac{1}{cu_1}. \end{aligned}$$

Transformation der Matrix A

Schritt $k = 0$

$A^{(0)} = A = (a_1, A'), \quad x = a_1, \quad Q_0 = S_0,$

$$A^{(1)} = Q_0 A^{(0)} = S_0(a_1, A') = (S_0 a_1, S_0 A') = (c_1 e_1, S_0 A') = \begin{pmatrix} c_1 & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix}.$$

Schritt $k = 1, 2, 3, ..., n-2$

$$A^{(k)} = \begin{pmatrix} c_1 & \dots & * & \dots & * \\ 0 & \ddots & \vdots & & \vdots \\ \cdot & & c_k & \dots & * \\ \cdot & & 0 & & \\ \vdots & & \vdots & T^{(k)} & \\ 0 & \dots & 0 & & \end{pmatrix}.$$

Wir definieren die symmetrische und orthogonale Matrix

$$Q_k = \left(\begin{array}{c|c} I(k,k) & 0 \\ \hline 0 & S_k \end{array}\right)$$

mit der Reflexionsmatrix S_k, welche die Matrix $T^{(k)}$ auf

$$S_k T^{(k)} = \begin{pmatrix} c_{k+1} & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{pmatrix}$$

transformiert. Ihre j-te Spalte ($j = k+2, k+3, ..., n$) ergibt sich zu

$$(S_k T^{(k)})_j = ((I - f u u^T) T^{(k)})_j = T_j^{(k)} - f(u^T T_j^{(k)})\, u.$$

Damit berechnet man nun

$$A^{(k+1)} = Q_k A^{(k)} = \begin{pmatrix} c_1 & \dots & * & * & \dots & * \\ 0 & \ddots & \vdots & \vdots & & \vdots \\ \cdot & & c_k & * & \dots & * \\ \cdot & & 0 & c_{k+1} & \dots & * \\ \cdot & & 0 & 0 & & \\ \vdots & & \vdots & \vdots & T^{(k+1)} & \\ 0 & \dots & 0 & 0 & & \end{pmatrix}.$$

Es gilt $R = A^{(n-1)}$ und $Q = Q_0 Q_1 \cdot ... \cdot Q_{n-2}$ wegen $Q_k = Q_k^T = Q_k^{-1}$.
Die Analyse des Rechenaufwandes ergibt je $\mathcal{O}(2n^3/3)$ Multiplikationen und Additionen, also halb so viel Multiplikationen wie bei Givens-Rotationen.

Maple-Version

```
> house_r:=proc(n::posint,A::matrix)
  local Q,R,E,S,Qh,x,nx,c,e1,u,nu,nu1,i,j,k;

  # Eingangsparameter
  #  n        Dimension der Matrix
  #  A        Matrix
  #
  # Ergebnisse
  #  Q        Orthogonalmatrix
  #  R        Rechte Dreiecksmatrix

  # Initialisierungen
  Q:=evalm(array(identity,1..n,1..n));
  E:=evalm(Q);
  R:=evalm(A);

  # Schritt k=0
  if n>1 then
   x :=submatrix(R,1..n,[1]);
   nx:=norm(x,2);
   if evalf(nx)>0 then
    c:=-nx;
    if evalf(x[1,1])<0 then
     c:=nx;
    end if;
    e1:=submatrix(E,1..n,[1]);
    u :=evalm(x-c*e1);
    nu:=norm(u,2)^2;
     nu1:=rationalize(2/nu);
     S :=evalm(E-nu1*(u&*transpose(u)));
    # S :=evalm(E-2/nu*(u&*transpose(u)));

     for i from 1 by 1 to n do    # zur Vereinfachung von Formeln
      for j from 1 by 1 to n do
       S[i,j]:=expand(S[i,j]);
      end do;
     end do;

    Qh:=evalm(S);
    R :=simplify(evalm(Qh&*R));
    Q :=simplify(evalm(Q&*Qh));
   end if;
  end if;

  # Schritt k=1,2,...,n-2
  for k from 1 by 1 to n-2 do
   x :=submatrix(R,k+1..n,[k+1]);
   nx:=norm(x,2);
   if evalf(nx)>0 then
    c:=-nx;
    if evalf(x[1,1])<0 then
     c:=nx;
```

```
  end if;
  e1:=submatrix(E,k+1..n,[k+1]);
  u :=evalm(x-c*e1);
  nu:=norm(u,2)^2;
   nu1:=rationalize(2/nu);
   S :=simplify(evalm(submatrix(E,k+1..n,k+1..n)-nu1*(u&*transpose(u))));
  # S :=simplify(evalm(submatrix(E,k+1..n,k+1..n)-2/nu*(u&*transpose(u))));

   for i from 1 by 1 to n-k do   # zur Vereinfachung von Formeln
    for j from 1 by 1 to n-k do
     S[i,j]:=expand(S[i,j]);
    end do;
   end do;

  Qh:=evalm(concat(
   transpose(concat(submatrix(E,1..k,1..k),array(sparse,1..k,1..n-k))),
   transpose(concat(array(sparse,1..n-k,1..k),evalm(S)))
   ));

  # nicht
  # Qh:=[[submatrix(E,1..k,1..k),    array(sparse,1..k,1..n-k)],
  #      [array(sparse,1..n-k,1..k), evalm(S)]];

  R :=simplify(evalm(Qh&*R));
  Q :=simplify(evalm(Q&*Qh));
 end;

end do;
[Q,R];
end:


# Aufruf
# Initialisierungen: n, A

erg:=house_r(n,A);
Q:=simplify(evalm(erg[1]));
R:=simplify(evalm(erg[2]));

simplify(evalm(transpose(Q)&*Q));   # Orthonormalitaet
# evalm(transpose(Q)&*Q);           # nicht vereinfacht
evalf(evalm(Q));
evalf(evalm(R));
```

In der Umsetzung der Formeln in Maple sind einige Kommandos als zweite Variante stehengeblieben. Diese sollen bei einer symbolischen Rechnung in den Zwischenschritten bewirken, dass die stets umfangreicheren Ausdrücke vereinfacht werden. Dazu dienen z. B. die Befehle `simplify, expand, rationalize`.
Außerdem achte man auf die Art und Weise der Generierung der Matrix Qh.

Matlab-Version, Prozedur als m-File household_r.m

```
% household_r.m
% QR-Faktorisierung mit Householder-Reflexionen

function [Q,R] = household_r(n,A)

% Eingangsparameter
%  n        Dimension der Matrix
%  A        Matrix
%
% Ergebnisse
%  Q        Orthogonalmatrix
%  R        Rechte Dreiecksmatrix

Q = eye(n,n);
E = eye(n,n);
R = A;

% Schritt k=0
if (n>1)
 x = R(:,1);
 nx = norm(x,2);
 if (nx>0)
  c = -nx;
  if (x(1)<0)
   c = nx;
  end;
  e1 = E(:,1);
  u  = x-c*e1;
  nu = norm(u,2)^2;
  S  = E-2/nu*u*u';
  Qh = S;
  R  = Qh*R;
  Q  = Q*Qh;
 end;
end;

% Schritt k=1,2,...,n-2
for k=1:n-2
 x = R(k+1:n,k+1);
 nx = norm(x,2);
 if (nx>0)
```

```
  c = -nx;
  if (x(1)<0)
   c = nx;
  end;
  e1= E(k+1:n,k+1);
  u = x-c*e1;
  nu = norm(u,2)^2;
  S = E(k+1:n,k+1:n)-2/nu*u*u';
  Qh= [eye(k) zeros(k,n-k); zeros(n-k,k) S];
  R = Qh*R;
  Q = Q*Qh;
 end;
end;
% Ende Funktion househ_r


% Aufruf
% Initialisierungen: n, A
% m-File  househ_r  fuer QR-Faktorisierung

disp('m-File  househ_r  fuer QR-Faktorisierung')
disp('househ_r');
[Q,R] = househ_r(n,A)
A-Q*R
```

Die Beispielrechnungen können in den Arbeitsblättern nachvollzogen werden.
In den Maple-Versionen kann man in Abhängigkeit von den Ausgangsdaten eine exakte Berechnung (symbolische Auswertungen) bzw. Rechnungen in der GPA mit der Genauigkeit `Digits:=...` durchführen.

2.9 Turbo Pascal → Maple

Ein Plot einer Kurve oder Fläche lässt sich mit Maple bedeutend einfacher, eleganter und efektvoller machen als mit den Grafikkomponenten einer Hochsprache. Aber zuweilen hat man die Datenmengen zu einer Grafik aus einem Pascal- oder C-Programm und dann bietet es sich an, diese in Maple zu transferieren und dort geeignet mit den vielfältigen Möglichkeiten des CAS darzustellen.
Wir betrachten Grafikdateien in Turbo Pascal und ihre Einbeziehung in Maple-Kommandos für grafische Ausgaben.

2.9.1 Dateien mit Grafik in Turbo Pascal

Gegenstand der Illustrationen sind ebene und räumliche Kurven sowie Flächen im Raum.

- Ebene Kurve (auch Phasenkurve genannt)
  $P(t) = (x(t), y(t)),\ t \in [t_0, t_1],\ x(t), y(t)$ Koordinatenfunktionen,
- Raumkurve
  $P(t) = (x(t), y(t), z(t)),\ t \in [t_0, t_1],$
- Fläche
  $z = f(x, y)$ im rechteckigen Argumentbereich von $\mathbb{R}^2$.

Die Generierung mit gleichzeitiger Abspeicherung von $n + 1$ Kurvenpunkten erfolgt nach dem Muster

```
dt:=(t1-t0)/n;
t :=t0;
t1:=t1+dt*0.01;          { Sicherheitsschranke }

while t<=t1 do
  begin
    x:=...;
    y:=...;
    z:=...;

    { Abspeicherung des Punktes (x,y,z) in Datei }

    t:=t+dt;
  end;
```

Bei der 3D-Darstellung der Funktion $z = f(x, y)$ für das rechteckige Gebiet $[x_0, x_1] \times [y_0, y_1]$ der Argumente (x, y) mit einem $(m \times n)$-Gitter ist die Vorgehensweise so.

```
dx:=(x1-x0)/m;
x :=x0;
x1:=x1+dx*0.01;          { Sicherheitsschranke }
dy:=(y1-y0)/n;
y :=y0;
y1:=y1+dy*0.01;

{ Abspeicherung der Information (m+1,n+1,(m+1)(m+1)) in Datei }

{ zeilenweise Speicherung }
while x<=x1 do
  begin
    y:=y0;
    while y<=y1 do
      begin
       z:=f(x,y);

       { Abspeicherung des Tripels (x,y,z) in Datei }

        y:=y+dy;
      end;
    x:=x+dx;
  end;
```

Datenaufbereitung und erstelltes File:

```
m+1,n+1,(m+1)(n+1);
x[0],y[0],z; x[0],y[1],z; x[0],y[2],z; ... ;x[0],y[n],z;
x[1],y[0],z; x[1],y[1],z; x[1],y[2],z; ... ;x[1],y[n],z;
...
x[m],y[0],z; x[m],y[1],z; x[m],y[2],z; ... ;x[m],y[n],z
```

Es genügt für das Bild, alle Werte im Format *single* abzulegen.

Natürlich kann man die Grafik von TP für einen relativ bescheidenen Plot bemühen. Bei solchen Simulationen sind Manipulationen des Bildes wie Drehen, Strecken, Verfeinern oder Drucken möglich.
Wir beschränken uns aber hier auf die wenigen Grafikanteile und ein kleines Menü im Fall einer ebenen Kurve.

```
{$N+}
program Grafik_Phasenkurve_2;
{(C)  W.Neundorf IfMath TUI 2002
      P(t)=(x(t),y(t)) fuer t in [t0,t1] mit Datei/Grafik
      A9_K2.PAS                                          }

uses  crt,graph;

const bgi_pfad='C:\D\Neundorf\Maple4\tp_plot';

type  float=single;      { reicht fuer Grafik }
      floatdatei=text;   { file of float }
var   ch,v:char;
      punkte:floatdatei;
      a,b:float;

{ Definition der 2 Funktionen x und y der Phase }
procedure Kurve(v:char;t:float;var x,y:float);
begin
  { a,b,v global }
  { verschiedene Testkurven }
  case v of
  'E': begin  {Epizykloide}
        x:=(a+b)*cos(b/a*t)-b*cos((a+b)/a*t);
        y:=(a+b)*sin(b/a*t)-b*sin((a+b)/a*t);
       end;
  'H': begin  {Hypozykloide}
        x:=(a-b)*cos(b/a*t)+b*cos((a-b)/a*t);
        y:=(a-b)*sin(b/a*t)-b*sin((a-b)/a*t);
       end;
  end;
end;

{ Anlegen der Punktdatei }
procedure Schreiben(var punkte:floatdatei);
var t,t0,t1,dt,x,y:float;
    n:longint;
    name:string[80];
begin
  writeln;
  writeln('Art der Kurve:  (E)pizykloide / (H)ypozykloide');
  write(' v = '); readln(v);
  v:=upcase(v);
  writeln('Eingabe der Radien a,b>0 ');
  write(' a = '); readln(a);
  write(' b = '); readln(b);
  writeln('Eingabe der Intervallgrenzen  0<=t0<t1 ');
  write(' t0 = '); readln(t0);
  write(' t1 = '); readln(t1);
  write(' Kurvenpunkte (0..n mit n>0)  n = '); readln(n);
  writeln;
  write(' Name der zu erstellenden Datei = '); readln(name);
```

```
  delay(1000);
  writeln;
  writeln('*** Abspeicherung laeuft - Bitte warten ***');
  writeln;
  delay(2000);
  dt:=(t1-t0)/n;
  t :=t0;
  t1:=t1+dt*0.01;         { Sicherheitsschranke }

  assign(punkte,name);
  rewrite(punkte);        { Anlegen einer neuen Datei }

  while t<=t1 do
    begin
      Kurve(v,t,x,y);
      writeln(punkte,x:9:6,' ',y:9:6);  { Abspeicherung des Punktes }
      t:=t+dt;
    end;
  close(punkte);          { Schliessen der Datei }
  writeln('*** Abspeicherung durchgefuehrt ***');
  delay(2000);
end;

{ Lesen der Datei, Zeichnen der Kurve }
procedure Lesen_und_Zeichnen(var punkte:floatdatei);
var x,y,xmin,xmax,ymin,ymax,dx,dy:float;
    rand,io,xx,yy,graphdriver,graphmode:integer;
    i,l2:longint;
    name:string[80];
    ch:char;
begin
  repeat
    writeln;
    write('Name der Datei : ');
    readln(name);
    assign(punkte,name);
    {$I-}
    reset(punkte);
    {$I+}
    io:=IOResult;
    if io<>0 then
      begin
        writeln('Datei existiert nicht oder falscher Name');
        delay(3000);
      end;
  until io=0;
  {$I-}
    readln(punkte,x,y);
  {$I+}
  io:=IOResult;
  if io<>0 then
    begin
```

```
      writeln('*** Datei enthaelt keine Punkte ***');
      delay(4000);
      close(punkte);
      exit;
    end;
  l2:=1;
  writeln('Bestimmung der Extremwerte');
  xmin:=x; xmax:=x;
  ymin:=y; ymax:=y;
  while not eof(punkte) do
    begin
      inc(l2);
      readln(punkte,x,y);
      if x<xmin then xmin:=x;
      if x>xmax then xmax:=x;
      if y<ymin then ymin:=y;
      if y>ymax then ymax:=y;
    end;
  if xmax=xmin then xmax:=xmin+1;  { sinnvoll aufweiten }
  if ymax=ymin then ymax:=ymin+1;

  { Grafik initialisieren,  VGA-Grafik : 640x480 Punkte }
  graphdriver:=detect;
  initgraph(graphdriver,graphmode,bgi_pfad);

  rectangle(0,0,getmaxX,getmaxY);
  outtextxy(10,10,'Grafik der Phasenkurve (x(t),y(t)), t in [t0,t1]');
  delay(1000);
  rand:=40;       { Rand = 40 Pixel }
  dx:=(xmax-xmin)/(getmaxX-2*rand);
  dy:=(ymax-ymin)/(getmaxY-2*rand);
  reset(punkte);
  for i:=1 to l2 do
    begin
      readln(punkte,x,y);
      { Transformation Welt(x,y) --> BS(xx,yy) }
      xx:=trunc((x-xmin)/dx+rand);
      yy:=trunc(getmaxY-rand-(y-ymin)/dy);
      if (xx>=rand-1) and (xx<=getmaxX-rand+1) and
         (yy>=rand-1) and (yy<=getmaxY-rand+1) then putpixel(xx,yy,white);
    end;
  close(punkte);
  outtextxy(getmaxX-100,getmaxY-20,'KeyPressed');
  ch:=readkey;
  cleardevice;
  closegraph;
end;

begin
  repeat
    clrscr;
    writeln('Grafik einer Phasenkurve  (x(t),y(t)), t in [t0,t1]');
```

```
    writeln('================================================');
    writeln;
    writeln(' M e n u e');
    writeln;
    writeln('  (A)nlegen einer Punktdatei');
    writeln('  (L)esen der Datei und Zeichnen');
    writeln('  (E)nde des Programmes');
    writeln;
    write(' Wahl : ');
    readln(ch);
    ch:=upcase(ch);
    writeln('================================================');
    case ch of
     'A' : Schreiben(punkte);
     'L' : Lesen_und_Zeichnen(punkte);
    end
  until ch='E';
end.
```

Analog ist das Programm für eine räumliche Kurve.
Die Generierung der Flächenpunkte verläuft folgendermaßen.

```
{$N+}
program Datei_Flaeche;
{(C)  W.Neundorf IfMath TUI 2002
      P=(x,y,z=f(x,y)) fuer (x,y) in [x0,x1] x [y0,y1] mit Datei
      A9_F3.PAS                                                  }

uses  crt;

type  float=single;      { reicht fuer Grafik }
      floatdatei=text;   { file of float }
var   ch,v:char;
      flaeche:floatdatei;

{ Definition der Funktion z=f(x,y) }
function f(x,y:float):float;
begin
  f:=x*x*x-2*x*y*y;
end;

{ Anlegen der Punktdatei }
procedure Schreiben(var flaeche:floatdatei);
var x0,x1,dx,y0,y1,dy,x,y,z:float;
    m,n:longint;
    name:string[80];
begin
  writeln;
  writeln('Flaeche im 3D');
  writeln('Eingabe der Bereichsgrenzen  0<=x0<x1 ');
  write(' x0 = '); readln(x0);
```

```
  write(' x1 = '); readln(x1);
  write(' Kurvenpunkte in x-Richtung (0..m mit m>0)  m = '); readln(m);
  writeln('Eingabe der Bereichsgrenzen  0<=y0<y1 ');
  write(' y0 = '); readln(y0);
  write(' y1 = '); readln(y1);
  write(' Kurvenpunkte in y-Richtung (0..n mit n>0)  n = '); readln(n);
  writeln;
  write(' Name der zu erstellenden Datei = '); readln(name);
  delay(1000);
  writeln;
  writeln('*** Abspeicherung laeuft - Bitte warten ***');
  writeln;
  delay(2000);
  dx:=(x1-x0)/m;
  x :=x0;
  x1:=x1+dx*0.01;          { Sicherheitsschranke }
  dy:=(y1-y0)/n;
  y :=y0;
  y1:=y1+dy*0.01;

  assign(flaeche,name);
  rewrite(flaeche);        { Anlegen einer neuen Datei }

  writeln(flaeche,m+1,'  ',n+1,'  ',(m+1)*(n+1));
  while x<=x1 do
    begin
      y:=y0;
      while y<=y1 do
        begin
         z:=f(x,y);
         writeln(flaeche,x:9:6,' ',y:9:6,' ',z:9:6);
               { Abspeicherung des Punktes der Flaeche }
         y:=y+dy;
        end;
      x:=x+dx;
    end;
  close(flaeche);          { Schliessen der Datei }
  writeln('*** Abspeicherung durchgefuehrt ***');
  delay(2000);
end;

begin
  repeat
    clrscr;
    writeln('Datei zur Flaeche z=f(x,y), (x,y) in [x0,x1] x [y0,y1]');
    writeln('=====================================================');
    writeln;
    writeln(' M e n u e');
    writeln;
    writeln('  (A)nlegen einer Punktdatei');
    writeln('  (N) ...');
    writeln('  (E)nde des Programmes');
```

```
    writeln;
    write(' Wahl : ');
    readln(ch);
    ch:=upcase(ch);
    writeln('=================================================');
    case ch of
     'A' : Schreiben(flaeche);
     'N' : ;
    end;
  until ch='E';
end.
```

Dann finden wir unter anderem folgende Grafikdateien vor.
Einige von diesen sind per Hand generiert worden.

k2_01.dat, Punktdatei im $\mathbb{R}^2$, Polygonzug

```
 1.000000  0.000000
 1.000000  1.000000
 0.500000  2.000000
 0.000000  1.500000
-0.500000  1.000000
-0.750000  0.500000
```

ep1.dat, Punktdatei im $\mathbb{R}^2$, Epizykloide

```
 1.000000  0.000000
 1.002972  0.000156
 1.011832  0.001247
 ...
 1.000847  0.000024
```

hy1.dat, Punktdatei im $\mathbb{R}^2$, Hypozykloide

```
 4.000000  0.000000
 3.999512  0.000001
 3.998050  0.000007
 ...
-0.000098  3.989010
```

k3_01.dat, Punktdatei im $\mathbb{R}^3$, Raumkurve

```
 3.000000  0.000000  1.000000
 2.991574  0.049870  0.998750
 2.981307  0.099460  0.995004
 ...
 0.516166 -1.163229 -0.839061
```

k3_03.dat, Punktdatei im $\mathbb{R}^3$, Schraubenspirale

```
 3.000000  0.000000  0.000000
 2.976688  0.119211  0.060000
 2.942898  0.236568  0.120000
 ...
 0.023028 -0.098384 29.999924
```

surf2.dat, Datei zur Fläche im $\mathbb{R}^3$, "Zeltdach"

```
0  0  0
0  1  0
0  2  0
1  0  0
1  1  1
1  2  2
2  0  0
2  1  2
2  2  4
```

f3_01.dat, Datei zur Fläche im $\mathbb{R}^3$, "Affensattel"

```
11  21  231
-1.000000 -1.000000  1.000000
-1.000000 -0.900000  0.620000
-1.000000 -0.800000  0.280000
 ...
-1.000000  0.900000  0.620001
-1.000000  1.000000  1.000000
-0.800000 -1.000000  1.088000
-0.800000 -0.900000  0.784000
 ...
 1.000000  1.000000 -1.000000
```

list1.dat, Datei von z-Werten zur Fläche im $\mathbb{R}^3$ über Standardgitter, "Pyramide"

```
0  0  0  0
0  1  1  0
0  1  2  0
0  1  1  0
0  0  0  0
```

2.9.2 TP-Grafikdateien in Maple

Die Dateien aus dem vorherigen Abschnitt werden nun in Maple mit `readdata` eingelesen und durch geeignete `plot`-Kommandos in Bilder umgesetzt.

Der `readdata`-Befehl basiert auf der Spaltenstruktur der Datei und kann im Allgemeinen in seiner Langform `readdata(fileID,format,n)` n Spalten der Daten von einem gegebenen Format lesen. Fehlt die Option n, so wird die erste Spalte nur genommen.

Die hier verwendeten `plot`-Befehle sind

- `pointplot` für Kurven im $\mathbb{R}^2$,
- `pointplot3d` für Kurven im $\mathbb{R}^3$,
- `surfdata` und `listplot3d` für Flächen im $\mathbb{R}^3$.

Kommentare dazu und einige Erläuterungen zu Besonderheiten, z. B. wenn man bei einer ebenen Kurve nur die erste Spalte einliest, findet man im Arbeitsblatt.

Rechnungen in Maple (Datei: *a9tp1.mws*)

Verarbeitung der TP-Dateien durch Maple
Zeichnen von Kurven und Flaechen

```
> pfad:=‘C:/D/Neundorf/maple4/tp_plot/‘:
```

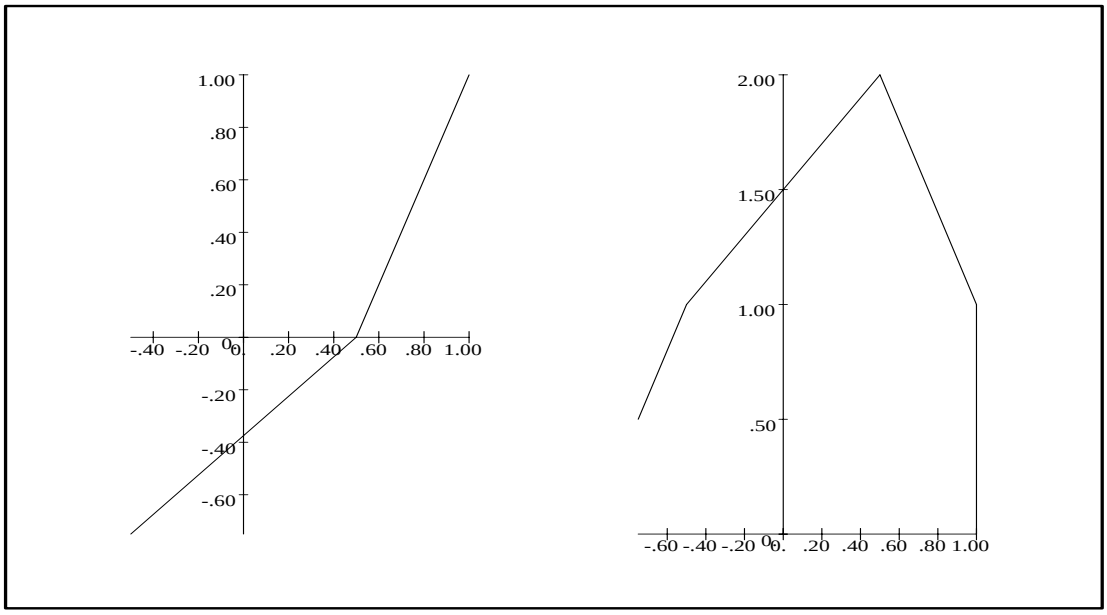
Kurven in 2D

```
k2_01.dat: Punkte P(x,y), Polygonzug
 1.000000  0.000000
 1.000000  1.000000
 0.500000  2.000000
 0.000000  1.500000
-0.500000  1.000000
-0.750000  0.500000
```

```
> p:=array(1..2,[]):
  name1:=‘k2_01.dat‘:
  datei1:=cat(pfad,name1):
  p[1]:=pointplot(readdata(datei1,float),style=line,axes=normal):
     # Lesen der 1. Spalte
     # Anzahl der Werte ist gerade, damit Paarebildung,
     # wenn ungerade, dann Fehler

  p[2]:=pointplot(readdata(datei1,float,2),style=line,axes=normal):
     # Lesen der Spalten 1,2
     # ohne Spaltenzahl = 1.Spalte

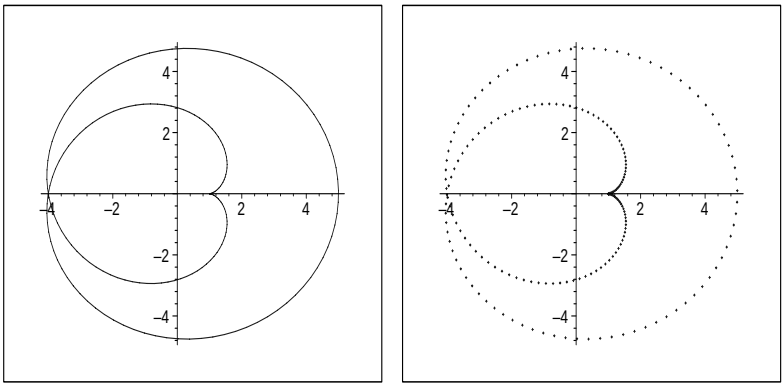
  plots[display](p);
```



ep1.dat: Epizykloide, Punkte P(x,y)
x:=(a+b)*cos(b/a*t)-b*cos((a+b)/a*t); y:=(a+b)*sin(b/a*t)-b*sin((a+b)/a*t);

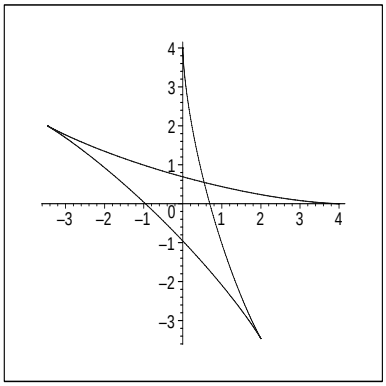
```
> name2:=‘ep1.dat‘:
  datei2:=cat(pfad,name2):
  pep2:=readdata(datei2,2):    # Lesen der Spalten 1,2
  pointplot(pep2,
       style=line,   # style<>line --> symbol=...
       # connect=true, symbol=box,
       color=blue,axes=normal);

> pointplot(pep2,
       # style=line, connect=true, symbol=box, scaling=constrained,
         color=blue,axes=normal);
```



hy1.dat: Hypozykloide, Punkte P(x,y)
x:=(a-b)*cos(b/a*t)+b*cos((a-b)/a*t); y:=(a-b)*sin(b/a*t)-b*sin((a-b)/a*t);

```
> name3:=‘hy1.dat‘:
  datei3:=cat(pfad,name3):
  pep3:=readdata(datei3,2):    # Lesen der Spalten 1,2
  pointplot(pep3,
       style=line,   # style<>line --> symbol=...
       # connect=true, symbol=box,
       color=blue,axes=normal);
```



Kurven in 3D

```
k3_01.dat: Punkte P(x,y,z), Raumkurve
 3.000000  0.000000  1.000000
 2.991574  0.049870  0.998750
 2.981307  0.099460  0.995004
 2.969214  0.148740  0.988771
 2.955311  0.197680  0.980067
 2.939617  0.246252  0.968912
 2.922151  0.294427  0.955337
 ...
```

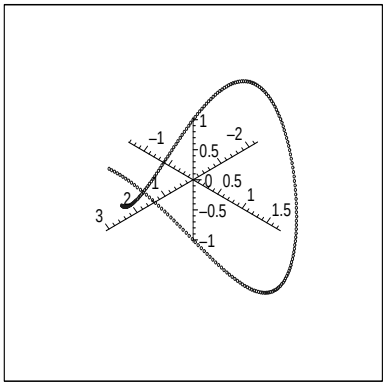
```
> name4:=‘k3_01.dat‘:
  datei4:=cat(pfad,name4):
  pep4:=readdata(datei4,3):    # Lesen der Spalten 1,2,3

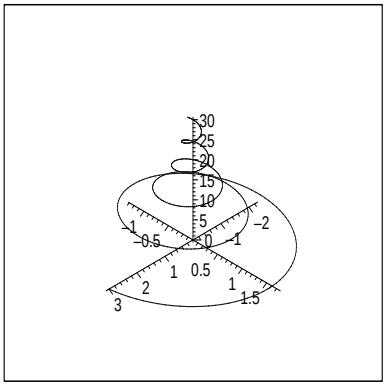
  pointplot3d(pep4,symbol=circle,color=blue,axes=normal);
```

```
k3_03.dat: Punkte P(x,y,z)
x:=3.0*exp(-0.1*t)*cos(t); y:=2.0*exp(-0.1*t)*sin(t); z:=t; 0<=t<=30
 3.000000  0.000000  0.000000
 2.976688  0.119211  0.060000
 2.942898  0.236568  0.120000
 2.898879  0.351672  0.180000
 2.844910  0.464131  0.240000
 2.781306  0.573573  0.300000
 2.708411  0.679636  0.360000
...
```

```
> name5:=‘k3_03.dat‘:
  datei5:=cat(pfad,name5):
  pep5:=readdata(datei5,3):

  pointplot3d(pep5,style=line,color=blue,axes=normal);
```





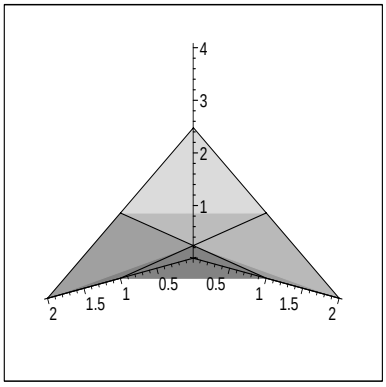
Flaechen in 3D

```
surf2.dat:  Liste von Werten (x,y,f(x,y)), "Zeltdach"
0  0  0, 0  1  0, 0  2  0,
1  0  0, 1  1  1, 1  2  2,
2  0  0, 2  1  2, 2  2  4
```

```
> name6:=‘surf2.dat‘:
  datei6:=cat(pfad,name6):
  pep6:=readdata(datei6,3);
  pep6[1..3];
     # Lesen von 3 Spalten zu jeweils 3 Zeilen aus Datei

  surfdata([pep6[1..3],pep6[4..6],pep6[7..9]],
     thickness=2,axes=normal,orientation=[45,75]); # list of lists
```

$$pep6 :=$$
$$[[0.,0.,0.],[0.,1.,0.],[0.,2.,0.],[1.,0.,0.],[1.,1.,1.],[1.,2.,2.],[2.,0.,0.],[2.,1.,2.],[2.,2.,4.]]$$
$$[[0.,\ 0.,\ 0.],\ [0.,\ 1.,\ 0.],\ [0.,\ 2.,\ 0.]]$$




```
f3_01.dat: Anzahl der x-Werte, Anzahl der y-Werte, ihr Produkt
und Liste von Werten
(x,y,f(x,y)), x=-1(0.2)1, y=-1(0.1)1,~f:=x^3-2xy^2, "Affensattel"
11  21  231
-1.000000 -1.000000  1.000000
-1.000000 -0.900000  0.620000
-1.000000 -0.800000  0.280000
-1.000000 -0.700000 -0.020000
-1.000000 -0.600000 -0.280000
-1.000000 -0.500000 -0.500000
. . .
-1.000000  1.000000  1.000000
-0.800000 -1.000000  1.088000
 . . .
```

```
> name7:='f3_01.dat':
  datei7:=cat(pfad,name7):
  pep7:=readdata(datei7,3):
  p:=pep7[1];
  m1:=round(p[1]);
  n1:=round(p[2]);
  pep71:=op(2..m1*n1+1,pep7):
  pep71[1];
  pep71[1][3];     # oder
  op(3,pep71[1]);

  pep72:=[seq([seq(pep71[k],k=(i-1)*n1+1..i*n1)],i=1..m1)]:
           # list of lists

  surfdata(pep72,axes=normal);
```

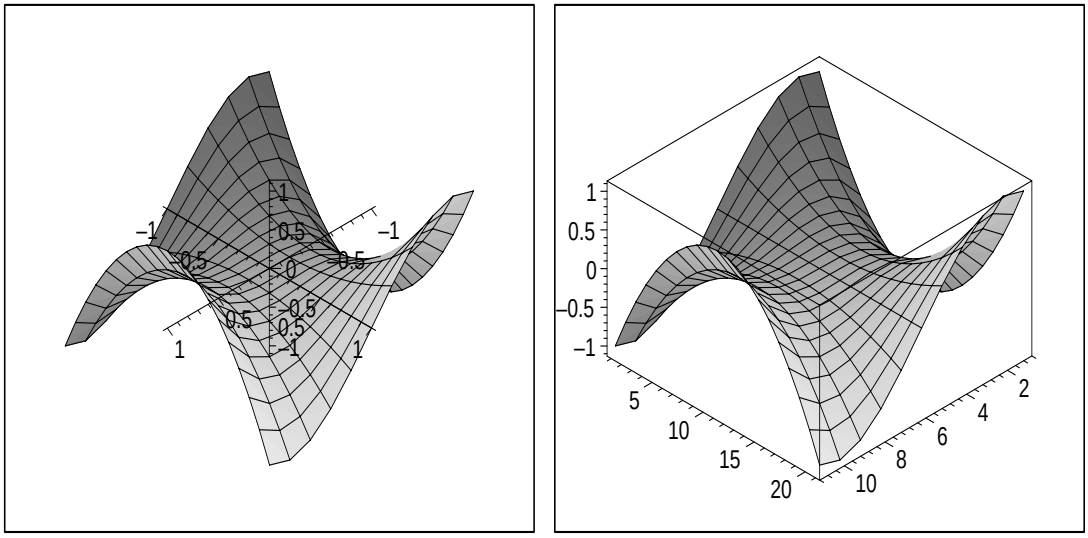
$$p := [11., 21., 231.]$$
$$m1 := 11$$
$$n1 := 21$$
$$[-1.000000, -1.000000, 1.000000]$$
$$1.000000$$
$$1.000000$$

Extrahieren der z-Werte (3. Operand der Tripel) und (x, y) als Standardgitter $n1 * m1$

```
> pep73:=[seq([seq(op(3,pep71[k]),k=(i-1)*n1+1..i*n1)],i=1..m1)]:
  pep73[1];

  listplot3d(pep73,axes=boxed);
```

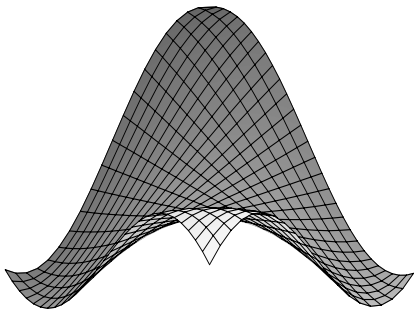
[1.000000, 0.620000, 0.280000, -0.020000, -0.280000, -0.500000, -0.680000, -0.820000, -0.920000, -0.980000, -1.000000, -0.980000, -0.920000, -0.820000, -0.680000, -0.500000, -0.280000, -0.020000, 0.280000, 0.620001, 1.000000]

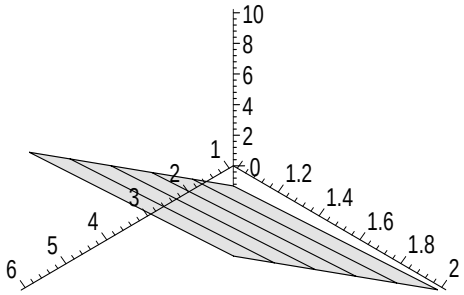


3D-Plot einer Liste von Listen numerischer Werte

```
> # ueber 30*20-Standardgitter die Funktionswerte zeichnen
  listplot3d([seq([seq(sin((i-15)*(j-10)/Pi/20),i=1..30)],j=1..20)]);

> # ueber 2*6-Standardgitter die Funktionswerte zeichnen
  listplot3d([[-1,0],[1,2],[3,4],[5,6],[7,8],[9,10]],
             color=yellow,axes=normal);
```



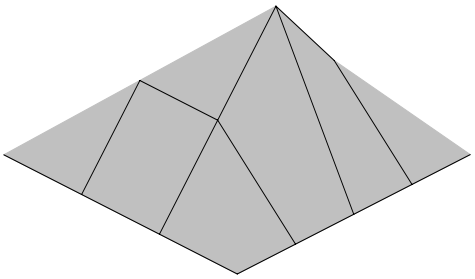



```
list1.dat: Matrix von Werten f(x(1..m),y(1..n)), "Pyramide"
 0 0 0 0
 0 1 1 0
 0 1 2 0
 0 1 1 0
 0 0 0 0
```

```
> name8:=‘list1.dat‘:
  datei8:=cat(pfad,name8):
  pep8:=readdata(datei8,float,4);
  pep8[1];  # 1. Zeile

  listplot3d(pep8,color=gray);  # shading=z
```

$$pep8 := [[0.,\ 0.,\ 0.,\ 0.],\ [0.,\ 1.,\ 1.,\ 0.],\ [0.,\ 1.,\ 2.,\ 0.],\ [0.,\ 1.,\ 1.,\ 0.],\ [0.,\ 0.,\ 0.,\ 0.]]$$
$$[0.,\ 0.,\ 0.,\ 0.]$$



2.10 Dateiarbeit in Maple

In den hier gezeigten Varianten der Dateiarbeit in Maple verwenden wir die Kommandos/Kombinationen

- `save, read`,
- `writedata, readdata`,
- `fprintf, fscanf`,

letztere in Verbindung mit `fopen, fclose`.
Dabei spielt die Spaltenstruktur der Datei eine wichtige Rolle, so dass das Lesen von ausgewählten Spalten möglich ist.

Detailinformationen kann der Anwender aus der Online-Hilfe entnehmen. Ansonsten sind im Arbeitsblatt entsprechende Kommentare angebracht.

Rechnungen in Maple (Datei: *file1.mws*)

Erzeugung einiger Daten, u. a. mit Zufallszahlengenerator

```
> die:=rand(10):   # Zufallszahlen aus 0..10-1
  die();

  n:=5;
  seq(die(),i=1..n);
```

$$n := 5$$

$$1$$

$$0,\ 7,\ 3,\ 6,\ 8$$

```
> An1:=matrix(n,1,i->i);
  a:=vector(n,[seq(die(),i=1..n)]);
  A:=augment(An1,a);    # A:=concat(An1,a);
```

$$An1 := \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

$$a := [3, 7, 0, 4, 5]$$

$$A := \begin{bmatrix} 1 & 3 \\ 2 & 7 \\ 3 & 0 \\ 4 & 4 \\ 5 & 5 \end{bmatrix}$$

```
> die:=rand(10):
  A:=matrix(5,2,(i,j)->if j=1 then i else die() end if);
```

$$A := \begin{bmatrix} 1 & 8 \\ 2 & 6 \\ 3 & 9 \\ 4 & 3 \\ 5 & 1 \end{bmatrix}$$

Information zu `readlib`

```
> # readlib();
  # Read a library file to define a specified name.
  # This command is obsolete. Procedures are automatically invoked
  # from the library without the need of a readlib call

  # betrifft auch
  # readlib(readdata): readlib(writedata):
```

Speichern einer Matrix

```
> pfad:=‘C:/D/Neundorf/Maple4/File/‘:
```

1. Variante

Falls Datei unter den Namen schon existiert, dann wird sie ueberschrieben.

```
> # Langform
  # save A,‘C:/D/Neundorf/Maple4/File/werte1.txt‘;
  # save A,"C:/D/Neundorf/Maple4/File/werte1.txt";

  # eleganter
  name1:=‘werte1.txt‘:
  datei1:=cat(pfad,name1);
  datei11:=cat(pfad,‘werte1.m‘);
  datei12:=cat(pfad,‘werte1.dat‘);

  save A,datei1;  # written into the file, Maple language format is used
  save A,datei11; # written into the file, Maple internal format is used
  save B,datei12; # B ist atomar
  save A;         # Fehler
```

$$datei1 := C:/D/Neundorf/Maple4/File/werte1.txt$$
$$datei11 := C:/D/Neundorf/Maple4/File/werte1.m$$
$$datei12 := C:/D/Neundorf/Maple4/File/werte1.dat$$

```
Warning, unassigned variable ‘B‘ in save statement

Error, must specify names to save
```

Speicherergebnisse sind

ASCII-Datei werte1.txt mit Variablennamen A

```
A := array(1 .. 5, 1 .. 2,[(2, 2)=7,(4, 1)=4,(4, 2)=4,(3, 1)=3,(1, 2)=3,
                           (2, 1)=2,(1, 1)=1,(5, 2)=5,(5, 1)=5,(3, 2)=0]);
```

Binaer-Datei werte1.m mit Variablennamen A

```
M7R0
I"A=6"6$;""""""&;F’""#E\[l+6$""%F’F-6$F(F*F’6$F(F’F(6$""$F*""*6$F*F’F*6$F’F*"")6
$F*F*""’6$F-F*F16$F1F’F16$F’F’F’F$
```

ASCII-Datei werte1.dat mit Variablennamen B (keine Werte)

```
B := ’B’;
```

Lesen der Datei mit Anzeige des Inhalts

```
> # read "C:/D/Neundorf/Maple4/File/werte1.txt";
  # read ‘C:/D/Neundorf/Maple4/File/werte1.txt‘;

  name1:=‘werte1.txt‘:
  datei1:=cat(pfad,name1);
  read datei1;
```

$$datei1 := C:/D/Neundorf/Maple4/File/werte1.txt$$

$$A := \begin{bmatrix} 1 & 8 \\ 2 & 6 \\ 3 & 9 \\ 4 & 3 \\ 5 & 1 \end{bmatrix}$$

```
> print(A);
```

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

```
> xd:=col(A,1);
  yd:=col(A,2);
```

$$xd := [1, 2, 3, 4, 5]$$
$$yd := [8, 6, 9, 3, 1]$$

2. Variante

```
> # fd := fopen("C:/D/Neundorf/Maple4/File/werte2.txt",WRITE,TEXT):
  # fd := fopen(‘C:/D/Neundorf/Maple4/File/werte2.txt‘,WRITE,TEXT):

  name2:=‘werte2.txt‘:
  datei2:=cat(pfad,name2);
  fd := fopen(datei2,WRITE,TEXT):
                          # Explicitly open the file of type TEXT
  print(A);
  writedata(terminal,A); # Printed on the user’s terminal
  writedata(fd,A);       # Write the matrix to file
  fclose(fd);            # Close the file
```

datei2 := C:/D/Neundorf/Maple4/File/werte2.txt

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

```
1          8
2          6
3          9
4          3
5          1
```

Speicherergebnis ist ASCII-Datei werte2.txt ohne Variablennamen

1 8
2 6
3 9
4 3
5 1

```
> datei2:=cat(pfad,name2);
  fd := fopen(datei2,APPEND,TEXT):
                          # Explicitly open the file of type TEXT
                          # to append something
  writedata(fd,A);       # Write the matrix
  fclose(fd);            # Close the file
```

datei2 := C:/D/Neundorf/Maple4/File/werte2.txt

Speicherergebnis ist ergaenzte ASCII-Datei werte2.txt ohne Variablennamen

1 8
.. ..
(Daten zweimal hintereinander)

Lesen der Datei ganz oder teilweise mit Anzeige des Inhalts

```
> # readdata("C:/D/Neundorf/Maple4/File/werte2.txt",2);
  # readdata(‘C:/D/Neundorf/Maple4/File/werte2.txt‘,2);
  readdata(datei2,2);          # read the first two columns of the data
  ll2:=readdata(datei2,2);     # read the first two columns of the data
```

$$[[1., 8.], [2., 6.], [3., 9.], [4., 3.], [5., 1.], [1., 8.],[2., 6.], [3., 9.], [4., 3.], [5., 1.]]$$

$$ll2 := [[1., 8.], [2., 6.], [3., 9.], [4., 3.], [5., 1.], [1., 8.], [2., 6.], [3., 9.], [4., 3.], [5., 1.]]$$

```
> nops(ll2);
  op(ll2);    # op(1..nops(ll2),ll2)
  op(ll2)[1]; op(ll2)[1][1]; op(ll2)[1][2];
  seq(op(ll2)[i][1],i=1..nops(ll2));
  seq(op(ll2)[i][2],i=1..nops(ll2));
```

$$10$$

$$[1., 8.], [2., 6.], [3., 9.], [4., 3.], [5., 1.], [1., 8.], [2., 6.], [3., 9.], [4., 3.], [5., 1.]$$

$$[1., 8.]$$

$$1.$$

$$8.$$

$$1., 2., 3., 4., 5., 1., 2., 3., 4., 5.$$

$$8., 6., 9., 3., 1., 8., 6., 9., 3., 1.$$

```
> readdata(datei2,1);          # read the first column of the data
  ll1:=readdata(datei2,1);     # read the first column of the data
```

$$[1., 2., 3., 4., 5., 1., 2., 3., 4., 5.]$$

$$ll1 := [1., 2., 3., 4., 5., 1., 2., 3., 4., 5.]$$

```
> nops(ll1);
  op(ll1);    # op(1..nops(ll1),ll1)
```

$$10$$

$$1., 2., 3., 4., 5., 1., 2., 3., 4., 5.$$

```
> readdata(datei2);            # read the first column of the data
```

$$[1., 2., 3., 4., 5., 1., 2., 3., 4., 5.]$$

```
> xd:=[seq(op(ll2)[i][1],i=1..nops(ll2))];
  yd:=[seq(op(ll2)[i][2],i=1..nops(ll2))];
```

$$xd := [1., 2., 3., 4., 5., 1., 2., 3., 4., 5.]$$

$$yd := [8., 6., 9., 3., 1., 8., 6., 9., 3., 1.]$$

3. Variante

```
> # fd := fopen("C:/D/Neundorf/Maple4/File/werte3.txt",WRITE,TEXT):
  # fd := fopen(‘C:/D/Neundorf/Maple4/File/werte3.txt‘,WRITE,TEXT):

  name3:=‘werte3.txt‘:
  datei3:=cat(pfad,name3);
  fd := fopen(datei3,WRITE,TEXT):
                          # Explicitly open the file of type TEXT
```

$$datei3 := C:/D/Neundorf/Maple4/File/werte3.txt$$

```
> f:=x->sin(x)*exp(-x);
  g:=x->x*cos(x);

  for i from 0 to 20 do
   x:=i/10;
   fprintf(fd,‘%4.2f  %+.10e  %+.10e\n‘,x,f(x),g(x));
  end do:
  fclose(fd);
```

$$f := x \rightarrow \sin(x)\, e^{(-x)}$$
$$g := x \rightarrow x\, \cos(x)$$

Speicherergebnis ist ASCII-Datei werte3.txt

```
 0.00  +0.0000000000e-01  +0.0000000000e-01
  .10  +9.0333010950e-02  +9.9500416530e-02
  .20  +1.6265669080e-01  +1.9601331560e-01
  .30  +2.1892675370e-01  +2.8660094670e-01
  .40  +2.6103492110e-01  +3.6842439760e-01
  .50  +2.9078628820e-01  +4.3879128100e-01
  .60  +3.0988235960e-01  +4.9520136890e-01
  .70  +3.1990903590e-01  +5.3538953110e-01
  .80  +3.2232886920e-01  +5.5736536740e-01
  .90  +3.1847695510e-01  +5.5944897150e-01
 1.00  +3.0955987570e-01  +5.4030230590e-01
 1.10  +2.9665715980e-01  +4.9895573350e-01
 1.20  +2.8072477800e-01  +4.3482930540e-01
 1.30  +2.6260023990e-01  +3.4774847720e-01
 1.40  +2.4300891150e-01  +2.3795400010e-01
 1.50  +2.2257121610e-01  +1.0610580250e-01
 1.60  +2.0181042990e-01  -4.6719235680e-02
 1.70  +1.8116082230e-01  -2.1903564030e-01
 1.80  +1.6097593070e-01  -4.0896377050e-01
 1.90  +1.4153679750e-01  -6.1425017710e-01
 2.00  +1.2306002480e-01  -8.3229367300e-01
```

Lesen von Datei mit Formatangabe

fscanf - parses expressions from a file or pipe based on a format string

```
> # Lesen der 3 Spalten der Datei werte3.txt und Kontrollausgaben
  # Ergebnis: Liste von Listen von Tripeln
  ls123:=[seq([0,0,0],i=1..21)]:
  fd := fopen(datei3,READ):
  for i from 0 to 20 do
   ls123[i+1]:= fscanf(fd,‘%f%e%e\n‘);
  end do:
  ls123[1];
  ls123[21];
  ls123;
  fclose(fd);
```

[0., 0., 0.]

[2.00, 0.1230600248, -0.8322936730]

[[0., 0., 0.], [0.10, 0.09033301095, 0.09950041653], [0.20, 0.1626566908, 0.1960133156], [0.30, 0.2189267537, 0.2866009467], [0.40, 0.2610349211, 0.3684243976], [0.50, 0.2907862882, 0.4387912810], [0.60, 0.3098823596, 0.4952013689], [0.70, 0.3199090359, 0.5353895311], [0.80, 0.3223288692, 0.5573653674], [0.90, 0.3184769551, 0.5594489715], [1.00, 0.3095598757, 0.5403023059], [1.10, 0.2966571598, 0.4989557335], [1.20, 0.2807247780, 0.4348293054], [1.30, 0.2626002399, 0.3477484772], [1.40, 0.2430089115, 0.2379540001], [1.50, 0.2225712161, 0.1061058025], [1.60, 0.2018104299, -0.04671923568], [1.70, 0.1811608223, -0.2190356403], [1.80, 0.1609759307, -0.4089637705], [1.90, 0.1415367975, -0.6142501771], [2.00, 0.1230600248, -0.8322936730]]

```
> # Lesen der ersten 2 Spalten der Datei werte3.txt
  ls12:=[seq([0,0],i=1..21)]:
  fd := fopen(datei3,READ):
  for i from 0 to 20 do
   ls12[i+1]:= fscanf(fd,‘%f%e%*e\n‘);
     # The optional * in the format string indicates
     # that the object is to be scanned, but not returned
     # as part of the result (that is, it is discarded).
  end do:
  ls12[1];
  ls12[21];
  ls12;
  fclose(fd);
```

[0., 0.]

[2.00, 0.1230600248]

[[0., 0.], [0.10, 0.09033301095], [0.20, 0.1626566908], [0.30, 0.2189267537], [0.40, 0.2610349211], [0.50, 0.2907862882], [0.60, 0.3098823596], [0.70, 0.3199090359], [0.80, 0.3223288692], [0.90, 0.3184769551], [1.00, 0.3095598757], [1.10, 0.2966571598], [1.20, 0.2807247780],

[1.30, 0.2626002399], [1.40, 0.2430089115], [1.50, 0.2225712161],
[1.60, 0.2018104299], [1.70, 0.1811608223], [1.80, 0.1609759307],
[1.90, 0.1415367975], [2.00, 0.1230600248]]

```
> # Lesen der 1. und 3. Spalte der Datei
  ls13:=[seq([0,0],i=1..21)]:
  fd := fopen(datei3,READ):
  for i from 0 to 20 do
   ls13[i+1]:= fscanf(fd,‘%f%*e%e\n‘);
  end do:
  ls13[1]; ls13[21];
  ls13;
  fclose(fd);
```

[0., 0.]

[2.00, -0.8322936730]

[[0., 0.], [0.10, 0.09950041653], [0.20, 0.1960133156], [0.30, 0.2866009467],
[0.40, 0.3684243976], [0.50, 0.4387912810], [0.60, 0.4952013689],
[0.70, 0.5353895311], [0.80, 0.5573653674], [0.90, 0.5594489715],
[1.00, 0.5403023059], [1.10, 0.4989557335], [1.20, 0.4348293054],
[1.30, 0.3477484772], [1.40, 0.2379540001], [1.50, 0.1061058025],
[1.60, -0.04671923568], [1.70, -0.2190356403], [1.80, -0.4089637705],
[1.90, -0.6142501771], [2.00, -0.8322936730]]

```
> # Lesen der Spalten einzeln
  # Man bemerke den Unterschied zwischen den Ergebnissen
  # uu (Liste von Listen mit Wert)  und  xx (Liste von Werten)

  ls1:=[seq(0,i=1..21)];
  fd := fopen(datei3,READ):
  for i from 0 to 20 do
   ls1[i+1]:= fscanf(fd,‘%f%*e%*e\n‘);
  end do:
  uu:=ls1;
  fclose(fd);

  fd := fopen(datei3,READ):
  for i from 0 to 20 do
   ls1[i+1]:= op(fscanf(fd,‘%f%*e%*e\n‘));
  end do:
  xx:=ls1;
  fclose(fd);

  fd := fopen(datei3,READ):
  for i from 0 to 20 do
   ls1[i+1]:= op(fscanf(fd,‘%*f%e%*e\n‘));
  end do:
  ff:=ls1;
  fclose(fd);
```

```
fd := fopen(datei3,READ):
for i from 0 to 20 do
 ls1[i+1]:= op(fscanf(fd,‘%*f%*e%e\n‘));
end do:
ls1[1],ls1[2];
gg:=ls1;

# am Dateiende und weiterlesen --> 0
for i from 0 to 20 do
 ls1[i+1]:= op(fscanf(fd,‘%*f%*e%e\n‘));
end do:
ls1[1],ls1[2];
hh:=ls1;
fclose(fd);
```

$$ls1 := [\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\ ]$$

$$uu := [[0.], [0.10], [0.20], [0.30], [0.40], [0.50], [0.60], [0.70], [0.80], [0.90], [1.00], [1.10], [1.20], [1.30], [1.40], [1.50], [1.60], [1.70], [1.80], [1.90], [2.00]]$$

$$xx := [0., 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00, 1.10, 1.20, 1.30, 1.40, 1.50, 1.60, 1.70, 1.80, 1.90, 2.00]$$

$$ff := [0., 0.09033301095, 0.1626566908, 0.2189267537, 0.2610349211, 0.2907862882, 0.3098823596, 0.3199090359, 0.3223288692, 0.3184769551, 0.3095598757, 0.2966571598, 0.2807247780, 0.2626002399, 0.2430089115, 0.2225712161, 0.2018104299, 0.1811608223, 0.1609759307, 0.1415367975, 0.1230600248]$$

$$0., 0.09950041653$$

$$gg := [0., 0.09950041653, 0.1960133156, 0.2866009467, 0.3684243976, 0.4387912810, 0.4952013689, 0.5353895311, 0.5573653674, 0.5594489715, 0.5403023059, 0.4989557335, 0.4348293054, 0.3477484772, 0.2379540001, 0.1061058025, -0.04671923568, -0.2190356403, -0.4089637705, -0.6142501771, -0.8322936730]$$

$$0, 0$$

$$hh := [\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\ ]$$

Verbinden von Listen bzw. Vektoren miteinander durch `zip`-Kommando

```
> Pkte1:=zip((x,y)->[x,y],xx,ff):      # wie ls12
  Pkte2:=zip((x,y)->[x,y],xx,gg):      # wie ls13
```

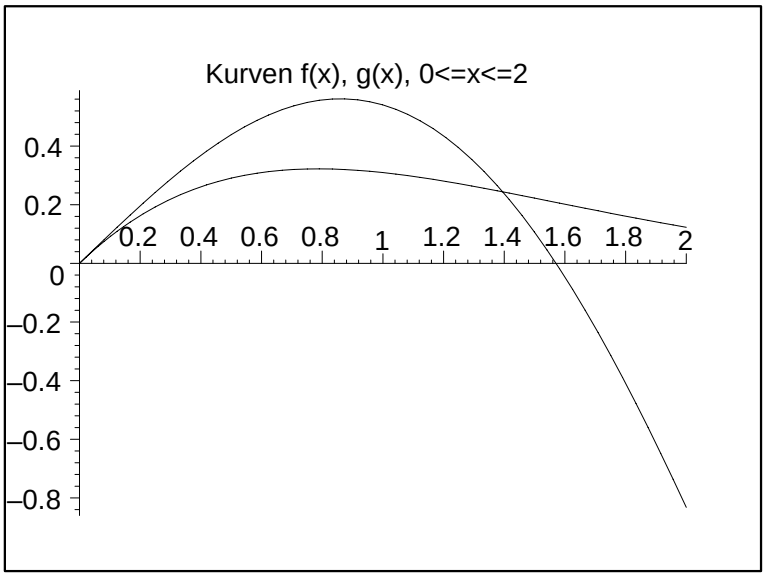
Grafik

```
> x:='x':
  f(x);
  g(x);
```

$$\sin(x)\, e^{(-x)}$$

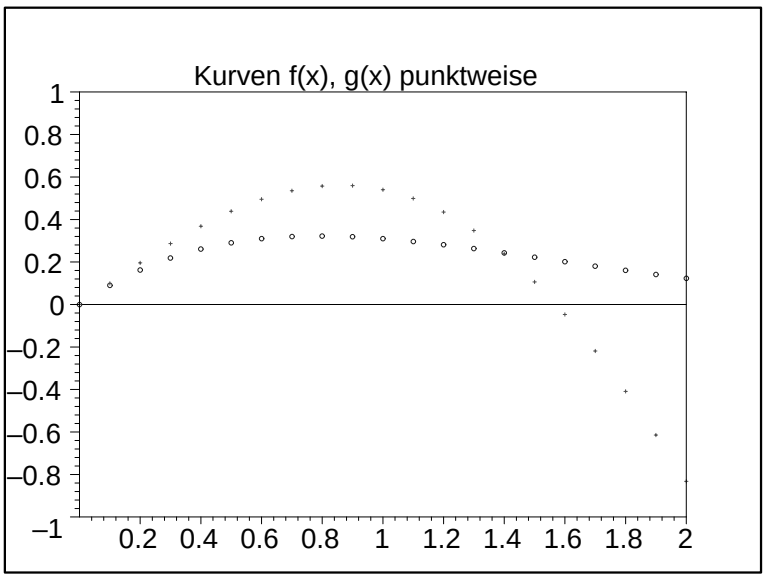
$$x\, \cos(x)$$

```
> plot([f,g],0..2,color=black,title=‘Kurven f(x), g(x), 0<=x<=2‘);
```




```
> Stil1:=style=POINT,symbol=circle:
  Stil2:=style=POINT,symbol=cross:
  PlotOpts:=scaling=unconstrained,axes=boxed:
  p:=plot(Pkte1,Stil1,PlotOpts,color=blue):
  q:=plot(Pkte2,Stil2,PlotOpts,color=red):
  r:=plot(0,0..2,color=black):    # x-Achse

  plots[display]({p,q,r},view=[0..2,-1..1],
                 title=‘Kurven f(x), g(x) punktweise‘);
```



4. Variante, Komma zwischen den Daten in einer Zeile

```
> # fd := fopen("C:/D/Neundorf/Maple4/File/werte4.txt",WRITE,TEXT):
  # fd := fopen(‘C:/D/Neundorf/Maple4/File/werte4.txt‘,WRITE,TEXT):

  name4:=‘werte4.txt‘:
  datei4:=cat(pfad,name4);
  fd := fopen(datei4,WRITE,TEXT):
                          # Explicitly open the file of type TEXT
```

$$datei4 := C:/D/Neundorf/Maple4/File/werte4.txt$$

```
> h:=x->sin(10*x)*exp(-x^2);

  for i from 0 to 10 do
   x:=i/10;
   fprintf(fd,‘%4.2f,  %+.10e\n‘,x,h(x));
  end do:
  fclose(fd);
```

$$h := x \to \sin(10\,x)\,e^{(-x^2)}$$

Speicherergebnis ist ASCII-Datei werte4.txt

```
 0.00,  +0.0000000000e-01
  .10,  +8.3309820860e-01
  .20,  +8.7364336480e-01
  .30,  +1.2897397630e-01
  .40,  -6.4490454590e-01
  .50,  -7.4681097610e-01
  .60,  -1.9494157820e-01
  .70,  +4.0248733100e-01
  .80,  +5.2168110810e-01
  .90,  +1.8333423240e-01
 1.00,  -2.0013418230e-01
```

```
> # Lesen der 2 Spalten mit Uebergehen des Komma
  # durch die ausblendende Option * im Leseformat
  ls4:=[seq([0,0],i=1..11)]:
  fd := fopen(datei4,READ):
  for i from 0 to 10 do
   ls4[i+1]:= fscanf(fd,‘%f%*s%e\n‘);
  end do:
  ls4;
  fclose(fd);
```

[[0., 0.], [0.10, 0.8330982086], [0.20, 0.8736433648], [0.30, 0.1289739763], [0.40, -0.6449045459], [0.50, -0.7468109761], [0.60, -0.1949415782], [0.70, 0.4024873310], [0.80, 0.5216811081], [0.90, 0.1833342324], [1.00, -0.2001341823]]

2.11 Maple → Matlab

Nutzung von Matlab-Möglichkeiten in Maple und auch umgekehrt ist sehr vielfältig. Zum ersten Teil wollen wir auf drei Aspekte eingehen.

- Überladung von Funktionen (overloaded functions),
- Parameterkonzept von Matlab Funktionen, insbesondere in der Maple-Matlab Verbindung,
- nutzerdefinierte Matlab Funktionen unter Maple.

Die Darstellungen verwenden also die CAS Maple und Matlab und werden ergänzt durch geeignete Beispiele, wobei bei numerischen Problemen wie QR-Faktorisierung, Ausgleichsrechnung oder Nullstellenbestimmung auch einige Informationen zum mathematischen Hintergrund erfolgen.

Rechnungen in Maple und Matlab (Datei: *m2m1.mws*)

Wenn es "Verstaendigungsschwierigkeiten" zwischen Maple und Matlab gibt, so lese man nach unter:

Matlab (Introduction to the Matlab package) oder Configuring a Computer for Matlab.

```
> restart:
  interface(warnlevel=0):  #  0 = suppress all warnings
  with(plots):
```

Module im Maple-Paket linalg

```
> with(linalg);
```

[$BlockDiagonal, GramSchmidt, JordanBlock, LUdecomp, QRdecomp, Wronskian,$
$addcol, addrow, adj, adjoint, angle, augment, backsub, band, basis, bezout,$
$blockmatrix, charmat, charpoly, cholesky, col, coldim, colspace, colspan,$
$companion, concat, cond, copyinto, crossprod, curl, definite, delcols, delrows,$
$det, diag, diverge, dotprod, eigenvals, eigenvalues, eigenvectors, eigenvects,$
$entermatrix, equal, exponential, extend, ffgausselim, fibonacci, forwardsub,$
$frobenius, gausselim, gaussjord, geneqns, genmatrix, grad, hadamard, hermite,$
$hessian, hilbert, htranspose, ihermite, indexfunc, innerprod, intbasis, inverse,$
$ismith, issimilar, iszero, jacobian, jordan, kernel, laplacian, leastsqrs,$
$linsolve, matadd, matrix, minor, minpoly, mulcol, mulrow, multiply, norm,$
$normalize, nullspace, orthog, permanent, pivot, potential, randmatrix, randvector,$
$rank, ratform, row, rowdim, rowspace, rowspan, rref, scalarmul, singularvals,$
$smith, stackmatrix, submatrix, subvector, sumbasis, swapcol, swaprow, sylvester,$
$toeplitz, trace, transpose, vandermonde, vecpotent, vectdim, vector, wronskian$]

The Maple-Matlab link works with Matlab 6 on most platforms that have Maple and Matlab versions. To determine if the link works on a platform, use the Maple command 'Matlab/valid_os'()

```
> 'Matlab/valid\_os'();
  # open a Matlab session and links to the current Maple session
  Matlab[openlink]();
```

$$true$$

2.11.1 Überladung von Funktionen

The Matlab package "overloads" some Maple's symbolic functions in some packages, using the same function name. So you find numeric version of the overloaded functions for example in the linalg package det or transpose.

Then use the online Help for the version. There are several matching topics. Try one of the following:

- linalg,det
- Matlab,det
- LinearAlgebra,Determinant

There are several matching topics. Try one of the following:

- linalg,transpose
- Matlab,transpose
- ListTools,Transpose
- LinearAlgebra,Transpose

To use a Matlab function in Maple, which lets you access all Matlab package functions from within Maple.

```
> with(Matlab);
```

$$[chol, closelink, defined, det, dimensions, eig, evalM, fft, getvar, inv, lu, ode45, openlink, qr, setvar, size, square, transpose]$$

To avoid problems, make then long Maple statements to choose the correct commands from Maple or Matlab

```
linalg[det](A)
Matlab[det](A)   or  det(A)
```

respectively

```
linalg[transpose](A)
Matlab[transpose](A)    or  transpose(A)
```

Beispiel: Tridiagonalmatrix, Bandbreite=3
Ueberladungsfunktionen `det` und `transpose`

```
> n:=4:
  # Variante 1
  A:=evalm(2*diag(1$n)):     # A:=2*Matrix(n,n,shape=identity);
  A:=matrix(n,n,(i,j)->`if`(abs(i-j)=1,-1,A[i,j])):

  # Variante 2
  A:=matrix(n,n,(i,j)->if i=j then 2 elif abs(i-j)=1 then -1 else 0 end if):

  # Variante 3
  A:=band([-1,2,-1],n):

  # Variante 4
  A:=matrix(n,n,[[ 2,-1, 0, 0],
                 [-1, 2,-1, 0],
                 [ 0,-1, 2,-1],
                 [ 0, 0,-1, 2]]);
```

$$A := \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

```
> # Maple
  linalg[det](A);
  An:=evalm(A): An[1,1]:=1.0*An[1,1]:
  linalg[det](An);

  # Matlab (GPA)
  Matlab[det](A);   # det(A);
```

$$5$$
$$5.0$$
$$5.$$

```
> # Maple
  linalg[transpose](A);

  # Matlab (GPA)
  Matlab[transpose](A);   # transpose(A);
```

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$
$$\begin{bmatrix} 2. & -1. & 0. & 0. \\ -1. & 2. & -1. & 0. \\ 0. & -1. & 2. & -1. \\ 0. & 0. & -1. & 2. \end{bmatrix}$$

2.11.2 Parameterkonzept von Matlab Funktionen

Anwendung in Matlab Sitzung oder in Maple-Matlab Verbindung

Matlab session

```
Matlab Help by  ->  help qr

 qr   Orthogonal-triangular decomposition

  qr(A)           is the output of LAPACK'S DGEQRF or ZGEQRF routine.
                  triu(qr(A)) is R.
  [Q,R] = qr(A)   produces an upper triangular matrix R of the same
                  dimension as A and a unitary matrix Q, so that A=Q*R.
  [Q,R,P] = qr(A) produces a permutation matrix P, an upper triangular R
                  and a unitary Q so that A*P = Q*R.
                  The column permutation P is chosen so that
                  abs(diag(R)) is decreasing.
  .........
```

Die Rechnungen in Matlab erfolgen im Format *double* und 15..16 Dezimalstellen werden bei `format long` bzw. `format long e` angezeigt.

Maple session

```
Maple Help  ->  online Help  qr

Matlab[qr] - compute the QR orthogonal-triangular decomposition of
             a MapleMatrix or MatlabMatrix in Matlab, where A*P=Q*R

Calling Sequence in Maple

     qr(A, output='R')
     qr(A, output='QR')
     qr(A, output='QRP')

Parameters
     A       - MapleMatrix or MatlabMatrix
     output  - specify the form of the output (optional)
     R       - return the upper triangular matrix R
     QR      - return unitary matrix Q and upper triangular R matrix,
               so that A=QR
     QRP     - return Q, R, and permutation matrix P, so that AP=QR

Note that the R in output='R' is surrounded by quotation marks,
since the variable R was assigned previously.
```

How to obtain the matrices? There are some notations in Maple,
the left hand side and the parameter notation are a little bit other like in Matlab
(vgl. Abschnitt 2.8).

```
Q and R   (default)

  qr(A)                  # 2 results with 18 digits mantissa
  Matlab[qr](A)
  Matlab[qr](A,output='QR');
  (Q,R):=Matlab[qr](A);
  Q;
  R;

  QR:=Matlab[qr](A);
  Q:=QR[1];
  R:=QR[2];
  Q:=evalm(QR[1]);       # Q with 10 digits mantissa

R only

  qr(A,output='R')       # result with 18 digits mantissa
  Matlab[qr](A,output='R')
  Matlab[qr](A,output='R');

  T:=Matlab[qr](A,output='R');
  R:=matrix(n,n,(i,j)->if i<=j then T[i,j] else 0 end if);
                         # clear the lower triangular part of T
  evalm(R);              # 10 digits mantissa


Q, R and P

  qr(A,output='QRP')     # 3 results with 18 digits mantissa
  Matlab[qr](A,output='QRP');
  (Q,R,P):=Matlab[qr](A,output='QRP');
  Q;
  R;
  P;

  T:=Matlab[qr](A,output='QRP');
  Q:=T[1];
  R:=T[2];
  P:=T[3];
  Q:=evalm(T[1]);        #  Q with 10 digits mantissa
```

Beispiel 1: Anwendung auf Tridiagonalmatrix, $A = A^T > 0$

```
> n:=3:
  A:=band([-1,2,-1],n); # alternative Definitionen siehe Bsp. in 2.11.1
```

$$A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

(a) Berechnungen in Matlab mit Kommando `qr`
QR-Faktorisierung von $A, Q(m,m), R(m,n), A = QR$
Matlab im *double*-Format mit 15-stelliger Anzeige

```
qr(A) is the output of LAPACK'S DGEQRF or ZGEQRF routine. triu(qr(A)) is R.
       -2.23606797749979   1.78885438199983  -0.44721359549996
       -0.23606797749979  -1.67332005306815   1.91236577493503
                        0  -0.33167926656828   1.06904496764970

[Q,R] = qr(A) produces an upper triangular matrix R of the same dimension
             as A and a unitary matrix Q so that A = Q*R.
   Q =  -0.89442719099992  -0.35856858280032   0.26726124191242
         0.44721359549996  -0.71713716560064   0.53452248382485
                        0   0.59761430466720   0.80178372573727

   R =  -2.23606797749979   1.78885438199983  -0.44721359549996
                        0  -1.67332005306815   1.91236577493503
                        0                  0   1.06904496764970,
                                      abs(diag(R)) is already decreasing

 [Q,R,P] = qr(A) produces a permutation matrix P, an upper triangular R and
                a unitary Q so that A*P = Q*R. The column permutation P
                is chosen so that abs(diag(R)) is decreasing.
   Q =  -0.40824829046386  -0.43643578047198   0.80178372573727
         0.81649658092773   0.21821789023599   0.53452248382485
        -0.40824829046386   0.87287156094397   0.26726124191242

   R =   2.44948974278318  -1.63299316185545  -1.63299316185545
                        0   1.52752523165195  -1.09108945117996
                        0                  0   1.06904496764970,
                                      abs(diag(R)) is decreasing
   P =    0     0     1
          1     0     0
          0     1     0
```

Frage: Warum wird A spaltenvertauscht, obwohl $A = A^T > 0$ und `[Q,R]=qr(A)` eine obere Dreiecksmatrix R liefert, wo die Diagonalelemente schon betragsmaessig fallend geordnet sind?

(b) Nutzung der Maple Funktion zur QR-Faktorisierung

`QRdecomp`

```
> # For exact symbolic computation the low speed Gram-Schmidt process
  # is applied.
  # For matricies of floating point entries, the high speed numerically
  # stable Householder-transformations are used,
  # like variant with Matlab qr.

  # Gram-Schmidt: symbolisch
  Q:='Q':
  R:=QRdecomp(A,Q='q',rank='r');   # R[i,i]>=0
  r;
  Q:=evalm(q);
  evalf(evalm(R));
  evalf(evalm(Q));

  # genauere Rechnung
  Digits:=18:
  evalf(evalm(R));
  evalf(evalm(Q));
  Digits:=10:
```

$$R := \begin{bmatrix} \sqrt{5} & -\frac{4\sqrt{5}}{5} & \frac{\sqrt{5}}{5} \\ 0 & \frac{\sqrt{70}}{5} & -\frac{8\sqrt{70}}{35} \\ 0 & 0 & \frac{2\sqrt{14}}{7} \end{bmatrix}$$

$$3$$

$$Q := \begin{bmatrix} \frac{2\sqrt{5}}{5} & \frac{3\sqrt{70}}{70} & \frac{\sqrt{14}}{14} \\ -\frac{\sqrt{5}}{5} & \frac{3\sqrt{70}}{35} & \frac{\sqrt{14}}{7} \\ 0 & -\frac{\sqrt{70}}{14} & \frac{3\sqrt{14}}{14} \end{bmatrix}$$

$$\begin{bmatrix} 2.236067977 & -1.788854382 & 0.4472135954 \\ 0. & 1.673320053 & -1.912365775 \\ 0. & 0. & 1.069044968 \end{bmatrix}$$

$$\begin{bmatrix} 0.8944271908 & 0.3585685828 & 0.2672612419 \\ -0.4472135954 & 0.7171371655 & 0.5345224840 \\ 0. & -0.5976143047 & 0.8017837258 \end{bmatrix}$$

$$\begin{bmatrix} 2.23606797749978970 & -1.78885438199983176 & 0.447213595499957940 \\ 0. & 1.67332005306815110 & -1.91236577493502982 \\ 0. & 0. & 1.06904496764969754 \end{bmatrix}$$

$$\begin{bmatrix} 0.894427190999915880 & 0.358568582800318092 & 0.267261241912424385 \\ -0.447213595499957940 & 0.717137165600636184 & 0.534522483824848769 \\ 0. & -0.597614304667196820 & 0.801783725737273156 \end{bmatrix}$$

Kurze Fehlerbetrachtung bei GP-Rechnung
Auswirkung der Fehler bei Rechnung mit 10-stelliger Mantisse

```
> sqrt(5);
  evalf(%);   # 18-stellige Anzeige, ca. 15..16-stellige Genauigkeit
  evalf(sqrt(5.0),10),evalf(sqrt(5.0),11),
  evalf(sqrt(5.0),18),evalf(sqrt(5.0),20);
  sqrt(5)/5;
  evalf(%);   # 18-stellige Anzeige, ca. 15..16-stellige Genauigkeit
  evalf(sqrt(5.0)/5,10),evalf(sqrt(5.0)/5,11),
  evalf(sqrt(5.0)/5,18),evalf(sqrt(5.0)/5,20);
  2*sqrt(5)/5;
  evalf(%);   # 18-stellige Anzeige, ca. 15..16-stellige Genauigkeit
  evalf(2*sqrt(5.0)/5,10),evalf(2*sqrt(5.0)/5,11),
  evalf(2*sqrt(5.0)/5,18),evalf(2*sqrt(5.0)/5,20);
```

$$\sqrt{5}$$

$$2.23606797749978970$$

$$2.236067977,\ 2.2360679775,\ 2.23606797749978970,$$
$$2.2360679774997896964$$

$$\frac{\sqrt{5}}{5}$$

$$0.447213595499957940$$

$$0.4472135954,\ 0.44721359550,\ 0.447213595499957940,$$
$$0.44721359549995793928$$

$$\frac{2\sqrt{5}}{5}$$

$$0.894427190999915880$$

$$0.8944271908,\ 0.89442719100,\ 0.894427190999915880,$$
$$0.89442719099991587856$$

```
> # Householder: numerisch
  Digits:=10:
  An:=evalm(A): An[1,1]:=1.0*An[1,1]:
  evalm(An);
  Rn:=QRdecomp(An,Q='q',rank='r');   # Rn[i,i]>=0
  # Warum wird Rn mit 18-stelliger Mantisse angezeigt, obwohl ungefaehr
  # die letzten 2 Stellen ungenau sind?
  r;
  Qn:=evalm(q);  # 10 Dezimalstellen angezeigt
```

$$\begin{bmatrix} 2.0 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

$$Rn := \begin{bmatrix} 2.23606797749978981 & -1.78885438199983194 & 0.447213595499957928 \\ 0. & 1.67332005306815090 & -1.91236577493503024 \\ 0. & 0. & 1.06904496764969714 \end{bmatrix}$$

$$
\begin{array}{c}
3 \\
Qn := \begin{bmatrix} 0.8944271910 & 0.3585685828 & 0.2672612419 \\ -0.4472135955 & 0.7171371656 & 0.5345224838 \\ 0. & -0.5976143047 & 0.8017837257 \end{bmatrix}
\end{array}
$$

(c) Maple → Matlab verwendet die Variante `qr(A)`

```
> qr(A);     # Matlab[qr](A);
```

$$
\begin{bmatrix} -0.894427190999915744 & -0.358568582800318114 & 0.267261241912424396 \\ 0.447213595499957928 & -0.717137165600636340 & 0.534522483824848794 \\ 0. & 0.597614304667196894 & 0.801783725737273079 \end{bmatrix}
$$

$$
\begin{bmatrix} -2.23606797749978981 & 1.78885438199983150 & -0.447213595499957928 \\ 0. & -1.67332005306815090 & 1.91236577493502979 \\ 0. & 0. & 1.06904496764969758 \end{bmatrix}
$$

```
> QR:=Matlab[qr](A):  # oder QR:=qr(A): QR:=Matlab[qr](A,output='QR'):
  Q:=QR[1];
  R:=QR[2];
  # warum 18-stellige Mantisse?
```

$$
Q := \begin{bmatrix} -0.894427190999915744 & -0.358568582800318114 & 0.267261241912424396 \\ 0.447213595499957928 & -0.717137165600636340 & 0.534522483824848794 \\ 0. & 0.597614304667196894 & 0.801783725737273079 \end{bmatrix}
$$

$$
R := \begin{bmatrix} -2.23606797749978981 & 1.78885438199983150 & -0.447213595499957928 \\ 0. & -1.67332005306815090 & 1.91236577493502979 \\ 0. & 0. & 1.06904496764969758 \end{bmatrix}
$$

```
> (Q,R,P):=Matlab[qr](A,output='QRP');
```

$$
Q, R, P :=
$$

$$
\begin{bmatrix} -0.894427190999915744 & -0.358568582800318114 & 0.267261241912424396 \\ 0.447213595499957928 & -0.717137165600636340 & 0.534522483824848794 \\ 0. & 0.597614304667196894 & 0.801783725737273079 \end{bmatrix},
$$

$$
\begin{bmatrix} -2.23606797749978981 & 1.78885438199983150 & -0.447213595499957928 \\ 0. & -1.67332005306815090 & 1.91236577493502979 \\ 0. & 0. & 1.06904496764969758 \end{bmatrix},
$$

$$
\begin{bmatrix} 0. & 0. & 1. \\ 1. & 0. & 0. \\ 0. & 1. & 0. \end{bmatrix}
$$

Achtung
Es ist schon eigenartig, dass bei einer Matrix $A = A^T > 0$ und A irreduzibel diagonaldominant bei der Matlab QR-Faktorisierung eine Vertauschung der linear unabhaengigen Spalten vorgenommen wird.

Beispiel 2: Anwendung auf Tridiagonalmatrix, $A > 0$

```
> n:=3:
  A:=band([1,2,-1],n); # alternative Definitionen siehe Bsp. in (2.11.1)
```

$$A := \begin{bmatrix} 2 & -1 & 0 \\ 1 & 2 & -1 \\ 0 & 1 & 2 \end{bmatrix}$$

(a) Berechnungen in Matlab mit Kommando `qr`
QR-Faktorisierung von $A, Q(m,m), R(m,n), A = QR$
Matlab im *double*-Format mit 15-stelliger Anzeige

```
[Q,R] = qr(A) produces an upper triangular matrix R of the same dimension
              as A and a unitary matrix Q so that A = Q*R.

   Q =  -0.89442719099992   0.40824829046386   0.18257418583506
        -0.44721359549996  -0.81649658092773  -0.36514837167011
                         0  -0.40824829046386   0.91287092917528

   R =  -2.23606797749979  -0.00000000000000   0.44721359549996
                         0  -2.44948974278318                  0
                         0                   0   2.19089023002066
                                          abs(diag(R)) is not decreasing

[Q,R,P] = qr(A) produces a permutation matrix P, an upper triangular R
                and a unitary Q so that A*P = Q*R.
                The column permutation P is chosen so that abs(diag(R))
                is decreasing.

   Q =  -0.40824829046386  -0.89442719099992   0.18257418583506
         0.81649658092773  -0.44721359549996  -0.36514837167011
         0.40824829046386   0.00000000000000   0.91287092917528

   R =   2.44948974278318                   0                   0
                         0  -2.23606797749979   0.44721359549996
                         0                   0   2.19089023002066
                                          abs(diag(R)) is decreasing

   P =    0     1     0
          1     0     0
          0     0     1
```

(b) Nutzung der Maple Funktion zur QR-Faktorisierung
`QRdecomp`

```
> # Gram-Schmidt: symbolisch
  Q:='Q':
  R:=QRdecomp(A,Q='q',rank='r'):   # R[i,i]>=0
  Q:=evalm(q);
  evalm(R);
  evalf(evalm(Q));
  evalf(evalm(R));
```

$$Q := \begin{bmatrix} \frac{2\sqrt{5}}{5} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{30}}{30} \\ \frac{\sqrt{5}}{5} & \frac{\sqrt{6}}{3} & -\frac{\sqrt{30}}{15} \\ 0 & \frac{\sqrt{6}}{6} & \frac{\sqrt{30}}{6} \end{bmatrix}$$

$$R := \begin{bmatrix} \sqrt{5} & 0 & -\frac{\sqrt{5}}{5} \\ 0 & \sqrt{6} & 0 \\ 0 & 0 & \frac{2\sqrt{30}}{5} \end{bmatrix}$$

$$\begin{bmatrix} 0.8944271908 & -0.4082482906 & 0.1825741858 \\ 0.4472135954 & 0.8164965809 & -0.3651483717 \\ 0. & 0.4082482906 & 0.9128709293 \end{bmatrix}$$

$$\begin{bmatrix} 2.236067977 & 0. & -0.4472135954 \\ 0. & 2.449489743 & 0. \\ 0. & 0. & 2.190890230 \end{bmatrix}$$

(c) Maple → Matlab verwendet die Variante `qr(A)`

```
> QR:=Matlab[qr](A):
  Q:=QR[1];
  R:=QR[2];
  # warum 18-stellige Mantisse?
```

$$Q := \begin{bmatrix} -0.894427190999915744 & 0.408248290463863018 & 0.182574185835055386 \\ -0.447213595499957928 & -0.816496580927726146 & -0.365148371670110772 \\ 0. & -0.408248290463863072 & 0.912870929175276791 \end{bmatrix}$$

$$R := \begin{bmatrix} -2.23606797749978981 & -0.111022302462515654\,10^{-15} & 0.447213595499957928 \\ 0. & -2.44948974278317788 & 0. \\ 0. & 0. & 2.19089023002066428 \end{bmatrix}$$

Beispiel 3: Verwendung der QR-Faktorisierung fuer Ausgleichsrechnung (quadratischer Ausgleich) $Ax \approx y$,
Methode der kleinsten Quadrate, Loesung des Normalgleichungssystems $A^T Ax = A^T y$

Maple: `QRdecomp`

Daten

```
> m:=5;      # m<=n
> n:=3;
```

$$m := 5$$
$$n := 3$$

```
> tdatag:=vector(m,[1,2,3,4,5]);
  ydatag:=vector(m,[3,5,6,10,20]);
  pot:=(i,j) -> tdatag[i]^(j-1):
  Ag:=matrix(m,n,pot);
```

$$tdatag := [1, 2, 3, 4, 5]$$
$$ydatag := [3, 5, 6, 10, 20]$$
$$Ag := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \end{bmatrix}$$

(a) `QRdecomp`, Gram-Schmidt

```
> # For exact symbolic computation the low speed Gram-Schmidt process
  # is applied. For matricies of floating point entries, the high speed
  # numerically stable Householder-transformations are used,
  # like variant with Matlab qr

  # Gram-Schmidt: symbolisch
  Q:='Q':
  R:=QRdecomp(Ag,Q='q',rank='r');   # R[i,i]>=0
  r;
  Q:=evalm(q);
  evalf(evalm(R));
  evalf(evalm(Q));

  # genauere Rechnung
  Digits:=18:
  evalf(evalm(R));
  evalf(evalm(Q));
  Digits:=10:
```

$$R := \begin{bmatrix} \sqrt{5} & 3\sqrt{5} & 11\sqrt{5} \\ 0 & \sqrt{10} & 6\sqrt{10} \\ 0 & 0 & \sqrt{14} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$Q := \begin{bmatrix} \frac{\sqrt{5}}{5} & -\frac{\sqrt{10}}{5} & \frac{\sqrt{14}}{7} & \frac{2\sqrt{35}}{35} & 0 \\ \frac{\sqrt{5}}{5} & -\frac{\sqrt{10}}{10} & -\frac{\sqrt{14}}{14} & -\frac{9\sqrt{35}}{70} & \frac{\sqrt{5}}{10} \\ \frac{\sqrt{5}}{5} & 0 & -\frac{\sqrt{14}}{7} & \frac{3\sqrt{35}}{70} & -\frac{3\sqrt{5}}{10} \\ \frac{\sqrt{5}}{5} & \frac{\sqrt{10}}{10} & -\frac{\sqrt{14}}{14} & \frac{\sqrt{35}}{14} & \frac{3\sqrt{5}}{10} \\ \frac{\sqrt{5}}{5} & \frac{\sqrt{10}}{5} & \frac{\sqrt{14}}{7} & -\frac{3\sqrt{35}}{70} & -\frac{\sqrt{5}}{10} \end{bmatrix}$$

$$\begin{bmatrix} 2.236067977 & 6.708203931 & 24.59674775 \\ 0. & 3.162277660 & 18.97366596 \\ 0. & 0. & 3.741657387 \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

$$\begin{bmatrix} 0.4472135954 & -0.6324555320 & 0.5345224840 & 0.3380617019 & 0. \\ 0.4472135954 & -0.3162277660 & -0.2672612419 & -0.7606388294 & 0.2236067977 \\ 0.4472135954 & 0. & -0.5345224840 & 0.2535462764 & -0.6708203931 \\ 0.4472135954 & 0.3162277660 & -0.2672612419 & 0.4225771274 & 0.6708203931 \\ 0.4472135954 & 0.6324555320 & 0.5345224840 & -0.2535462764 & -0.2236067977 \end{bmatrix}$$

$$\begin{bmatrix} 2.23606797749978970 & 6.70820393249936910 & 24.5967477524976867 \\ 0. & 3.16227766016837933 & 18.9736659610102760 \\ 0. & 0. & 3.74165738677394139 \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

$$\begin{bmatrix} 0.447213595499957940 & -0.632455532033675866 & 0.534522483824848769 & 0.338061701891406631 & 0. \\ 0.447213595499957940 & -0.316227766016837933 & -0.267261241912424385 & -0.760638829255664917 & 0.223606797749978970 \\ 0.447213595499957940 & 0. & -0.534522483824848769 & 0.253546276418554973 & -0.670820393249936910 \\ 0.447213595499957940 & 0.316227766016837933 & -0.267261241912424385 & 0.422577127364258289 & 0.670820393249936910 \\ 0.447213595499957940 & 0.632455532033675866 & 0.534522483824848769 & -0.253546276418554973 & -0.223606797749978970 \end{bmatrix}$$

(b) `QRdecomp`, Householder

```
> # Householder: numerisch
  Digits:=10:
  Agn:=evalm(Ag): Agn[1,1]:=1.0*Agn[1,1]:
  evalm(Agn);
  Rn:=QRdecomp(Agn,Q='q',rank='r');   # Rn[i,i]>=0
  # Warum wird Rn mit 18-stelliger Mantisse angezeigt, obwohl ungefaehr
  # die letzten drei Stellen ungenau sind?
  r;
  Qn:=evalm(q);
```

$$\begin{bmatrix} 1.0 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \end{bmatrix}$$

$$Rn := \begin{bmatrix} 2.23606797749978981 & 6.70820393249937030 & 24.5967477524976880 \\ 0. & 3.16227766016837952 & 18.9736659610102762 \\ 0. & 0. & 3.74165738677393956 \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

$$3$$

$$Qn :=$$

$$\begin{bmatrix} 0.4472135955 & -0.6324555320 & 0.5345224838 & -0.02576090840 & -0.3370787592 \\ 0.4472135955 & -0.3162277660 & -0.2672612419 & 0.2809186861 & 0.7413879688 \\ 0.4472135955 & 0.2775557562\,10^{-16} & -0.5345224838 & -0.6881906081 & -0.2016913515 \\ 0.4472135955 & 0.3162277660 & -0.2672612419 & 0.6366687913 & -0.4724661668 \\ 0.4472135955 & 0.6324555320 & 0.5345224838 & -0.2036359610 & 0.2698483087 \end{bmatrix}$$

(c) Loesung des Ausgleichsproblems mit Ergebnissen von Gram-Schmidt

```
> q:=coldim(Q);   #  q:=rowdim(Q);
  r:=rank(R);

  Q1:=submatrix(Q,1..q,1..r);
  R1:=submatrix(R,1..r,1..r);
  c:=evalm(linalg[transpose](Q1)&*ydatag);   # Transposition symb.
  xs:=linsolve(R1,c);
  evalf(xs);
  Digits:=16: evalf(xs); Digits:=10:

  Res:=norm(evalm(Ag&*xs-ydatag));
  evalf(Res);
  Digits:=16: evalf(Res); Digits:=10:
```

$$q := 5$$

$$r := 3$$

$$Q1 := \begin{bmatrix} \frac{\sqrt{5}}{5} & -\frac{\sqrt{10}}{5} & \frac{\sqrt{14}}{7} \\ \frac{\sqrt{5}}{5} & -\frac{\sqrt{10}}{10} & -\frac{\sqrt{14}}{14} \\ \frac{\sqrt{5}}{5} & 0 & -\frac{\sqrt{14}}{7} \\ \frac{\sqrt{5}}{5} & \frac{\sqrt{10}}{10} & -\frac{\sqrt{14}}{14} \\ \frac{\sqrt{5}}{5} & \frac{\sqrt{10}}{5} & \frac{\sqrt{14}}{7} \end{bmatrix}$$

$$R1 := \begin{bmatrix} \sqrt{5} & 3\sqrt{5} & 11\sqrt{5} \\ 0 & \sqrt{10} & 6\sqrt{10} \\ 0 & 0 & \sqrt{14} \end{bmatrix}$$

$$c := \left[\frac{44\sqrt{5}}{5}, \frac{39\sqrt{10}}{10}, \frac{19\sqrt{14}}{14}\right]$$

$$xs := \left[\frac{33}{5}, \frac{-297}{70}, \frac{19}{14}\right]$$

[6.600000000, -4.242857143, 1.357142857]

$$Res := \frac{51}{35}$$

1.457142857

(d) Loesung des Ausgleichsproblems mit Ergebnissen von Householder

```
> q:=coldim(Qn);   #  q:=rowdim(Qn);
  r:=rank(Rn);

  Qn1:=submatrix(Qn,1..q,1..r);
  Rn1:=submatrix(Rn,1..r,1..r);
  cn:=evalm(transpose(Qn1)&*ydatag);    # Transposition mit Matlab
  xns:=linsolve(Rn1,cn);
  Res:=norm(evalm(Agn&*xns-ydatag));
```

$$q := 5$$

$$r := 3$$

$$Qn1 := \begin{bmatrix} 0.4472135955 & -0.6324555320 & 0.5345224838 \\ 0.4472135955 & -0.3162277660 & -0.2672612419 \\ 0.4472135955 & 0.2775557562\,10^{-16} & -0.5345224838 \\ 0.4472135955 & 0.3162277660 & -0.2672612419 \\ 0.4472135955 & 0.6324555320 & 0.5345224838 \end{bmatrix}$$

$$Rn1 := \begin{bmatrix} 2.23606797749978981 & 6.70820393249937030 & 24.5967477524976880 \\ 0. & 3.16227766016837952 & 18.9736659610102762 \\ 0. & 0. & 3.74165738677393956 \end{bmatrix}$$

$$cn := [19.67739820, 12.33288287, 5.077963599]$$

$$xns := [6.600000014, -4.242857150, 1.357142858]$$

$$Res := 1.457142854$$

Matlab: `qr`

Daten

```
> tdata:=vector(m,[1.,2.,3.,4.,5.]);
  ydata:=vector(m,[3.,5.,6.,10.,20.]);
  pot:=(i,j) -> tdata[i]^(j-1):
  A:=matrix(m,n,pot):
```

$$tdata := [1., 2., 3., 4., 5.]$$

$$ydata := [3., 5., 6., 10., 20.]$$

(a) Matlab-Versionen, Rechnungen mit akzeptabler Genauigkeit

```
 x=A\y
 norm(A*x-y,inf)
 x=(A'*A)\(A'*y)
 norm(A*x-y,inf)

 [Q,R] = qr(A)
 Q1=Q(1:5,1:3);
 R1=R(1:3,1:3);

 x=(R'*R)\(R'*Q'*y)
 norm(A*x-y,inf)

 x=(R1)\(Q1'*y)
 norm(A*x-y,inf)
-------------------------------------
V1:     x =
           6.59999999999999
          -4.24285714285714
           1.35714285714286
        ans =
           1.45714285714286

V2=V1: x =
           6.59999999999999
          -4.24285714285714
           1.35714285714286
        ans =
           1.45714285714286

V3,4: Q =
        Columns 1 through 5
       -0.44721359549996  -0.63245553203368   0.53452248382485  -0.02576090839873  -0.33707875917089
       -0.44721359549996  -0.31622776601684  -0.26726124191243   0.28091868614925   0.74138796884196
       -0.44721359549996   0.00000000000000  -0.53452248382485  -0.68819060805535  -0.20169135150055
       -0.44721359549996   0.31622776601684  -0.26726124191242   0.63666879125789  -0.47246616684123
       -0.44721359549996   0.63245553203368   0.53452248382485  -0.20363596095305   0.26984830867071

        R =
          -2.23606797749979  -6.70820393249937 -24.59674775249768
                          0   3.16227766016838  18.97366596101028
                          0                   0   3.74165738677394
                          0                   0                  0
                          0                   0                  0
```

```
V3:    x =
          6.59999999999981
         -4.24285714285699
          1.35714285714283
       ans =
          1.45714285714284

V4:    x =
          6.60000000000000
         -4.24285714285714
          1.35714285714286
       ans =
          1.45714285714285
```

(b) $Ax \approx y \;\rightarrow$ Normalgleichungssystem, Gausssche Normalgleichungen
$A^TAx = A^Ty$, $\text{rang}(A) = n$,
$xs = 6.6,\ -4.2428571428571428571,\ 1.3571428571428571429$
Transpositionen mit Matlab Funktion

```
> # Rechnungen relativ ungenau
  x:=linsolve(transpose(A)&*A,transpose(A)&*ydata);
  Res:=norm(evalm(A&*x-ydata));
```

$$x := [6.600000267,\ -4.242857365,\ 1.357142894]$$
$$Res := 1.457142887$$

```
Digits:=16:
> x:=linsolve(transpose(A)&*A,transpose(A)&*ydata);
> Res:=norm(evalm(A&*x-ydata));
> Digits:=10:
```

$$x := [6.600000000000026,\ -4.242857142857173,\ 1.357142857142863]$$
$$Res := 1.457142857142868$$

(c) QR-Faktorisierung von A,
$Q(m,m),\ R(m,n),\ Ax = QRx\ y \rightarrow R^TQ^TQRx = R^TQ^Ty \rightarrow R^TRx = R^TQ^Ty$,
$xs = 6.6,\ -4.2428571428571428571,\ 1.3571428571428571429$

```
> QR:=Matlab[qr](A):   # oder QR:=qr(A):  QR:=Matlab[qr](A,output='QR'):
                       # Ergebnisvektor mit 2 Komponenten = Matrizen Q,R
                       # kein Permutationsvektor
  Q:=QR[1];
  R:=QR[2];
  x:=linsolve(transpose(R)&*R,transpose(R)&*transpose(Q)&*ydata);
  Res:=norm(evalm(A&*x-ydata));
```

$$Q :=$$

$$\begin{bmatrix}
-0.447213595499957872 & -0.632455532033675993 & 0.534522483824849348 & -0.025760908398730533 & -0.337078759170888032 \\
-0.447213595499957872 & -0.316227766016837886 & -0.267261241912426284 & 0.280918686149245755 & 0.741387968841961786 \\
-0.447213595499957872 & 0.277555756156289136\,10^{-16} & -0.534522483824847906 & -0.688190608055352926 & -0.201691351500550731 \\
-0.447213595499957872 & 0.316227766016837996 & -0.267261241912423508 & 0.636668791257890998 & -0.472466166841230051 \\
-0.447213595499957872 & 0.632455532033675993 & 0.534522483824848350 & -0.203635960953053419 & 0.269848308670707192
\end{bmatrix}$$

$$R := \begin{bmatrix}
-2.23606797749978981 & -6.70820393249936941 & -24.5967477524976844 \\
0. & 3.16227766016837864 & 18.9736659610102834 \\
0. & 0. & 3.74165738677394400 \\
0. & 0. & 0. \\
0. & 0. & 0.
\end{bmatrix}$$

$$x := [6.600000461, -4.242857518, 1.357142918]$$

$$Res := 1.457142903$$

Man sollte die Genauigkeit von Matlab (Format *double*, 15..16 Dezimalstellen) bei Anzeige einer 18-stelligen Mantisse (die letzten 2..3 Stellen sind damit, wie schon erwaehnt, ungenau) durch die Maple-Matlab Verbindung dann in Maple weiter nutzen. Es macht meist wenig Sinn, mit Maple Kommandos wie hier &∗, `linsolve` oder `norm` fortzusetzen, die wieder in der einfachen Genauigkeit `Digits=10` arbeiten.
Genauso sollten schon die Daten mit entsprechender Genauigkeit bereitgestellt werden (im Beispiel hier enthaelt die Referenz fuer den Datenausgleich exakte Werte). Deshalb

```
> Digits:=16:   # kein Einfluss auf  QR:=Matlab[qr](A):
  x:=linsolve(transpose(R)&*R,transpose(R)&*transpose(Q)&*ydata);
  Res:=norm(evalm(A&*x-ydata));
  Digits:=10:
```

$$x := [6.600000000000297, -4.242857142857397, 1.357142857142900]$$

$$Res := 1.457142857142897$$

(d) $R(m,n) \to R1(n,n),\ \mathrm{rang}(R1) = n,\ Q(m,m) \to Q1(m,n)$,
$A = Q1 * R1,\ R1\,x = Q1^T y$,
$xs = 6.6,\ -4.2428571428571428571,\ 1.3571428571428571429$
wie bei `QRdecomp` mit Householder-Transformation (etwas genauer)

```
> Digits:=10:
  q:=coldim(Q);   #  q:=rowdim(Q);
  r:=rank(R);
  Q1:=submatrix(Q,1..q,1..r):   # Genauigkeit auf 10 Digits
  R1:=submatrix(R,1..r,1..r):
  c:=evalm(linalg[transpose](Q1)&*ydata); # c:=evalm(transpose(Q1)&*ydata);
  x:=linsolve(R1,c);
  Res:=norm(evalm(A&*x -ydata));
```

$$q := 5$$
$$r := 3$$
$$c := [\text{-19.67739820, 12.33288287, 5.077963599}]$$
$$x := [\text{6.600000014, -4.242857150, 1.357142858}]$$
$$Res := 1.457142854$$

```
> Digits:=16:
  Q1:=submatrix(Q,1..q,1..r);
  R1:=submatrix(R,1..r,1..r);
  c:=evalm(linalg[transpose](Q1)&*ydata);
  x:=linsolve(R1,c);
  Res:=norm(evalm(A&*x -ydata));
  Digits:=10:
```

$$Q1 := \begin{bmatrix} \text{-0.4472135954999579} & \text{-0.6324555320336760} & 0.5345224838248493 \\ \text{-0.4472135954999579} & \text{-0.3162277660168379} & \text{-0.2672612419124263} \\ \text{-0.4472135954999579} & 0.2775557561562891\,10^{-16} & \text{-0.5345224838248479} \\ \text{-0.4472135954999579} & 0.3162277660168380 & \text{-0.2672612419124235} \\ \text{-0.4472135954999579} & 0.6324555320336760 & 0.5345224838248484 \end{bmatrix}$$

$$R1 := \begin{bmatrix} \text{-2.236067977499790} & \text{-6.708203932499369} & \text{-24.59674775249768} \\ 0. & 3.162277660168379 & 18.97366596101028 \\ 0. & 0. & 3.741657386773944 \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

$$c := [\text{-19.67739820199815, 12.33288287465668, 5.077963596336064}]$$
$$x := [\text{6.599999999999999, -4.242857142857137, 1.357142857142856}]$$
$$Res := 1.457142857142851$$

Wenn man also die QR-Faktorisierung von A hat, so ist zur Loesung des Ausgleichsproblems die "kurze" Formel $R1\,x = Q1^T y$ zu verwenden.

2.11.3 Nutzerdefinierte Matlab Funktionen unter Maple

Die Funktionen müssen in einem Verzeichnis des "MATLABPATH" enthalten sein, deshalb:

MATLAB Command Window → File → SetPath... → **Add Folder ...** →
Add Folder to Path → Eintrag des Pfads und **OK** → zurueck und **Save**

Man beachte, dass grafische Resultate in einem Grafikfenster von Matlab angezeigt werden. Damit wird die letzte Grafik angezeigt. Will man mehrere Figuren sehen, dann sollte man diese in eine Tabelle (Matlab `subplot`) aufnehmen.
Die Grafik im aktuellen Figure-Fenster kann mittels

File → Export... → (Speichern in, Dateiname, Dateityp) → **Speichern**

abgelegt werden.
Falls mehrere Teilergebnisse vorliegen, dann diese als ein Ergebnis in Vektor/Liste zusammenfassen. Einige Matlab-Befehle in den m-Files werden ignoriert, z. B.

`pause, pause(n), echo, diary, disp, print *.ps -dps, ...`

Beispiel 1: Matrixmanipulationen

Generierung einer Bandmatrix aus A mit $u \geq 0$ oberen und $l \geq 0$ unteren Nebendiagonalen

Maple: Arbeit mit Bandmatrizen

```
> n:=4:
  A:=matrix(n,n,(i,j)->i+j^2);
  l:=2; u:=1;
  T:=matrix(n,n,(i,j)->if (i-l<=j) and (j<=i+u) then A[i,j] else 0 end if);
```

$$A := \begin{bmatrix} 2 & 5 & 10 & 17 \\ 3 & 6 & 11 & 18 \\ 4 & 7 & 12 & 19 \\ 5 & 8 & 13 & 20 \end{bmatrix}$$

$$l := 2$$

$$u := 1$$

$$T := \begin{bmatrix} 2 & 5 & 0 & 0 \\ 3 & 6 & 11 & 0 \\ 4 & 7 & 12 & 19 \\ 0 & 8 & 13 & 20 \end{bmatrix}$$

Bandmatrizen, i. Allg. in Rechteckform

```
> T:=Matrix(n,n,shape=band[1,1]):
  T[1,1]:=2: T[1,2]:=-1: T[2,1]:=-1: T:=evalm(T);
  T1:=Matrix([[-1,-1,-1],[2,2,2],[1,1,1]],shape=band[1,1]);
  T2:=Matrix([[-1,-1,-1],[2,2,2,2],[1,1,1]],shape=band[1,1],scan=band[1,1]);
  T3:=Matrix([[1,2,3],[1,1,1,1],[2,3,4,5],[6,7,8]],shape=band[1,2],
              scan=band[1,2]);
```

$$T := \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T1 := \begin{bmatrix} -1 & -1 & 0 \\ 2 & 2 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

$$T2 := \begin{bmatrix} 2 & 1 & 0 & 0 \\ -1 & 2 & 1 & 0 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

$$T3 := \begin{bmatrix} 1 & 2 & 6 & 0 & 0 \\ 1 & 1 & 3 & 7 & 0 \\ 0 & 2 & 1 & 4 & 8 \\ 0 & 0 & 3 & 1 & 5 \end{bmatrix}$$

```
> with(LinearAlgebra):
  LL:=[[1,1],[2,2,2],[3,3,3],[-1,-1]]:
  BandMatrix(LL,2);            # BandMatrix(LL);
  BandMatrix(LL,0);            # 0 untere Nebendiagonale
  BandMatrix(LL,1);            # 1 untere Nebendiagonale
  BandMatrix(LL,1,5);          # 1 untere Nebendiagonale, Zeilenanzahl=5
  BandMatrix([-1,4,-1],1,5);   # 1 untere Nebendiagonale, Dimension=5
```

$$\begin{bmatrix} 3 & -1 & 0 \\ 2 & 3 & -1 \\ 1 & 2 & 3 \\ 0 & 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & -1 & 0 \\ 0 & 1 & 2 & 3 & -1 \\ 0 & 0 & 0 & 2 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & -1 & 0 \\ 1 & 2 & 3 & -1 \\ 0 & 0 & 2 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & -1 & 0 \\ 1 & 2 & 3 & -1 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & -1 & 4 \end{bmatrix}$$

Diagonalen von A zu einer Liste von Listen machen

```
> evalm(A);
  l,u;
  LLA:=[seq([seq(A[i,i+k-l],i=max(l+1-k,1)..min(n,n+l-k))],k=0..l+u)];
  T:=BandMatrix(LLA,l);
```

$$\begin{bmatrix} 2 & 5 & 10 & 17 \\ 3 & 6 & 11 & 18 \\ 4 & 7 & 12 & 19 \\ 5 & 8 & 13 & 20 \end{bmatrix}$$

$$2{,}1$$

$$LLA := [[4, 8], [3, 7, 13], [2, 6, 12, 20], [5, 11, 19]]$$

$$T := \begin{bmatrix} 2 & 5 & 0 & 0 \\ 3 & 6 & 11 & 0 \\ 4 & 7 & 12 & 19 \\ 0 & 8 & 13 & 20 \end{bmatrix}$$

Matlab: `triu(tril(A,u),-l);`

```
A = [ 2, 5, 10, 17,
      3, 6, 11, 18,
      4, 7, 12, 19,
      5, 8, 13, 20 ]

triu(tril(A,1),-2)

ans =
     2     5     0     0
     3     6    11     0
     4     7    12    19
     0     8    13    20
```

Maple → Matlab mit nutzerdefiniertem Funktions-m-File
Anwendung der eleganten Kommandos `triu` und `tril`

```
% band_ul.m
% Bandmatrix mit u oberen und l unteren Nebendiagonalen
function T = band_ul(A,u,l)
  T=A;
  if (u>=0) & (l>=0)
    T = triu(tril(T,u),-l);
end;
% Ende band_ul
```

```
> # Test, ob Funktion existiert
  Matlab[defined]("band_ul",'function');
```

$$true$$

```
> # initialize variable or matrix in Maple and copy it to the Matlab
  # environment
  # Maple -> Matlab-Variable
  Matlab[setvar]("As",A);
  alpha:=2:
  beta:=1:
  Matlab[setvar]("l",alpha);
  Matlab[setvar]("u",beta);

> # evaluate a Matlab expression (is a string) using Matlab
  # Ausfuehrung der Matlab-Funktion
  Matlab[evalM]("T=band_ul(As,u,l)"):

> # get a numerical array or matrix in an open Matlab session
  # Rueckgabe der Bandmatrix
  T:=Matlab[getvar]("T");
```

$$T := \begin{bmatrix} 2. & 5. & 0. & 0. \\ 3. & 6. & 11. & 0. \\ 4. & 7. & 12. & 19. \\ 0. & 8. & 13. & 20. \end{bmatrix}$$

Beispiel 2: Nullstellenbestimmung

Zunächst die Darstellung einiger Grundlagen.

Die nichtlineare Gleichung (Normalform, Nullstellenform) $f(x) = 0$, x^* Lösung (Nullstelle von f), notiert man auch in der iterierfähigen Form (Fixpunktform) $x = g(x)$, x^* Lösung (Fixpunkt von g).

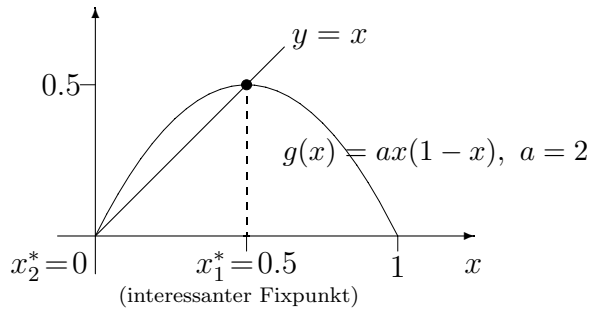


Abb. 2.1 Illustration der Fixpunktsituation

Ein Lösungsverfahren für die Fixpunktgleichung ist das allgemeine Iterationsverfahren (AIV, PICARD-Verfahren, Fixpunktiteration)

$$x_n = g(x_{n-1}), \quad n = 1, 2, 3, ..., \quad x_0 - \text{ Startnäherung.}$$

Der Banachsche Fixpunktsatz liefert Aussagen über Konvergenz, Grenzwert und Fehlerschätzungen.
Für den n-ten Iterationswert x_n gilt mit $|g'(x)| \leq \lambda < 1$ die

- a-priori–Fehlerschätzung

$$|x_n - x^*| \leq \frac{\lambda^n}{1-\lambda}|x_1 - x_0|,$$

- a-posteriori–Fehlerschätzung

$$|x_n - x^*| \leq \frac{\lambda}{1-\lambda}|x_n - x_{n-1}|.$$

Die erstere kann für eine grobe Abschätzung der Anzahl der notwendigen Iterationen bei gegebener Toleranz ε gemäß $\frac{\lambda^n}{1-\lambda}|x_1 - x_0| \approx \varepsilon$ verwendet werden. Die Umstellung nach n ergibt

$$n \approx \frac{\ln\left(\frac{\varepsilon(1-\lambda)}{|x_1 - x_0|}\right)}{\ln(\lambda)}, \quad \lambda \approx |g'(x^*)| < 1.$$

Des Weiteren kann man die Konvergenzordnung des AIV untersuchen.

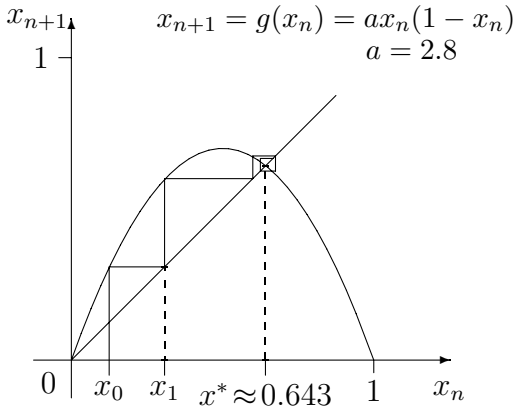


Abb. 2.2 Illustration des Iterationsverfahren

Die Implementierung in MATLAB basiert meist auf der Struktur

$$\textbf{Rahmenprogramm} \rightarrow \quad \{\textbf{AIV, evtl. Tabellierung}\} \quad \rightarrow \quad \{g,\ g'\},$$

wobei die einzelnen Komponenten als MATLAB-Funktionen in Form von entsprechenden Skript- oder Funktions-m-Files bereitgestellt werden.

Bei der Lösung einer Nullstellenaufgabe haben wir folgende übliche Vorgehensweise.

1. Man ermittelt grafisch die Anzahl und Lage der reellen Lösungen von $f(x) = 0$ bzw. $x = g(x)$.

2. Man wählt den Bereich (Intervall) um eine zu suchende Lösung und überprüft die Voraussetzungen des Fixpunktsatzes.

3. Bei zu erwartender Konvergenz bestimmt man die ausgewählte Lösung x^* der Gleichung $x = g(x)$ mit dem AIV $x_{n+1} = g(x_n),\ n = 0, 1, ...(\leq n_{max})$ bis auf eine Genauigkeit (Toleranz) von $\varepsilon > 0$. Der Startwert sei x_0.
   Man iteriert z. B. solange, bis die Lösung x^* auf t Nachkommastellen genau bestimmt wurde.

4. Der Iterationsverlauf wird grafisch dargestellt.

5. Man gibt die a-priori- und a-posteriori-Fehlerschätzung für den Fehler der Näherung x_n an.
   Man verwendet diese zur Abschätzung der Anzahl der notwendigen Iterationen bei gegebener Toleranz ε.

6. Man bemüht sich um die exakte Lösung x^* (evtl. AIV mit maximaler Genauigkeit rechnen).
   Man notiert für die Werte x_n den wahren Fehler $|x_n - x^*|$ und erläutert anhand der ermittelten wahren Fehler die lineare Konvergenz des AIV.
   Damit kann man auch die Fehlerabschätzungen überprüfen.

Nur einige dieser Ziele werden in den CAS-Arbeitsblättern umgesetzt.

Allgemeines Iterationsverfahren fuer Loesung der NLG $x = g(x)$

Matlab: m-Files fuer Funktionen

```
% aiv2.m
function  erg = aiv2(x0,a,b,g,gs,tol,nmax,tab,gra)
% Allgemeines Iterationsverfahren / Picard-Iteration fuer NLG x=g(x)
% mit Ausgaben, 1 Ergebnisgroesse

% Eingangsparameter
%  x0       Startwert
%  a,b      Intervallgrenzen a<b, x0 im betrachteten Intervall
%  g        g(x), Funktion als m-File
%  gs       g'(x),  Ableitung als m-File
%  tol      Toleranz
%  nmax     maximale Iterationsanzahl
%  tab      1=Tabellenausgabe, sonst nicht
%  gra      1=Grafik, sonst nicht
```

```
% Ergebnisse  erg=[xs,k]
%  xs       Loesung  oder letzte Iterierte
%  k        Anzahl der benoetigten Iterationen
%  Diverse Informationen ueber Iterationsverlauf (nur in Matlab) und Grafik

% Reelle Loesung von x=g(x)
% Fixpunktsituation
disp('Grafik: Fixpunktsituation')
clf;
if gra==1,
x = linspace(a,b,80);
y = feval(g,x);
subplot(3,1,1);
plot(x,x,'b',x,y,'g')
axis([a b a b])
title('Fixpunktsituation  x=g(x), x in [a,b]')
text(0.85,0.9,'x')
text(0.15,0.9,'g(x)')
xlabel('x')
print aivgr1.ps -dps  % wird ignoriert
pause                 % wird ignoriert
end;

% Ueberpruefung der Voraussetzungen des Fixpunktsatzes
% Ableitung berechnen und zeichnen
disp('Grafik: Ueberpruefung der Voraussetzungen des Fixpunktsatzes')
if gra==1,
ys = feval(gs,x);
eins = ones(max(size(x)));
subplot(3,1,2);
plot(x,ys,'b',x,0*y,'k',x,eins,'b',x,-eins,'b')
axis([a b -1.5 1.5])
title('Ableitung  g(x), x in [a,b]')
text(0.85,-0.6,'g(x)')
xlabel('x')
% print aivgr2.ps -dps
% pause
end;

% Fixpunktiteration, Picard-Verfahren
% Fixpunkt maximal genau bestimmen
disp('Fixpunktiteration')
format long;
x1 = x0;
ndiff = 1;
j = 0;
```

```
while (j<1000) & (ndiff>1e-15),
  j = j+1;
  xk = feval(g,x1);
  ndiff = abs(xk-x1);
  x1 = xk;
end;
xexakt = xk;

% diary aiv2.pro
% diary on
disp('Fixpunkt maximal genau bestimmt, tol=1e-15')
s1 = sprintf('%.13e',xexakt);
disp(['Fixpunkt xs = ',s1,' nach ',num2str(j),' Iterationen'])
pause
disp(' ')
disp('Tabelle zum Iterationsverlauf mit Abbruchbedingungen')
disp(['Max. Iterationsanzahl = ',num2str(nmax),', Toleranz = ',num2str(tol)])
% aenfanglicher Iterationsverlauf
xn(1) = x0;
ndiff = 1;
j = 0;
while (j<nmax) & (ndiff>tol),
  j = j+1;
  xk = xn(j);
  xn(j+1) = feval(g,xk);
  ndiff = abs(xn(j+1)-xk);
end;
k = j;
xs = xn(k+1);
erg = [xs,k];

% Tabelle zum Iterationsverlauf und lineare Konvergenz
if tab==1,
format short;
disp('Lineare Konvergenz, e(-1)=1')
disp('k  q=e(k)/e(k-1)   |x(k)-xexakt|')
disp('-------------------------------------')
q = 1;
for l=1:k+1,
  s1 = num2str(l-1);
  s2 = num2str(abs(xn(l)-xexakt));
  s3 = num2str(abs(xn(l)-xexakt)/q);
  q = abs(xn(l)-xexakt);
if l-1<10,
  s12 = [' ',s1,'    ',s3,'        ',s2];
else
```

```
  s12 = [s1,'    ',s3,'        ',s2];
end;
disp(s12)
end
disp(' ')
end;
% diary off
format long;

% Umspeichern von xn auf Polygonzug und Darstellung des Iterationsverlaufs
disp('Grafik: Fixpunktiteration, Picard-Verfahren')
if gra==1,
for l=1:k
  xng(2*l-1)=xn(l); xng(2*l)=xn(l);
  yng(2*l-1)=xn(l); yng(2*l)=xn(l+1);
end
subplot(3,1,3);
plot(x,x,'b',x,y,'g',xng,yng,'r',xng+0.001,yng+0.001,'r',...
     xng-0.001,yng-0.001,'r',xng(1:floor(k/2)),yng(1:floor(k/2)),'ro')
axis([a b a b])
title('Picard-Verfahren  x(n+1)=g(x(n)), x(0) gegeben')
text(0.85,0.9,'x')
text(0.15,0.9,'g(x)')
% print aivgr3.ps -dps
end;
% Ende der Funktion aiv2
%-------------------------------------------------------------------------
% g.m
function y = g(x)
   y = exp(-x);

% gs.m
function y = gs(x)
   y = -exp(-x);
%-------------------------------------------------------------------------
```

Maple → Matlab mit nutzerdefiniertem Funktions-m-File und weiteren Funktionen
`aiv2 → 'g', 'gs'`
Grafik von Matlab File `aiv2` exportieren

```
> # Test, ob Funktion existiert
  Matlab[defined]("aiv2",'function');
```

true

```
> # initialize a matrix in Maple and copy it to the Matlab environment
  # Maple -> Matlab-Variable
  x0:=0:
  Matlab[setvar]("x0",x0);
  tol:=1e-6:
  Matlab[setvar]("tol",tol);

> # evaluate a Matlab expression (is a string) using Matlab
  # Ausfuehrung der Funktion
  Matlab[evalM]("res=aiv2(x0,0,1,'g','gs',tol,20,1,1)");

> # get a numerical array or matrix in an open Matlab session
  # Rueckgabe des Resultats als Vektor
  RES:=Matlab[getvar]("res");
  xs:=RES[1];
  k:=round(RES[2]);
```

$$RES := [0.567135490206278402,\ 20.]$$
$$xs := 0.567135490206278402$$
$$k := 20$$

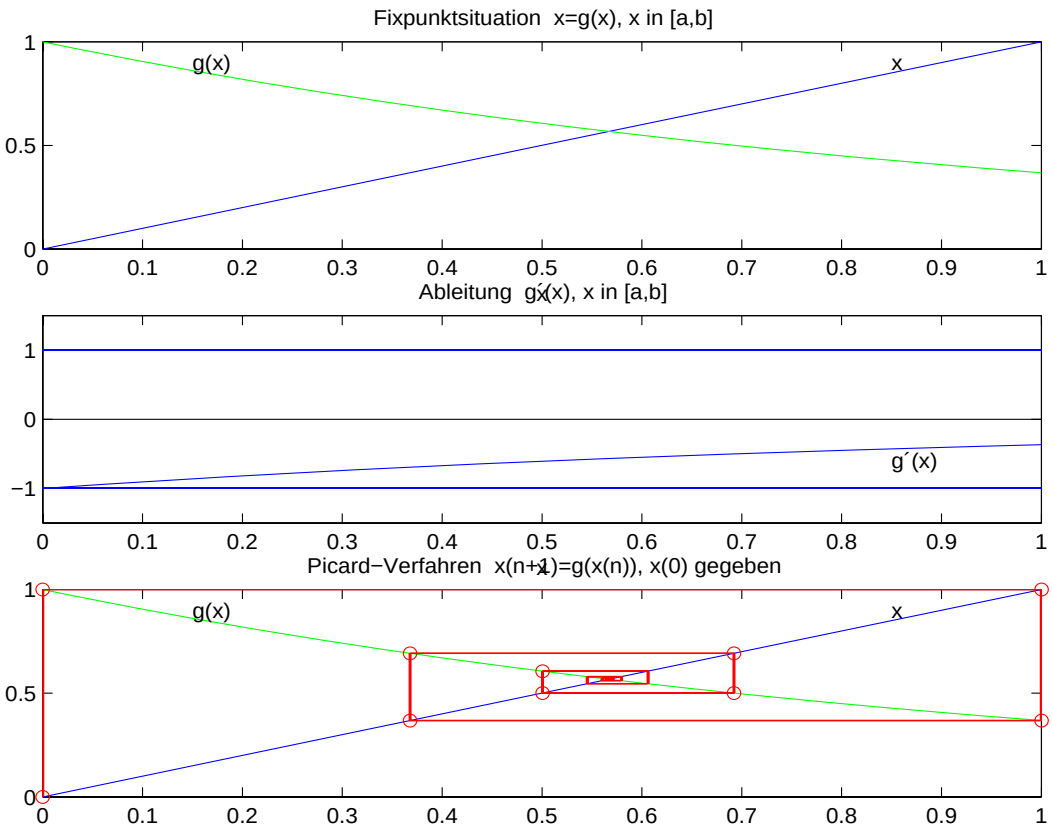


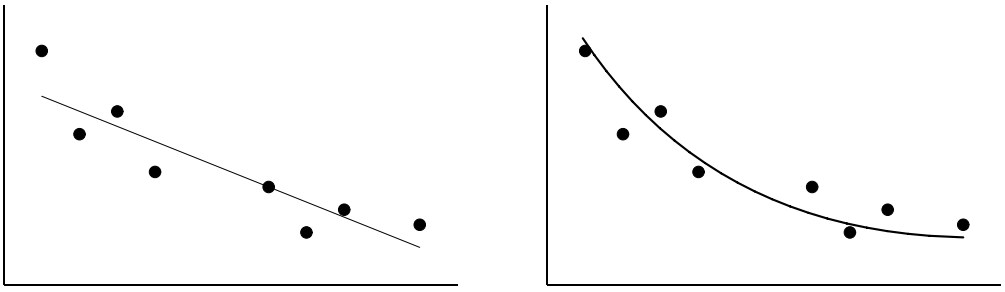
Abb. 2.3 Fixpunktsituation und grafischer Iterationsverlauf zum AIV für $x = e^{-x}$, Speichern der Grafik im aktuellen Figure-Fenster von Matlab mit File → Export... → **Speichern**

Beispiel 3: Ausgleichsrechnung

Die Ausgleichsrechnung wird auch bezeichnet als diskrete Approximation im Mittel, l_2-Approximation, Methode der kleinsten Quadrate oder Gaußsche Fehlerquadratmethode.

Problemstellung

- Zwischen zwei Größen x und y wird ein funktionaler Zusammenhang $y = f(x)$ vermutet, der jedoch häufig nur durch Messungen an den Messpunkten $x_j, j = 0(1)N$, bekannt ist.
- Eine grafische Darstellung der Messwerte y_j über den Messpunkten gibt mitunter Aufschluss über den Typ der zu Grunde liegenden Funktion. In der Abbildung wird nicht eine lineare Funktion wie in der linken Abbildung, sondern eine quadratische Funktion vermutet, d. h. ein Ansatz der Form $\varphi(x) = a_0 + a_1 x + a_2 x^2$ ist sinnvoll.



- Bei der numerischen Approximation wird eine konkrete Funktion $\varphi(x)$ aus einer vorgegebenen Funktionenklasse so durch die "Punktwolke" der Messwerte (x_j, y_j) hindurchgelegt, so dass ihr **Abstand** zu diesen Messpunkten insgesamt minimal wird. Der Abstandsbegriff kann sich sowohl auf das Lot als auch die Entfernung in vertikaler oder horizontaler Richtung beziehen.

Methode der kleinsten Quadrate

Aufgabenstellung

- Abgeschlossene, beschränkte Grundmenge $D \subset \mathbb{R}^r,\ r \geq 1$, und eine unbekannte reelle Funktion $f : D \to \mathbb{R}$.
- Stützstellen x_j und Stützwerte y_j

$$R = \{x_j : \quad x_j \quad = \quad (x_{j1}, x_{j2}, ..., x_{jr}) \in D, \\ x_j \text{ nicht unbedingt paarweise verschieden, } j = 0(1)N\}$$

eine Referenz mit $N+1$ Stützstellen (Messpunkten) und den $N+1$ zugehörigen Stützwerten (Messwerten) $y_j = f(x_j),\ j = 0(1)N$.

- Ansatzfunktion $\varphi(x)$ der Form

$$\varphi(x) = a_0\varphi_0(x) + a_1\varphi_1(x) + ... + a_n\varphi_n(x), \quad a_i \in \mathbb{R},$$

  mit vorgegebenen $n+1$ Basisfunktionen $\varphi_i(x),\ i = 0(1)n$.

- Approximationsforderung (Approximationsbedingung)

$$F = F(a_0, a_1, ..., a_n) = \frac{1}{2}\sum_{j=0}^{N} (y_j - \varphi(x_j))^2 \quad \longrightarrow \quad \min.$$

- Approximationsaufgabe
  Gesucht sind Koeffizienten $a_0, a_1, ..., a_n$ einer approximierenden Funktion $\varphi(x)$, so dass der Wert F minimal ist.

Als Typen von Ansatzfunktionen betrachten wir hier Polynome in $\mathbb{R}^1$

$$\varphi(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n, \quad a_i \in \mathbb{R},$$

mit den Basisfunktionen $\varphi_i(x) = x^i$.

Drei Fragen sind zu beantworten.

1. Existiert zu jeder gegebenen Referenz R mit $N+1$ Punkten eine approximierende Funktion $\varphi(x)$ aus der gegebenen Funktionenklasse ?
2. Ist $\varphi(x)$ eindeutig bestimmt?
3. Wie kann $\varphi(x)$ effektiv konstruiert werden?

Die Herleitung der Methode der kleinsten Quadrate von C.F. Gauß ist zugleich die Konstruktion der Ausgleichsfunktion.

Gesucht ist ein Koeffizientensatz $a_0, a_1, ..., a_n$, für den die Summe der Abweichungsquadrate, die Ausgleichsfunktion

$$F(a_0, a_1, ..., a_n) = \frac{1}{2}\sum_{j=0}^{N} \left[y_j - \sum_{i=0}^{n} a_i\varphi_i(x_j) \right]^2$$

minimal wird.
Notwendige Bedingungen für ein (lokales) Minimum sind

$$\frac{\partial F}{\partial a_k} = \sum_{j=0}^{N} \left[\sum_{i=0}^{n} a_i\varphi_i(x_j) - y_j \right] \varphi_k(x_j) = 0,\ k = 0(1)n.$$

Auflösung der Klammern und Umstellen liefert

$$\sum_{i=0}^{n}\left[\sum_{j=0}^{N}\varphi_k(x_j)\varphi_i(x_j)\right]a_i \quad = \quad \sum_{j=0}^{N}\varphi_k(x_j)y_j, \quad k=0(1)n.$$

Durch Einführung der Gaußschen Klammern (Skalarprodukte, Gewicht $\omega = 1$)

$$\begin{aligned}(\varphi_k,\varphi_i) &= \sum_{j=0}^{N}\varphi_k(x_j)\varphi_i(x_j)\\ (\varphi_k,y) &= \sum_{j=0}^{N}\varphi_k(x_j)y_j, \quad \varphi_k, y \in l_{2,1}\end{aligned}$$

lassen sich die $n+1$ Bedingungen für die $n+1$ Unbekannten a_i in der Form der **Normalgleichungen** notieren.

$$\begin{aligned}&(\varphi_0,\varphi_0)a_0+(\varphi_0,\varphi_1)a_1+\cdots+(\varphi_0,\varphi_n)a_n=(\varphi_0,y)\\ &(\varphi_1,\varphi_0)a_0+(\varphi_1,\varphi_1)a_1+\cdots+(\varphi_1,\varphi_n)a_n=(\varphi_1,y)\\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\ &(\varphi_n,\varphi_0)a_0+(\varphi_n,\varphi_1)a_1+\cdots+(\varphi_n,\varphi_n)a_n=(\varphi_n,y)\end{aligned}$$

Die Determinante der Koeffizientenmatrix

$$\det(G) = \det(G(\varphi_0,...,\varphi_n)) = \det((\varphi_i,\varphi_j)_{i,j=0}^{n})$$

heißt auch Gramsche Determinante.

Wir setzen voraus, dass für jede Referenz $R \subset D$ die Haarsche oder Vandermondesche Matrix

$$\Phi = \begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \varphi_2(x_0) & \cdots & \varphi_n(x_0)\\ \varphi_0(x_1) & \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_n(x_1)\\ \varphi_0(x_2) & \varphi_1(x_2) & \varphi_2(x_2) & \cdots & \varphi_n(x_2)\\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots\\ \varphi_0(x_N) & \varphi_1(x_N) & \varphi_2(x_N) & \cdots & \varphi_n(x_N)\end{pmatrix}$$

den maximalen Rang $r = \min(n, N) + 1$ besitzt.
Damit erfüllen die $n+1$ Basisfunktionen $\varphi_i(x)$, $i = 0(1)n$, die verallgemeinerte HAARsche Bedingung.
Das Normalgleichungssystem notieren wir damit in der Kurzform

$$Ga \;=\; \Phi^T\Phi a \;=\; \Phi^T y, \quad G \text{ Koeffizientenmatrix.}$$

Existenz und Eindeutigkeit

Falls $N \geq n$ und die verallgemeinerte HAARsche Bedingung erfüllt ist, liefern die Normalgleichungen das absolute Minimum der Ausgleichsfunktion $F(a_0, a_1, ..., a_n)$. Die Approximationsaufgabe ist für beliebige Referenzen $R \subset D$ stets eindeutig lösbar

Bemerkungen

(1) Falls $N \geq n$ ist müssen zur Sicherung von $\text{rang}(\Phi) = n + 1$ unter den $N + 1$ Stützstellen x_j mindestens $n + 1$ verschiedene sein.
Der Fall $N < n$ führt auf ein unterbestimmtes System (zu wenig Bedingungen) und wird i. Allg. wegen der Mehrdeutigkeit der Lösung nicht weiter betrachtet.

(2) Sei $N = n$. Das System von Basisfunktionen $\{\varphi_i(x)\}$, das mit einer beliebigen Referenz mit paarweise verschiedenen Stützstellen eine stets reguläre Matrix Φ bildet, heißt auch Haarsches, Tschebyscheff-, T- oder unisolventes System. Die Aufgabe ist dann wie ein Interpolationsproblem eindeutig lösbar.

(3) Sei $N \geq n$. Analoge Aussage zu (2) mit Referenz, die mindestens $n + 1$ verschiedene Stützstellen enthalten muss. Die Regularität von Φ wird ersetzt durch $\text{rang}(\Phi) = n + 1$ oder $\det(G) \neq 0$.

Unter Berücksichtigung der Erfüllung des Normalgleichungssystems durch die Koeffizienten $a_i,\ i = 0, 1, ..., n$, ergibt sich der Approximationsfehler in der l_2-Norm

$$\begin{aligned} F_{min} &= \tfrac{1}{2} \sum_{j=0}^{N} (y_j - \varphi(x_j))^2, \qquad \varphi(x_j) = \sum_{i=0}^{n} a_i \varphi_i(x_j) \\ &= \tfrac{1}{2} \Big[\sum_{j=0}^{N} y_j^2 - \sum_{i=0}^{n} a_i(\varphi_i, y) \Big] \ \geq \ 0. \end{aligned}$$

Zusammenfassung zum Ausgleich durch Polynome in $\mathbb{R}^1$
Ansatzfunktion

$$\varphi(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n, \quad a_i \in \mathbb{R},$$

mit Basisfunktionen (Monome) $\varphi_i(x) = x^i,\ i = 0(1)n$.

Spezialfälle

$$\begin{aligned} \varphi(x) &= a_0 + a_1 x && \text{(linearer Ausgleich)} \\ \varphi(x) &= a_0 + a_1 x + a_2 x^2 && \text{(quadratischer Ausgleich)} \\ \varphi(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 && \text{(kubischer Ausgleich)} \end{aligned}$$

Normalgleichungen
Mit den Gaußschen Klammern (Skalarprodukten)

$$\begin{aligned}(\varphi_k, \varphi_i) &= \sum_{j=0}^{N} x_j^k x_j^i = \sum_{j=0}^{N} x_j^{k+i},\\ (\varphi_k, y) &= \sum_{j=0}^{N} x_j^k y_j, \quad i,k = 0(1)n,\end{aligned}$$

lassen sich die $n+1$ Normalgleichungen für die $n+1$ Unbekannten a_i in der folgenden Form notieren.

$$\begin{pmatrix} N+1 & \sum x_j & \sum x_j^2 & \cdots & \sum x_j^n \\ \sum x_j & \sum x_j^2 & \sum x_j^3 & \cdots & \sum x_j^{n+1} \\ \sum x_j^2 & \sum x_j^3 & \sum x_j^4 & \cdots & \sum x_j^{n+2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum x_j^n & \sum x_j^{n+1} & \sum x_j^{n+2} & \cdots & \sum x_j^{2n} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{pmatrix} = \begin{pmatrix} \sum y_j \\ \sum x_j y_j \\ \sum x_j^2 y_j \\ \cdots \\ \sum x_j^n y_j \end{pmatrix}$$

Im allgemeinen Fall sind zum Aufstellen des LGS $(n+1)(n+2)$ Skalarprodukte auszuwerten. Dazu dient das FALKsche Schema.

					1	x_N	x_N^2	$\cdots$	x_N^n	y_N
					1	x_{N-1}	x_{N-1}^2	$\cdots$	x_{N-1}^n	y_{N-1}
					$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
					1	x_1	x_1^2	$\cdots$	x_1^n	y_1
					1	x_0	x_0^2	$\cdots$	x_0^n	y_0
1	1	...	1	1	$N+1$	$\sum x_j$	$\sum x_j^2$	$\cdots$	$\sum x_j^n$	$\sum y_j$
x_N	x_{N-1}	...	x_1	x_0	$\sum x_j$	$\sum x_j^2$	$\sum x_j^3$	$\cdots$	$\sum x_j^{n+1}$	$\sum x_j y_j$
x_N^2	x_{N-1}^2	...	x_1^2	x_0^2	$\sum x_j^2$	$\sum x_j^3$	$\sum x_j^4$	$\cdots$	$\sum x_j^{n+2}$	$\sum x_j^2 y_j$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
x_N^n	x_{N-1}^n	...	x_1^n	x_0^n	$\sum x_j^n$	$\sum x_j^{n+1}$	$\sum x_j^{n+2}$	$\cdots$	$\sum x_j^{2n}$	$\sum x_j^n y_j$

Existenz und Eindeutigkeit

Falls $N \geq n$ ist und mindestens $n+1$ Stützstellen x_j verschieden sind, so liefern die Normalgleichungen das absolute Minimum der Ausgleichsfunktion $F(a_0, a_1, ..., a_n)$. Die Approximationsaufgabe ist für beliebige Intervalle $\mathrm{I} = [a, b]$ und beliebige Referenzen $R \subset \mathrm{I}$ mit mindestens $n+1$ verschiedene Stützstellen stets eindeutig lösbar.

Lösung des Normalgleichungssystems

Zunächst beachte man, dass die Kondition der Koeffizientenmatrix G durchaus schlecht werden kann und $\det(G) = \det(\Phi^T \Phi) \approx 0$ für nahe beieinander liegende Stützstellen ist.

Wegen der Symmetrie und gleicher Summen kann man im FALKschen Schema den Aufwand reduzieren, so dass nur $2n + (n+1) = 3n+1$ Skalarprodukte bzw. Summen zu bestimmen sind.

Folgendes Rechenschema ist günstiger.

j	x	y	xy	x^2	x^2y	x^3	x^3y	x^4	$\cdots$	x^{2n}
0	x_0	y_0	x_0y_0	x_0^2	$x_0^2y_0$	x_0^3	$x_0^3y_0$	x_0^4	$\cdots$	x_0^{2n}
1	x_1	y_1	x_1y_1	x_1^2	$x_1^2y_1$	x_1^3	$x_1^3y_1$	x_1^4	$\cdots$	x_1^{2n}
2	x_2	y_2	x_2y_2	x_2^2	$x_2^2y_2$	x_2^3	$x_2^3y_2$	x_2^4	$\cdots$	x_2^{2n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
N	x_N	y_N	x_Ny_N	x_N^2	$x_N^2y_N$	x_N^3	$x_N^3y_N$	x_N^4	$\cdots$	x_N^{2n}
$N+1$	$\sum x_j$	$\sum y_j$	$\sum x_jy_j$	$\sum x_j^2$	$\sum x_j^2y_j$	$\sum x_j^3$	$\sum x_j^3y_j$	$\sum x_j^4$	$\cdots$	$\sum x_j^{2n}$

Linearer Ausgleich

Lineare Ansatzfunktion $\varphi(x) = a_0 + a_1 x$

Normalgleichungssystem

$$\begin{aligned} a_0(N+1) + a_1 \sum_{j=0}^{N} x_j &= \sum_{j=0}^{N} y_j \\ a_0 \sum_{j=0}^{N} x_j + a_1 \sum_{j=0}^{N} x_j^2 &= \sum_{j=0}^{N} x_j y_j \end{aligned}$$

mit der Koeffizientendeterminante

$$D = (N+1)\sum_{j=0}^{N} x_j^2 - \left(\sum_{j=0}^{N} x_j\right)^2 \neq 0$$

Lösung mit CRAMERscher Regel

$$
\begin{aligned}
a_0 &= \left(\sum_{j=0}^{N} y_j \sum_{j=0}^{N} x_j^2 - \sum_{j=0}^{N} x_j \sum_{j=0}^{N} x_j y_j \right) / D \\
a_1 &= \left((N+1) \sum_{j=0}^{N} x_j y_j - \sum_{j=0}^{N} x_j \sum_{j=0}^{N} y_j \right) / D
\end{aligned}
$$

Quadratischer Ausgleich

Quadratische Ansatzfunktion $\varphi(x) = a_0 + a_1 x + a_2 x^2$

Normalgleichungssystem

$$
\begin{aligned}
a_0 \ (N+1) + a_1 \sum_{j=0}^{N} x_j + a_2 \sum_{j=0}^{N} x_j^2 &= \sum_{j=0}^{N} y_j \\
a_0 \sum_{j=0}^{N} x_j + a_1 \sum_{j=0}^{N} x_j^2 + a_2 \sum_{j=0}^{N} x_j^3 &= \sum_{j=0}^{N} x_j y_j \\
a_0 \sum_{j=0}^{N} x_j^2 + a_1 \sum_{j=0}^{N} x_j^3 + a_2 \sum_{j=0}^{N} x_j^4 &= \sum_{j=0}^{N} x_j^2 y_j
\end{aligned}
$$

Lösung mit CRAMERscher Regel sowie SARRUSscher Regel zur Determinantenberechnung.

Die Kondition des entstehenden Normalgleichungssystems wird mit wachsender Dimension n immer schlechter. Um z. B. große Elemente der Koeffizientenmatrix zu vermeiden, empfiehlt sich eine Koordinatentransformation der Stützstellen x_j.
Damit erhalten wir ein skaliertes System mit besseren Eigenschaften.

Die Matlab Funktion `polyfit`

Matlab stellt für die Methode der kleinsten Quadrate mit Polynomen das Funktions-m-File `polyfit` zur Verfügung.

```
p = polyfit(x,y,n)
```

Die Funktion liefert in Form eines Zeilenvektors die Koeffizienten $p(1..n{+}1)$ des Ausgleichspolynoms $\varphi(x) = \sum_{i=0}^{n} a_i x^i$ vom Grade n zu den Daten x, y, also $p(i) = a_{n+1-i}$, $i = 1, 2, ..., n+1$.
Aus diesen Koeffizienten kann man in einem ausgewählten Intervall den Verlauf des Ausgleichspolynoms mittels `polyval` erzeugen. Numerische und grafische Vergleiche von $\varphi(x)$ mit $y = f(x)$ (falls y durch eine Funktion generiert wurde) sind möglich.

In `polyfit` wird nicht das Normalgleichungssystem aufgestellt, sondern das System mit der rechteckigen Haarschen Matrix Φ.

$$\sum_{i=n(-1)}^{0} x_j^i a_i \;\sim\; y_j, \quad j = 0, 1, ..., N.$$

In der Links-Division $x = A\backslash b$ wird eine quadratische Matrix mittels Gauß-Elimination faktorisiert und damit dann das LGS $Ax = b$ gelöst.
Für eine rechteckige Matrix A liegt die Householder-Orthogonalisierung (QR-Verfahren) zu Grunde, und man sucht die Lösung im Sinne der Methode der kleinsten Quadrate (least squares sense).
Die Funktion in der Matlab Toolbox ist folgende.

```
function [p,S] = polyfit(x,y,n)

%POLYFIT Polynomial curve fitting.
%   p = POLYFIT(x,y,n) finds the coefficients of a polynomial p(x) of
%   degree n that fits the data, p(x(i))~=y(i), in a least-squares
%   sense.
%
%   [p,S] = POLYFIT(x,y,n) returns the polynomial coefficients p and a
%   matrix S for use with POLYVAL to produce error estimates on
%   predictions.
%   If the errors in the data, y, are independent normal with constant
%   variance, POLYVAL will produce error bounds which contain
%   at least 50% of the predictions.
%
%   See also POLY, POLYVAL, ROOTS.

%   J.N. Little 4-21-85, 8-23-86; CBM, 12-27-91 BAJ, 5-7-93.
%   Copyright (c) 1984-94 by The MathWorks, Inc.
...
```

Approximation der Daten $(t(0:N), y(0:N))$ mit der Methode der kleinsten Quadrate mit Polynomen $p_n(t) = a_n t^n + a_{n-1} t^{n-1} + ... + a_0,\ n \leq N$

`p=polyfit(t,y,n)` -> `p[1]=a[n], p[2]=a[n-1],..., p[n+1]=a[0]`

Matlab im *double*-Format mit 15-stelliger Anzeige
Beispiel

```
t=(0:50)', y=(t+randn(51,1))/10, y(5)=2, y(15)=3, y(25)=4, y(35)=5.5

p=polyfit(t,y,1),  n=1  -> p =   0.09715338354849   0.19320404185047
```

```
 p=polyfit(t,y,10), n=10 -> Warning: Polynomial is badly conditioned.
                                     Remove repeated data points
                                     or try centering and scaling as
                                     described in HELP POLYFIT.

 p = 0.00000000000032 -0.00000000007798  0.00000000805700 -0.00000045392254
     0.00001508014528 -0.00029694725494  0.00325640560471 -0.01598806020339
    -0.00094880145930  0.30030787255682 -0.04684565372981
```

Aufruf einer nutzerdefinierten Matlab-Funktion unter Maple.
Die folgenden Funktionen muessen in einem Verzeichnis des
“MATLABPATH“ enthalten sein.

```
% usepolyfit.m
function p=usepolyfit(n)

[t,y]=gendata;
p=polyfit(t,y,n);
f=polyval(p,t);
a=min(t);  b=max(t);
y1=min(y); y2=max(y);
tt=a:(b-a)/100:b;
tt=tt(:);
pl1=polyval(p,tt);

clf;
% Plot von Messpunkten und Ausgleichspolynom
subplot(2,1,1);
plot(t,y,'r*',tt,pl1,'b');
legend('DATA','l2-APPROX',0);
% Bestimmung des Zeichenbereiches
dd=10;
axis([a-abs(a)/dd b+abs(a)/dd y1-abs(y1)/dd y2+abs(y1)/dd]);

% Berechnen der Fehler an den Messpunkten
errorl2=y-polyval(p,t);
subplot(2,1,2);
plot(tt,0,'k');
hold on;
p1=stem_(t,errorl2,'b','filled');
legend(p1,'l2-Error',0);
% Bestimmung des Zeichenbereiches
y1=min(min(errorl2),-1);  % y1=-1;
y2=max(errorl2);          % y2= 1;
dd=10;
```

```
axis([a-abs(a)/dd b+abs(a)/dd y1-abs(y1)/dd y2+abs(y1)/dd]);
hold off;
% Ende usepolyfit
-------------------------------------------------------------------------

% stem_.m
function hh = stem(varargin)

%STEM   Discrete sequence or "stem" plot.
%   STEM(Y) plots the data sequence Y as stems from the x axis
%   terminated with circles for the data value.
%
%   STEM(X,Y) plots the data sequence Y at the values specfied
%   in X.
%
%   STEM(...,'filled') produces a stem plot with filled markers.
%
%   STEM(...,'LINESPEC') uses the linetype specifed for the stems and
%   markers.  See PLOT for possibilities.
%
%   H = STEM(...) returns a vector of line handles.
%
%   See also PLOT, BAR, STAIRS.

%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.12 $  $Date: 1997/11/21 23:46:52 $

nin = nargin;
fill = 0;
ls = '-';
ms = 'o';
col = '';

% Parse the string inputs
while isstr(varargin{nin}),
  v = varargin{nin};
  if ~isempty(v) & strcmp(lower(v(1)),'f')
    fill = 1;
    nin = nin-1;
  else
    [l,c,m,msg] = colstyle(v);
    if ~isempty(msg), error(sprintf('Unknown option "%s".',v)); end
    if ~isempty(l), ls = l; end
    if ~isempty(c), col = c; end
    if ~isempty(m), ms = m; end
    nin = nin-1;
```

```
  end
end
error(nargchk(1,2,nin));

[msg,x,y] = xychk(varargin{1:nin},'plot');
if ~isempty(msg), error(msg); end
if min(size(x))==1, x = x(:); end
if min(size(y))==1, y = y(:); end

% Set up data using fancing indexing
[m,n] = size(x);
xx = zeros(3*m,n);
xx(1:3:3*m,:) = x;
xx(2:3:3*m,:) = x;
xx(3:3:3*m,:) = NaN;

[m,n] = size(y);
yy = zeros(3*m,n);
yy(2:3:3*m,:) = y;
yy(3:3:3*m,:) = NaN;

cax = newplot;
next = lower(get(cax,'NextPlot'));
hold_state = ishold;
if fill, h1 = plot(x,y,[col,ms],'parent',cax);
         else h1 = plot(x,y,[col,'.'],'parent',cax); end;
hold on;
h2 = plot(xx,yy,[col,ls],'parent',cax);
for i=1:length(h1), % Make the color of the 'o's the same as the lines.
  c = get(h2(i),'Color');
  set(h1(i),'Color',c)
  if fill, set(h1(i),'MarkerFaceColor',c), end
end
h = [h1;h2];
if ~hold_state, set(cax,'NextPlot',next); end
if nargout>0, hh = h; end
% Ende stem_
-------------------------------------------------------------------------
% gendata.m
lbnd=0; upperbnd=50; step=1;
noise=1;
t=(lbnd:step:upperbnd)';        % Spaltenvektor
y=(t+randn([max(size(t)),1])*noise)/10;
y(5)=2; y(15)=3; y(25)=4; y(35)=5.5;
% Ende gendata
-------------------------------------------------------------------------
```

Maple → Matlab mit nutzerdefiniertem Funktions-m-File und weiteren Funktionen `usepolyfit → gendata, stem_`

```
> # Test, ob Funktion existiert
  Matlab[defined]("usepolyfit",'function');
```

$$true$$

```
> # evaluate a Matlab expression (is a string) using Matlab
  # Ausfuehrung, Ausgleich mit Polynom 1. Grades
  Matlab[evalM]("res=usepolyfit(1)"):

> # get a numerical array or matrix in an open Matlab session
  # Rueckgabe des Resultates
  RES:=Matlab[getvar]("res");
       # 2 Koeffizienten des linearen Ausgleichspolynoms a[1]t+a[0]
```

$$RES := [0.0956429021907284628,\ 0.247437874560110432]$$

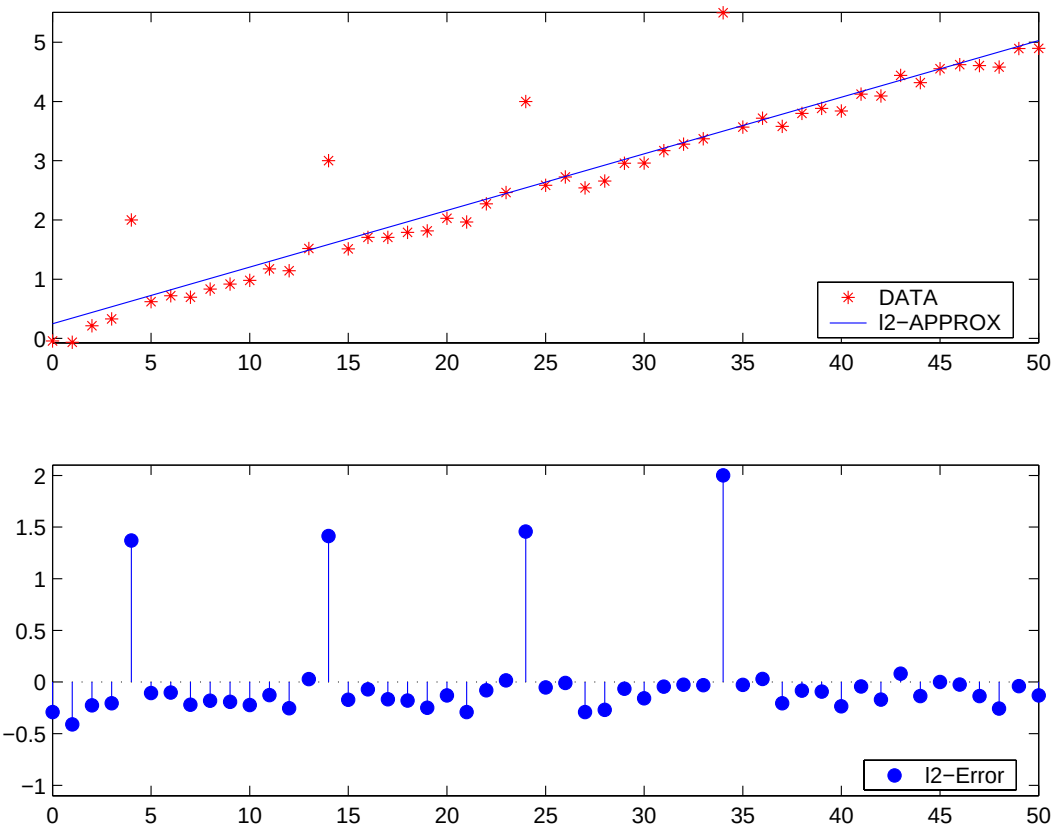


Abb. 2.4 Ausgleichsgerade und Abweichungen,
Speichern der Grafik im aktuellen Figure-Fenster von Matlab mit
File → Export... → **Speichern**

```
> # Ausfuehrung, Ausgleich mit Polynom 10. Grades
  Matlab[evalM]("res=usepolyfit(10)"):

> # Rueckgabe des Resultates
  RES:=Matlab[getvar]("res");
        # 11 Koeffizienten des Ausgleichspolynoms a[10]t^10+...+a[0]
  RES[1];
  RES[2];
  RES[3];
  RES[11];

  whattype(RES);
  Dimension(RES);
  m:=LinearAlgebra[Dimension](RES);
  for i from 1 to m do RES[i]; end do;
```

$$RES := \begin{bmatrix} \text{11 Element Row Vector} \\ \text{Data Type: float[8]} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

$$0.262428613660229993\,10^{-12}$$
$$-0.643738317752849694\,10^{-10}$$
$$0.660639281959337013\,10^{-8}$$
$$-0.0679064614781511450$$

$$\text{Dimension}\left(\begin{bmatrix} \text{11 Element Row Vector} \\ \text{Data Type: float[8]} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}\right)$$

$$Vector_{row}$$

$$m := 11$$

$$0.262428613660229993\,10^{-12}$$
$$-0.643738317752849694\,10^{-10}$$
$$0.660639281959337013\,10^{-8}$$
$$-0.366364730921782982\,10^{-6}$$
$$0.000011789658351579823 8$$
$$-0.000217334579030373172$$
$$0.002021203409024967 61$$
$$-0.004195902916030534 90$$
$$-0.063735213898866077 8$$
$$0.447525055982909648$$
$$-0.0679064614781511450$$

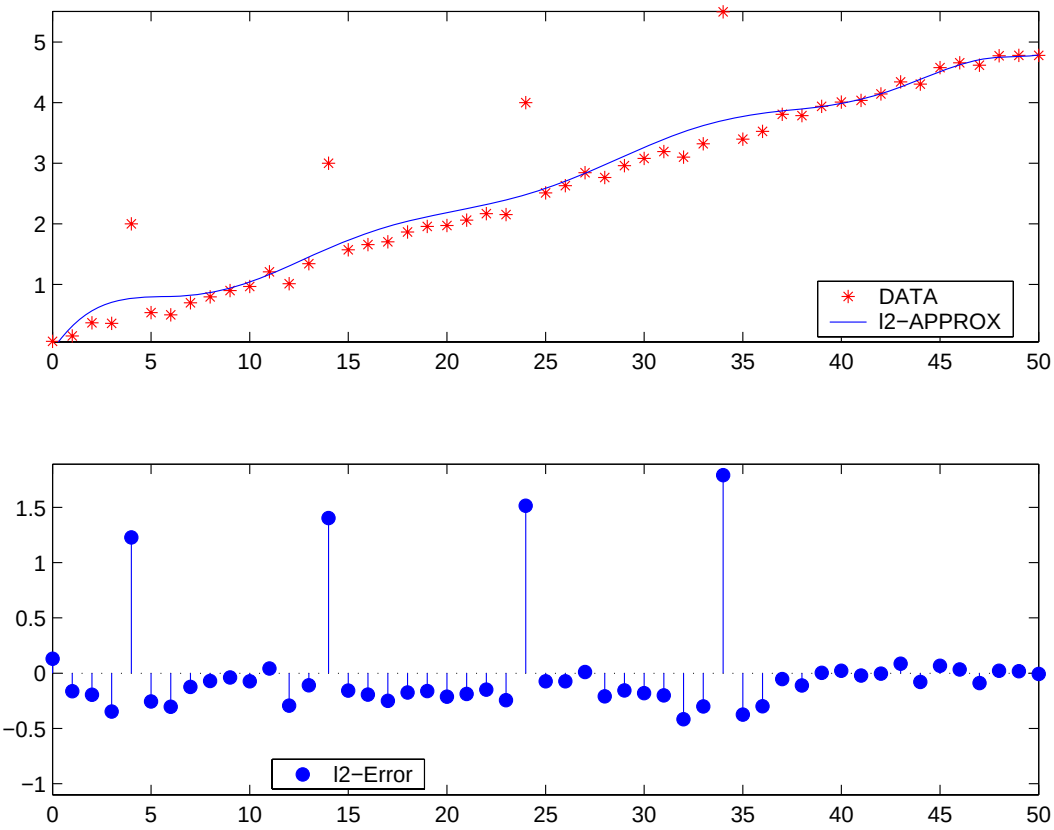


Abb. 2.5 Ausgleichspolynom 10. Grades und Abweichungen

2.12 Beispiel

Eine einfache Optimierungsaufgabe mit Kurvendiskussion soll den Abschluss bilden. Dabei soll gezeigt werden, wie man sich schrittweise an das Problem und seine Lösung "herantastet", was dann auch bei der Implementierung im CAS sichtbar sein wird.

Problemstellung

Gegeben ist die Funktion

$$f_r(x) = d\sin(x)\left(1 - \frac{\cos(x)}{\sqrt{1.519^2 - \sin(x)^2}}\right)$$

mit dem reellen Parameter $d > 0$.
Gesucht ist auf dem symmetrischen Intervall $I_\varepsilon = [-\varepsilon, \varepsilon],\ \varepsilon < \pi/2$, eine Approximation dieser durch die einfachere trigonometrische Funktion

$$f_l(x) = c\tan(x).$$

Das heißt, man bestimme in diesem Ansatz den reellen Parameter $c \geq 0$. Somit erhält man das Optimierungsproblem

$$\max_{x \in I_\varepsilon} |f_l(x) - f_r(x)| \ \longrightarrow \ \min_{c \geq 0}$$

bzw. mit der Maximumnorm

$$c_{opt} \ = \ \arg\min_{c \geq 0} \|f_l - f_r\|_\infty.$$

Zunächst machen wir einige Vorbetrachtungen zur Aufgabe.

1. Ein typische Größe für den Intervallparameter ε ist $\pi/10$. Wir wählen hier den Wert 0.3. Aber es wird auch ein Blick auf die Situation mit $\varepsilon = 1$ geworfen.

2. Die Aufgabe kann skaliert und damit vereinfacht werden, indem $d = 1$ gesetzt wird. Dies werden wir nunmehr voraussetzen, indem wir $\tilde{c} = c/d$ verwenden.

3. Die Funktionen $f_l(x)$ und $f_r(x)$ sind auf I_ε nicht identisch.

4. Wir definieren den Abstand (Abweichung) der beiden Funktionen

   $$f(x) \ = \ f_l(x) - f_r(x).$$

   Die Funktion $f(x)$ erfüllt wegen der Eigenschaften der darin enthaltenen trigonometrischen Funktionen die Beziehungen $f(x) = -f(-x)$ (Antisymmetrie) und somit $f(0) = 0$.

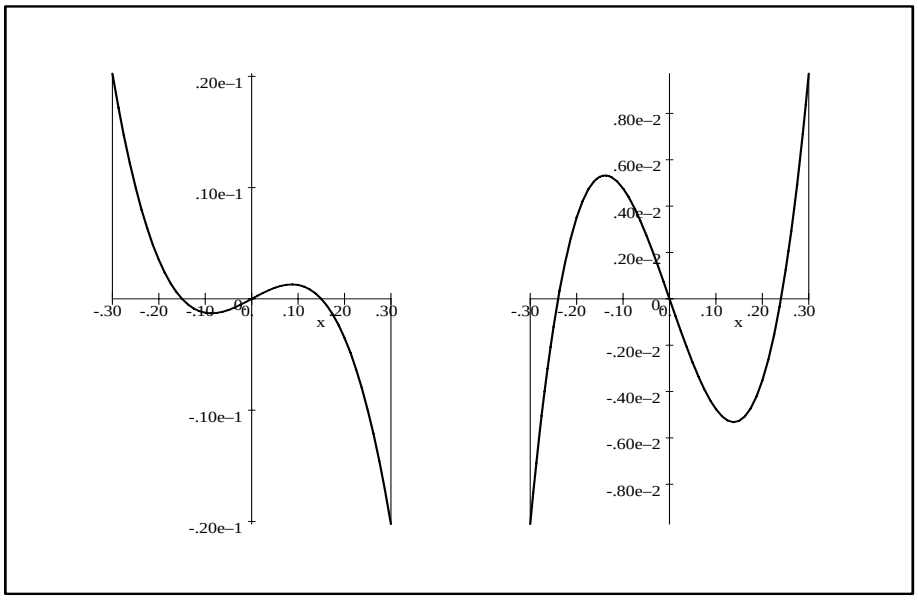


Abb. 2.6 Charakteristische Verläufe von $f(x)$ auf $I_\varepsilon,\ \varepsilon = 0.3$

5. Mit dem genannten Verhalten von $f(x)$ ist c so zu bestimmen, dass alle maximalen Abweichungen von Null dieser Funktion betragsgleich werden.
   Diese Abweichungen ergeben sich aus einer Kurvenbetrachtung und -diskussion, wobei die Gleichheit der Beträge der Funktionswerte an den Intervallenden sowie der Extremwerte angestrebt wird.

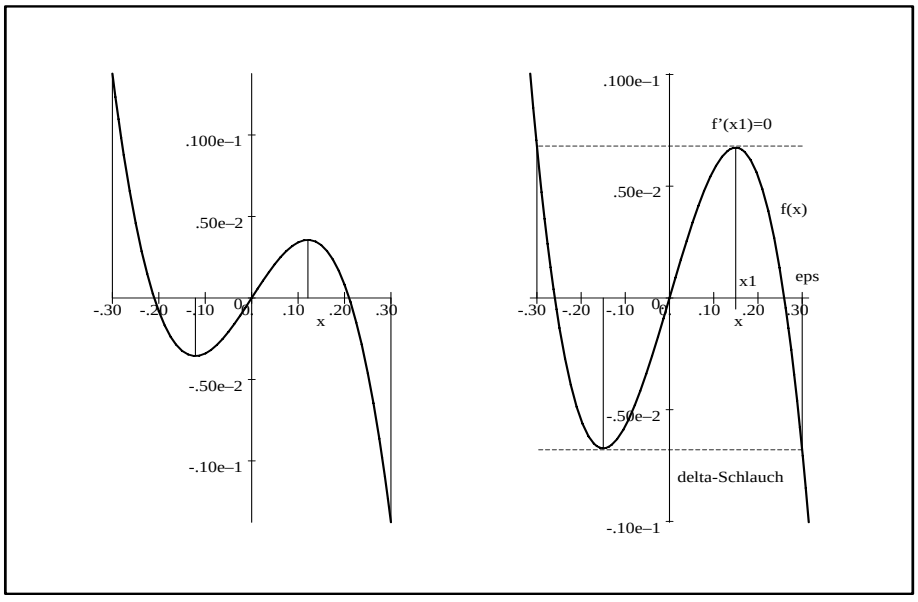


Abb. 2.7 "Verschiebung" von Parameter $\tilde{c}$ so, dass $f(x)$ auf $I_\varepsilon,\ \varepsilon = 0.3$, in δ-Schlauch liegt

Es gelten $|f(\varepsilon)| = |f(x_1)|$ und $f'(x_1) = 0$, wobei

$$\begin{aligned} f(x) &= \tilde{c}\,\tan(x) - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.519^2 - \sin(x)^2}}\Big), \\ &= \tilde{c}\,\tan(x) - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big), \end{aligned}$$

$$\begin{aligned} f'(x) &= \tilde{c}\,[1 + \tan(x)^2] - \cos(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big) \\ &\quad - \sin(x)^2\Big(\frac{1}{\sqrt{1.307361 + \cos(x)^2}} - \frac{\cos(x)^2}{\sqrt{[1.307361 + \cos(x)^2]^3}}\Big). \end{aligned}$$

6. Abschätzung von Grenzen für $\tilde{c}$.
   Für $x \geq 0$ und $\tilde{c} \in [0,1]$ erkennt man an den Intervallgrenzen $\tilde{c} = 0$ bzw. 1 folgendes typisches Verhalten der Funktion $f(x)$.

$$\tilde{c} = 0, \quad f(x) = 0 - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big) \leq 0,$$

$$\begin{aligned} \tilde{c} = 1, \quad f(x) &= \tan(x) - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big) \\ &= \sin(x)\Big[\frac{1}{\cos(x)} - \Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big)\Big] \geq 0. \end{aligned}$$

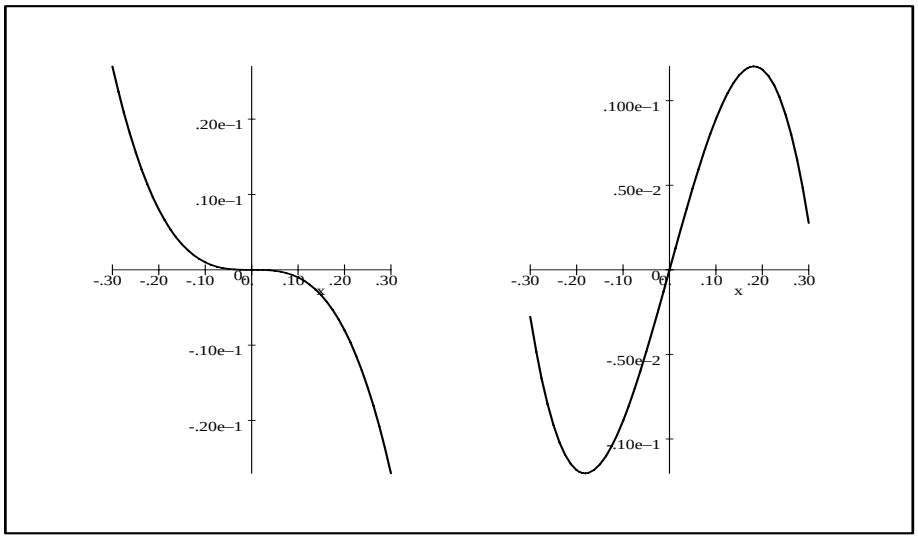


Abb. 2.8 Links: $\tilde{c} = 0$: $f(x) \leq 0$ auf $[0, \varepsilon]$, $\varepsilon = 0.3$,
rechts: $\tilde{c} = 1$: $f(x) \geq 0$ auf $[0, \varepsilon]$

Das Intervall für $\tilde{c}$, wo die Funktion $f(x)$ insgesamt ihr Vorzeichen von Minus auf Plus wechselt, soll etwas genauer bestimmt werden.
Dazu betrachten wir die beiden Bedingungen $f'(0) = 0$, was auf die untere Grenze c_u führt, bzw. $f(\varepsilon) = 0$, welche die obere Grenze c_o liefert.

$$\begin{aligned} f'(0) = 0, \quad 0 &= \tilde{c} - \cos(0)\Big(1 - \frac{\cos(0)}{\sqrt{1.307361 + \cos(0)^2}}\Big) + 0, \\ c_u &= 1 - \frac{1}{\sqrt{1.307361 + 1}} \approx 0.3417, \\ f(\varepsilon) = 0, \quad 0 &= \tilde{c}\tan(\varepsilon) - \sin(\varepsilon)\Big(1 - \frac{\cos(\varepsilon)}{\sqrt{1.307361 + \cos(\varepsilon)^2}}\Big), \\ c_o &= \cos(\varepsilon)\Big(1 - \frac{\cos(\varepsilon)}{\sqrt{1.307361 + \cos(\varepsilon)^2}}\Big) \approx 0.3428. \end{aligned}$$

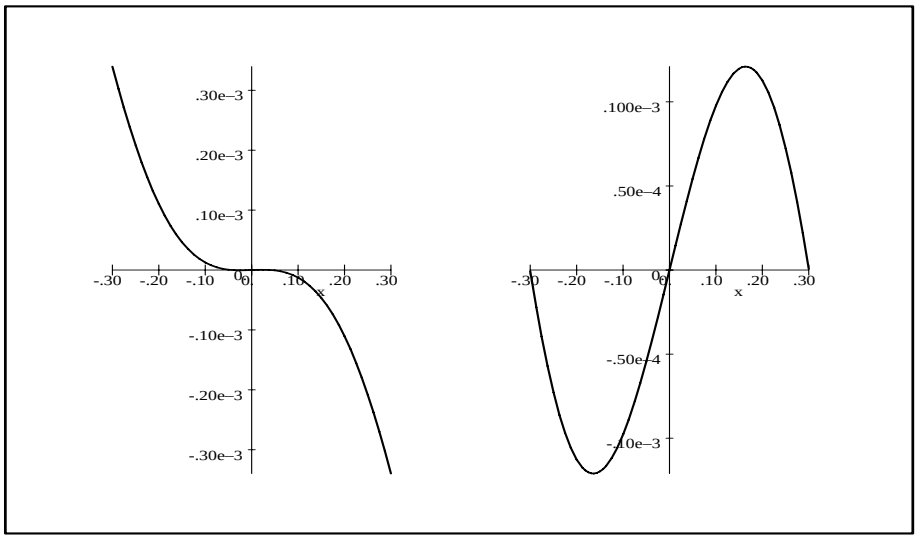


Abb. 2.9 Links: $\tilde{c} = c_u$: $f(x) \le 0$ auf $[0, \varepsilon]$, $f'(0) = 0$, $\varepsilon = 0.3$,
rechts: $\tilde{c} = c_o$: $f(x) \ge 0$ auf $[0, \varepsilon]$, $f(\varepsilon) = 0$

7. Für eine genauere Bestimmung des Parameters $\tilde{c} \in [c_u, c_o]$ für die optimale Situation mit dem δ-Schlauch genügt es, wegen des regulären Übergangs zwischen diesen beiden Randsituationen das einfache Halbierungsverfahren (Bisektion) zu benutzen.

8. Die Situation ändert sich, falls der Wert ε wächst.
So treten bei $\varepsilon = 1$ in der Funktion $f(x)$ zusätzliche Wendepunkte auf. Der kritische Bereich ist nahe den Rändern des Intervalls I_ε.

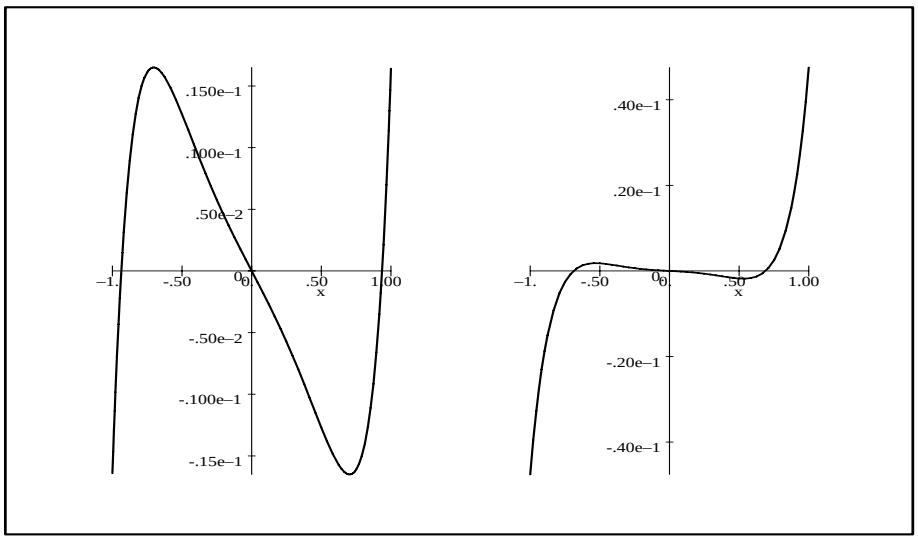


Abb. 2.10 Links: $\tilde{c} = 0.32$: $f(x)$ auf $[-\varepsilon, \varepsilon]$, $\varepsilon = 1$,
rechts: $\tilde{c} = 0.34$: $f(x)$ auf $[-\varepsilon, \varepsilon]$

Rechnungen in Maple (Datei: *vog1.mws*)

Parametervariante 1: $\varepsilon = 0.3$

```
> eps:=0.3;
  Ie:=-eps..eps;      # Intervall [-eps,eps]
```

$$eps := 0.3$$
$$Ie := \text{-0.3 .. 0.3}$$

Definition der Funktionen $f_r(x), f_l(x)$ und erster Vergleich

```
> d:=1;
  fr:=x->d*sin(x)*(1-cos(x)/sqrt(1.519^2-sin(x)^2));
```

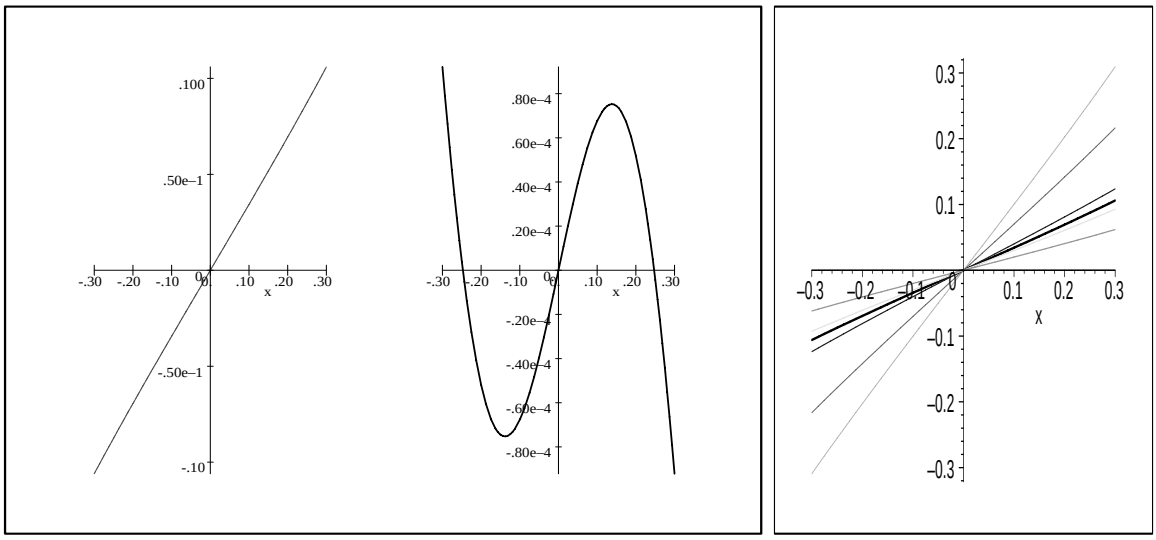
$$d := 1$$
$$fr := x \to d\,\sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{2.307361 - \sin(x)^2}}\Big)$$

```
> cs:=0.3425;           # cs=c/d, guter Kandidat
  fl :=x->cs*tan(x);
  fl1:=x->0;
  fl2:=x->0.2*tan(x);
  fl3:=x->0.3*tan(x);
  fl4:=x->0.4*tan(x);
  fl5:=x->0.7*tan(x);
  fl6:=x->tan(x);
```

$$cs := .3425$$
$$fl := x \to cs\ \tan(x)$$
$$fl1 := 0$$
$$fl2 := x \to 0.2\ \tan(x)$$
$$fl3 := x \to 0.3\ \tan(x)$$
$$fl4 := x \to 0.4\ \tan(x)$$
$$fl5 := x \to 0.7\ \tan(x)$$
$$fl6 := \tan$$

```
> # geringe Abweichung zwischen fl(x) und fr(x) im Intervall
  pl[1]:=plot([fl(x),fr(x)],x=Ie,thickness=2,linestyle=[SOLID,DASH]):
  pl[2]:=plot(fl(x)-fr(x),x=Ie,thickness=3,color=black):
  plots[display](pl);

> p1:=plot([fl1(x),fl2(x),fl3(x),fl4(x),fl5(x),fl6(x)],x=Ie,thickness=2):
  p2:=plot(fr(x),x=Ie,thickness=4,color=black):
  plots[display](p1,p2);
```



Vereinfachungen des Ausdrucks?

```
> cs:='cs':
  w:=cs*tan(x)-sin(x)*(1-cos(x)/sqrt(1.519^2-sin(x)^2));    # cs=c/d
  simplify(w);    # cos  ist Maple's Liebling
```

$$w := cs\ \tan(x) - \sin(x)\left(1 - \frac{\cos(x)}{\sqrt{2.307361 - \sin(x)^2}}\right)$$

$$\frac{\sin(x)(cs\sqrt{0.1307361\,10^7+0.1000000\,10^7\cos(x)^2}-1.\cos(x)\sqrt{0.1307361\,10^7+0.1000000\,10^7\cos(x)^2}+1000.\cos(x)^2)}{\cos(x)\sqrt{0.1307361\,10^7+0.1000000\,10^7\cos(x)^2}}$$

```
> w1:=subs(tan(x)=sin(x)/cos(x),subs(sin(x)^2=1-cos(x)^2,w));
```

$$w1 := \frac{cs\ \sin(x)}{\cos(x)} - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big)$$

Test eines guten Wertes cs, die maximalen Abweichungen von der x-Achse sind ungefaehr betragsgleich

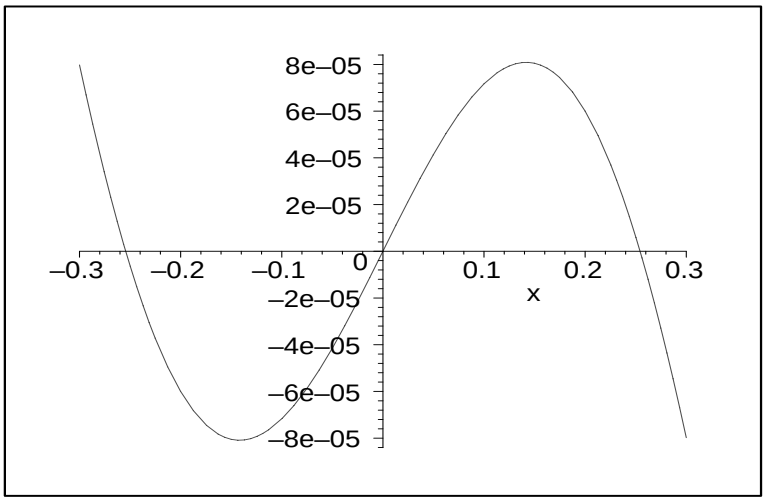
```
> cs:=0.34254;
  f:=x->cs*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  evalf(f(-eps));
  evalf(f(eps));
  pw1:=plot(f(x),x=Ie):
  display(pw1);
```

$$cs := .34254$$

$$f := x \to cs\ \tan(x) - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big)$$

$$0.0000797897$$

$$-0.0000797897$$




```
> diff(f(x),x):
  fs:=unapply(diff(f(x),x),x);
  x1:=fsolve(fs(x),x,x=-eps..0);
  x2:=fsolve(fs(x),x,x= 0..eps);
  evalf(f(x1));
  evalf(f(x2));
```

$$fs := x \to 0.34254 + 0.34254\ \tan(x)^2 - \cos(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big) - \sin(x)\Big(\frac{\sin(x)}{\sqrt{1.307361 + \cos(x)^2}} - \frac{\cos(x)^2\sin(x)}{(1.307361 + \cos(x)^2)^{(3/2)}}\Big)$$

$$x1 := \text{-}0.1417680295$$
$$x2 := 0.1417680295$$
$$\text{-}0.00008088674$$
$$0.00008088674$$

Vergleichswerte fuer *cs* (Grenzwerte und weiterer Wert)

```
> cu:=1-1/sqrt(2.307361);
  co:=evalf(cos(eps)*(1-cos(eps)/sqrt(1.307361+cos(eps)^2)));
  cw:=0.343;
```

$$cu := 0.3416721527$$
$$co := 0.3427979381$$
$$cw := 0.343$$

```
> p:=array(1..4,[]):

  fu:=x->cu*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  evalf(fu(-eps)), evalf(fu(eps));
  evalf(D(fu)(0));
  p[1]:=plot({fu(x),[[0,-0.00035],[0,0.00035]]},x=Ie,color=black):

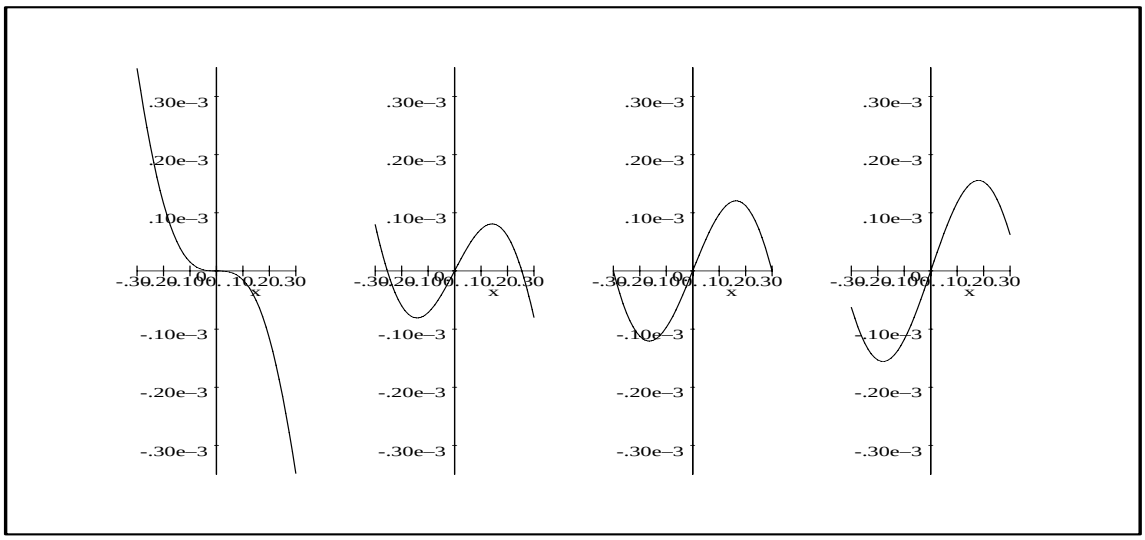
  p[2]:=plot({f(x),[[0,-0.00035],[0,0.00035]]},x=Ie,color=black):

  fo:=x->co*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  evalf(fo(-eps)), evalf(fo(eps));
  p[3]:=plot({fo(x),[[0,-0.00035],[0,0.00035]]},x=Ie,color=black):

  fw:=x->cw*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  evalf(fw(-eps)), evalf(fw(eps));
  p[4]:=plot({fw(x),[[0,-0.00035],[0,0.00035]]},x=Ie,color=black):

  plots[display](p);
```

$$fu := x \to cu\ \tan(x) - \sin(x)\left(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\right)$$
$$0.0003482463,\ \text{-}0.0003482463$$
$$fo := x \to co\ \tan(x) - \sin(x)\left(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\right)$$
$$0.$$
$$0.1\,10^{-9},\ \text{-}0.1\,10^{-9}$$
$$fw := x \to cw\ \tan(x) - \sin(x)\left(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\right)$$
$$\text{-}0.0000625050,\ 0.0000625050$$



Einschachteln von cs mittels Bisektion
$0 \leq cs \leq 1$

```
> a:=cu;
  b:=co;
  anz:=0:

  while b-a>1E-8 do
   cs:=(a+b)/2;
   f:=x->cs*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
   f11:=evalf(abs(f(eps)));
   fs1:=unapply(diff(f(x),x),x);
   x1:=fsolve(fs1(x),x,x=0..eps);
   f1x1:=evalf(f(x1));
   if f11>f1x1 then a:=cs else b:=cs end if;
   anz:=anz+1;
   printf(‘ cs= %.8e, |f(eps)|= %.8e\n‘,cs,f11);
   printf(‘ x1= %.8e, f(x1)   = %.8e, f’(x1)  = %+.8e\n\n‘,
          x1,f1x1,fs1(x1));
  end do:

  cs:=(a+b)/2:
  printf(‘ optimales cs = %.8e,  iter = %g\n‘,cs,anz);
```

$$a := .3416721527$$
$$b := .3427979381$$

```
cs= 3.42235045e-01, |f(eps)|= 1.74123200e-04
x1= 1.12336974e-01, f(x1)   = 4.18056900e-05, f’(x1)  = +4.70000000e-11

cs= 3.42516492e-01, |f(eps)|= 8.70616000e-05
x1= 1.39651451e-01, f(x1)   = 7.75568600e-05, f’(x1)  = +4.70000000e-11
```

```
cs= 3.42657215e-01, |f(eps)|= 4.35309000e-05
x1= 1.52055473e-01, f(x1)   = 9.82362700e-05, f'(x1)  = -8.00000000e-12

cs= 3.42586853e-01, |f(eps)|= 6.52963000e-05
x1= 1.45930400e-01, f(x1)   = 8.76736900e-05, f'(x1)  = -2.22000000e-10

cs= 3.42551672e-01, |f(eps)|= 7.61790000e-05
x1= 1.42811781e-01, f(x1)   = 8.25589400e-05, f'(x1)  = +1.00000000e-10

cs= 3.42534082e-01, |f(eps)|= 8.16203000e-05
x1= 1.41237045e-01, f(x1)   = 8.00437000e-05, f'(x1)  = -1.45000000e-10

cs= 3.42542877e-01, |f(eps)|= 7.88996000e-05
x1= 1.42025743e-01, f(x1)   = 8.12977800e-05, f'(x1)  = -1.12000000e-10

cs= 3.42538480e-01, |f(eps)|= 8.02600000e-05
x1= 1.41631730e-01, f(x1)   = 8.06698900e-05, f'(x1)  = +2.68000000e-10

cs= 3.42536281e-01, |f(eps)|= 8.09401000e-05
x1= 1.41434472e-01, f(x1)   = 8.03565700e-05, f'(x1)  = +2.08000000e-10

cs= 3.42537380e-01, |f(eps)|= 8.06001000e-05
x1= 1.41533122e-01, f(x1)   = 8.05131000e-05, f'(x1)  = -2.01000000e-10

cs= 3.42537930e-01, |f(eps)|= 8.04300000e-05
x1= 1.41582431e-01, f(x1)   = 8.05914700e-05, f'(x1)  = +1.30000000e-11

cs= 3.42537655e-01, |f(eps)|= 8.05150000e-05
x1= 1.41557783e-01, f(x1)   = 8.05522700e-05, f'(x1)  = -2.79000000e-10

cs= 3.42537518e-01, |f(eps)|= 8.05575000e-05
x1= 1.41545457e-01, f(x1)   = 8.05327300e-05, f'(x1)  = -4.00000000e-11

cs= 3.42537586e-01, |f(eps)|= 8.05363000e-05
x1= 1.41551620e-01, f(x1)   = 8.05425400e-05, f'(x1)  = +1.46000000e-10

cs= 3.42537552e-01, |f(eps)|= 8.05469000e-05
x1= 1.41548534e-01, f(x1)   = 8.05376500e-05, f'(x1)  = +2.35000000e-10

cs= 3.42537569e-01, |f(eps)|= 8.05416000e-05
x1= 1.41550077e-01, f(x1)   = 8.05401000e-05, f'(x1)  = -6.20000000e-11

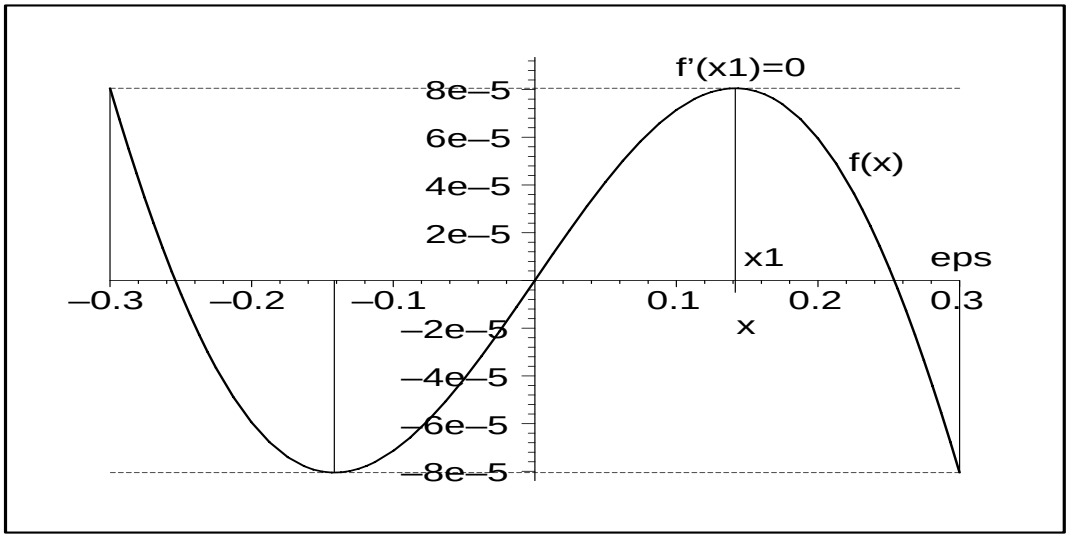
cs= 3.42537578e-01, |f(eps)|= 8.05389000e-05
x1= 1.41550848e-01, f(x1)   = 8.05413100e-05, f'(x1)  = -5.10000000e-11

optimales cs = 3.42537574e-01,  iter = 17
```

Optimale Situation

```
> 'f(x)'=f(x);
  pl1:=plot([f(x),[[eps,0],[eps,f(eps)]],[[-eps,0],[-eps,f(-eps)]],
                  [[x1,-0.000005],[x1,f(x1)]],[[-x1,0],[-x1,f(-x1)]]],
            x=Ie,thickness=[3,2,2,2,2],color=black):
  pl2:=plot({[[eps,f(eps)],[-eps,f(eps)]],[[eps,-f(eps)],[-eps,-f(eps)]]},
             linestyle=DASH,thickness=2,color=blue):
  pl3:=textplot([[0.16,0.00001,‘ x1‘],[0.30,0.00001,‘ eps‘],
                 [0.15,0.00009,‘f'(x1)=0‘],
                 [0.24,0.00005,‘ f(x)‘]]):
  plots[display](pl1,pl2,pl3);
```

$$\mathrm{f(x)}=0.3425375735\,\tan(x)-\sin(x)\Big(1-\frac{\cos(x)}{\sqrt{1.307361+\cos(x)^2}}\Big)$$



Test von Funktion nach Kuerzungen

```
> f(x)/tan(x);
  plot(f(x)/tan(x),x=Ie,title=‘f(x)/tan(x), -eps<=x<=eps‘);
```

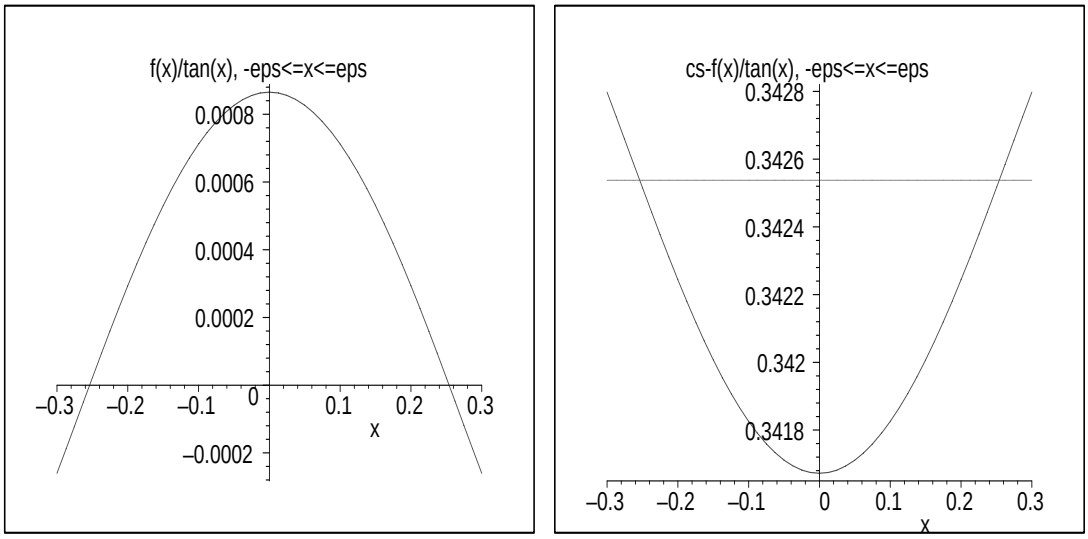
$$\frac{0.3425375735\,\tan(x)-\sin(x)\Big(1-\frac{\cos(x)}{\sqrt{1.307361+\cos(x)^2}}\Big)}{\tan(x)}$$

```
> fcs:=x->cs-cos(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  plot(fcs(x),x=Ie,title=‘f(x)/tan(x), -eps<=x<=eps‘); # Grafik wie vorher
```

$$fcs := x \to cs-\cos(x)\Big(1-\frac{\cos(x)}{\sqrt{1.307361+\cos(x)^2}}\Big)$$

```
> fcs1:=x->cos(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  plot([fcs1(x),cs],x=Ie,title=‘cs-f(x)/tan(x), -eps<=x<=eps‘);
         # Massstab beachten
```

$$fcs1 := x \to \cos(x)\left(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\right)$$



Parametervariante 2: $\varepsilon = 1$

```
> eps:=1;
  Ie:=-eps..eps;      # Intervall [-eps,eps]
```

$$eps := 1$$
$$Ie := \text{-1} \,..\, 1$$

Definition der Funktionen $f_r(x), f_l(x)$ und erster Vergleich

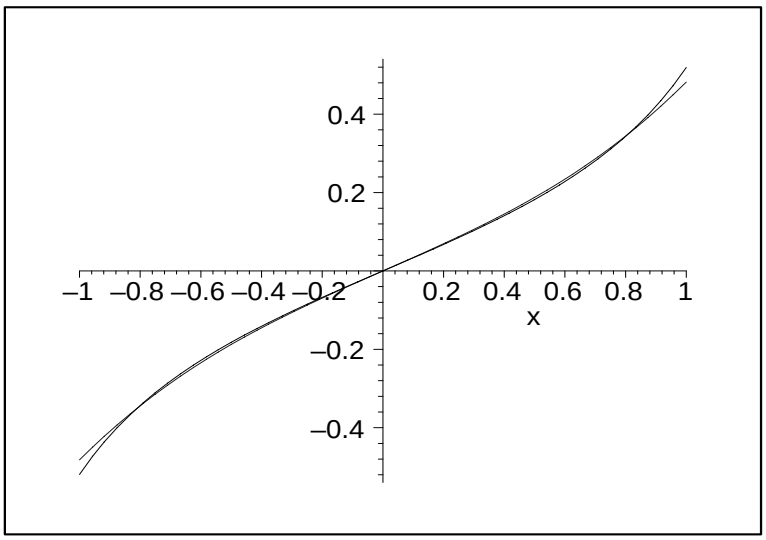
```
> d:=1;
  fr:=x->d*sin(x)*(1-cos(x)/sqrt(1.519^2-sin(x)^2));
```

$$d := 1$$
$$fr := x \to d\,\sin(x)\left(1 - \frac{\cos(x)}{\sqrt{2.307361 - \sin(x)^2}}\right)$$

```
> cs:=1/3;          # cs=c/d, guter Kandidat
  fl :=x->cs*tan(x);
```

$$cs := \frac{1}{3}$$
$$fl := x \to cs\tan(x)$$

```
> # Abweichung zwischen fl(x) und fr(x) im Intervall nahe den Raendern
  pl[1]:=plot([fl(x),fr(x)],x=Ie,color=[black,blue]):
```



Vereinfachungen des Ausdrucks?

```
> cs:='cs':
  w:=cs*tan(x)-sin(x)*(1-cos(x)/sqrt(1.519^2-sin(x)^2));    # cs=c/d
  simplify(w);
```

$$w := cs\ \tan(x) - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{2.307361 - \sin(x)^2}}\Big)$$

$$\frac{\sin(x)(cs\sqrt{0.1307361\,10^7+0.1000000\,10^7\cos(x)^2}-1.\cos(x)\sqrt{0.1307361\,10^7+0.1000000\,10^7\cos(x)^2}+1000.\cos(x)^2)}{\cos(x)\sqrt{0.1307361\,10^7+0.1000000\,10^7\cos(x)^2}}$$

```
> w1:=subs(tan(x)=sin(x)/cos(x),subs(sin(x)^2=1-cos(x)^2,w));
```

$$w1 := \frac{cs\ \sin(x)}{\cos(x)} - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big)$$

Test eines guten Wertes cs, die maximalen Abweichungen von der x-Achse sind ungefaehr betragsgleich

```
> cs:=0.32;
  ff:=x->cs*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  evalf(ff(-eps));
  evalf(ff(eps));

  pw1:=plot(ff(x),x=Ie):
  display(pw1);

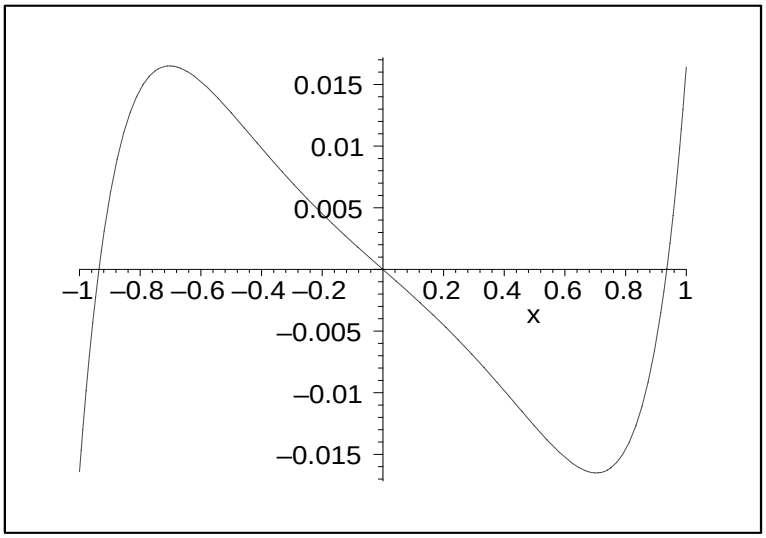
  diff(ff(x),x);
  ffs:=unapply(diff(ff(x),x),x);
  x1:=fsolve(ffs(x),x,x=-eps..0);
  x2:=fsolve(ffs(x),x,x= 0..eps);
  evalf(ff(x1));
  evalf(ff(x2));
```

$$cs := 0.32$$

$$ff := x \to cs\ \tan(x) - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big)$$

-0.0164109017

0.0164109017



$$ffs := x \to 0.32 + 0.32\ \tan(x)^2 - \cos(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big) - \sin(x)\Big(\frac{\sin(x)}{\sqrt{1.307361 + \cos(x)^2}} - \frac{\cos(x)^2 \sin(x)}{(1.307361 + \cos(x)^2)^{(3/2)}}\Big)$$

$x1$:= -0.7032606300

$x2$:= 0.7032606300

0.0165045363

-0.0165045363

Vergleichswerte fuer cs (Grenzwerte)

```
> co:=1-1/sqrt(2.307361);
  cu:=evalf(cos(eps)*(1-cos(eps)/sqrt(1.307361+cos(eps)^2)));
```

$$co := 0.3416721527$$

$$cu := 0.3094626813$$

```
> p:=array(1..3,[]):

  ffo:=x->co*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  evalf(ffo(-eps)), evalf(ffo(eps));
  evalf(D(ffo)(0));
  p[1]:=plot({ffo(x),[[0,-0.05],[0,0.05]]},x=Ie,color=black):
```

```
p[2]:=plot({ff(x),[[0,-0.05],[0,0.05]]},x=Ie,color=black):

ffu:=x->cu*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
evalf(ffu(-eps)), evalf(ffu(eps));
evalf(D(ffu)(0));
p[3]:=plot({ffu(x),[[0,-0.05],[0,0.05]]},x=Ie,color=black):

plots[display](p);
```

$$ffo := x \to co\,\tan(x) - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big)$$

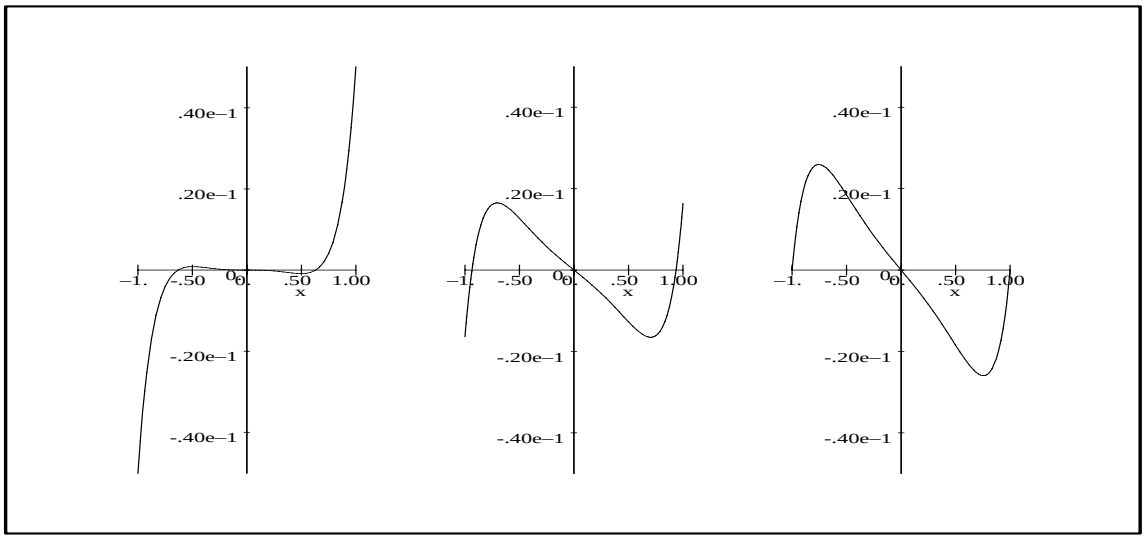
$$\text{-0.0501632797, } 0.0501632797$$

$$0.$$

$$ffu := x \to cu\,\tan(x) - \sin(x)\Big(1 - \frac{\cos(x)}{\sqrt{1.307361 + \cos(x)^2}}\Big)$$

$$\text{-0.2}\,10^{-9}\text{, } 0.2\,10^{-9}$$

$$\text{-0.0322094714}$$



Einschachteln von cs mittels Bisektion
$0 \le cs \le 1$

```
> a:=cu;
  b:=co;
  anz:=0:

  while b-a>1E-6 do
   cs:=(a+b)/2;
   ff:=x->cs*tan(x)-sin(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
   ff1:=evalf(abs(ff(eps)));
   ffs1:=unapply(diff(ff(x),x),x);
```

```
 x1:=fsolve(ffs1(x),x,x=-eps..-0.6);
 ffx1:=evalf(ff(x1));
 if ff1>ffx1 then b:=cs else a:=cs end if;
 anz:=anz+1;
 printf(‘ cs= %.8e, |ff(eps)|= %.8e\n‘,cs,ff1);
 printf(‘ x1= %.8e, ff(x1)  = %.8e\n\n‘,x1,ffx1);
end do:

cs:=(a+b)/2:
printf(‘ optimales cs = %.8e,  iter = %g\n‘,cs,anz);
```

$$a := 0.3094626813$$
$$b := 0.3416721527$$

```
cs=  3.25567417e-01, |ff(eps)|= 2.50816399e-02
x1= -6.69843990e-01, ff(x1)   = 1.19366951e-02

cs=  3.17515049e-01, |ff(eps)|= 1.25408200e-02
x1= -7.16588787e-01, ff(x1)   = 1.86404518e-02

cs=  3.21541233e-01, |ff(eps)|= 1.88112300e-02
x1= -6.94549576e-01, ff(x1)   = 1.52091615e-02

cs=  3.19528141e-01, |ff(eps)|= 1.56760252e-02
x1= -7.05856471e-01, ff(x1)   = 1.69056712e-02

cs=  3.20534687e-01, |ff(eps)|= 1.72436277e-02
x1= -7.00279799e-01, ff(x1)   = 1.60525512e-02

cs=  3.20031414e-01, |ff(eps)|= 1.64598264e-02
x1= -7.03086668e-01, ff(x1)   = 1.64779055e-02

cs=  3.20283051e-01, |ff(eps)|= 1.68517270e-02
x1= -7.01687946e-01, ff(x1)   = 1.62649258e-02

cs=  3.20157233e-01, |ff(eps)|= 1.66557768e-02
x1= -7.02388475e-01, ff(x1)   = 1.63713401e-02

cs=  3.20094323e-01, |ff(eps)|= 1.65578015e-02
x1= -7.02737863e-01, ff(x1)   = 1.64246041e-02

cs=  3.20062869e-01, |ff(eps)|= 1.65088139e-02
x1= -7.02912338e-01, ff(x1)   = 1.64512502e-02

cs=  3.20047142e-01, |ff(eps)|= 1.64843202e-02
x1= -7.02999521e-01, ff(x1)   = 1.64645766e-02
```

```
cs=  3.20039278e-01, |ff(eps)|= 1.64720734e-02
x1= -7.03043099e-01, ff(x1)   = 1.64712408e-02

cs=  3.20035346e-01, |ff(eps)|= 1.64659500e-02
x1= -7.03064884e-01, ff(x1)   = 1.64745730e-02

cs=  3.20037312e-01, |ff(eps)|= 1.64690117e-02
x1= -7.03053992e-01, ff(x1)   = 1.64729069e-02

cs=  3.20038295e-01, |ff(eps)|= 1.64705426e-02
x1= -7.03048545e-01, ff(x1)   = 1.64720738e-02

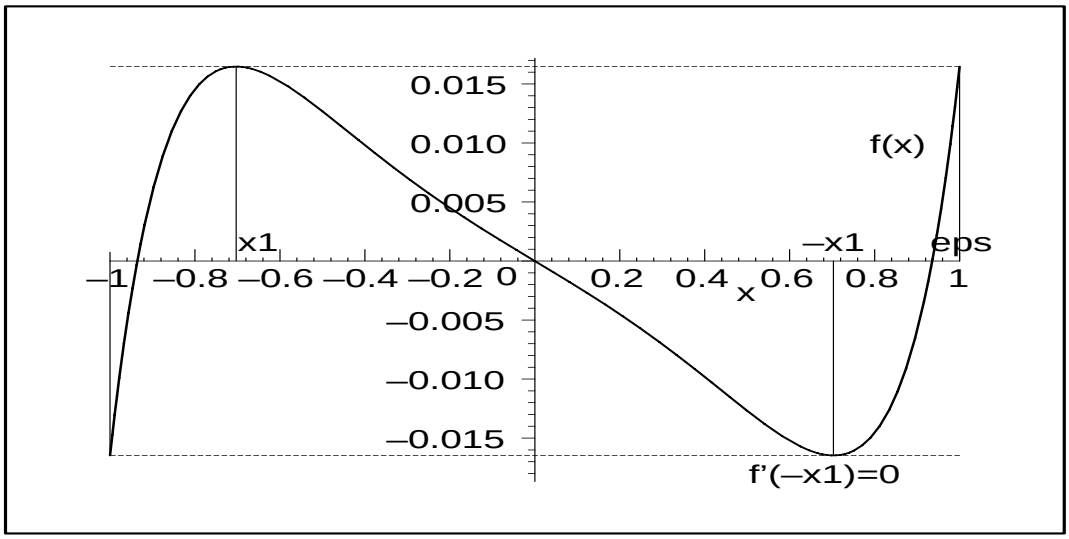
optimales cs = 3.20038787e-01,  iter = 15
```

Optimale Situation

```
> 'f(x)'=ff(x);
  pl1:=plot([ff(x),[[eps,0],[eps,ff(eps)]],[[-eps,0],[-eps,ff(-eps)]],
                  [[x1,0],[x1,ff(x1)]],[[-x1,0],[-x1,ff(-x1)]]],
            x=Ie,thickness=[3,2,2,2,2],color=black):
  pl2:=plot({[[eps,ff(eps)],[-eps,ff(eps)]],[[eps,-ff(eps)],
             [-eps,-ff(eps)]]},linestyle=DASH,thickness=2,color=blue):
  pl3:=textplot([[-0.67,0.0015,` x1`],[0.7,0.0015,` -x1`],[1,0.0015,` eps`],
                [0.7,-0.018,`f'(-x1)=0`],
                [0.85,0.01,` f(x)`]]):

  plots[display](pl1,pl2,pl3);
```

$$\mathrm{f(x)}{=}0.3200387865\,\tan(x)-\sin(x)\Big(1-\frac{\cos(x)}{\sqrt{1.307361+\cos(x)^2}}\Big)$$



Test von Funktion nach Kuerzungen

```
> ff(x)/tan(x);
  plot(ff(x)/tan(x),x=Ie,title=‘f(x)/tan(x), -eps<=x<=eps‘);
```

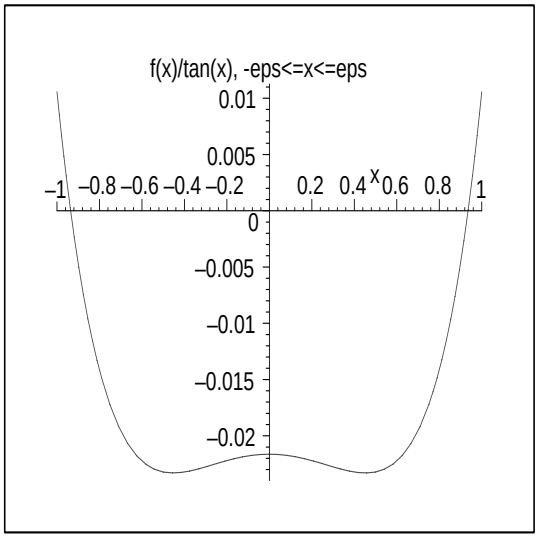
$$\frac{0.3200387865\,\tan(x)-\sin(x)\left(1-\dfrac{\cos(x)}{\sqrt{1.307361+\cos(x)^2}}\right)}{\tan(x)}$$

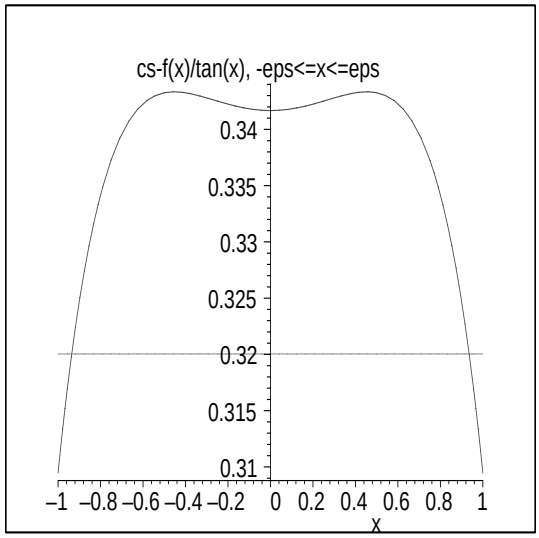
```
> ffcs:=x->cs-cos(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  plot(fcs(x),x=Ie,title=‘f(x)/tan(x), -eps<=x<=eps‘);
        # Grafik wie vorher
```

$$ffcs := x \to cs-\cos(x)\left(1-\frac{\cos(x)}{\sqrt{1.307361+\cos(x)^2}}\right)$$

```
> ffcs1:=x->cos(x)*(1-cos(x)/sqrt(1.307361+cos(x)^2));
  plot([ffcs1(x),cs],x=Ie,title=‘cs-f(x)/tan(x), -eps<=x<=eps‘);
           # Massstab beachten
```

$$ffcs1 := x \to \cos(x)\left(1-\frac{\cos(x)}{\sqrt{1.307361+\cos(x)^2}}\right)$$





2.13 Dateien und Arbeitsblätter

Zu den einzelnen Kapiteln gibt es Arbeitsblätter in den CAS, Programme und diverse Dateien.

1. Rechengenauigkeit, Auswertung von Ausdrücken, Funktionen und Kommandos
   calc1.mws
2. Lange Listen und Folgen
   list1.mws
3. Matrixgenerierung
   genarr1.mws
4. Felder und das Kommando `evalm`
   evalm_t1.mws
5. Produkte, Differenzen und Potenzen mit Matrizen und Vektoren
   matvek1.mws
6. Aufwand und Zeitmessungen beim Umgang mit Matrizen
   timearr1.mws, fpuspee1.pas, uhr.pas
7. *LU*-Faktorisierung
   lu1.mws, luf1.m, lu1.m
8. Orthogonalisierung und *QR*-Faktorisierung
   qrf1.mws, qrf1.m, gram_sch.m, givens_r.m, househ_r.m
9. Turbo Pascal → Maple
   a9tp1.mws, a9_k2.pas, a9_k3.pas, a9_f3.pas
10. Dateiarbeit in Maple
    file1.mws, werte1.m, werte1.txt, werte2.txt, werte3.txt, werte4.txt, werte1.dat
11. Maple → Matlab
    m2m1.mws, band_ul.m, aiv2.m, g.m, gs.m, usepolyfit.m, stem_.m, gendata.m
12. Beispiel
    vog1.mws

Kapitel 3

Was leistet der Computer?

An solchen Beispielen zeigt sich, was ein Computer schon und noch nicht leisten kann. Der wirklich kreative Anteil am Problemlösungsprozess bleibt beim Menschen. Der Phantasie sind aber keine Grenzen gesetzt. Natürlich ist es zumeist nicht einfach, komplizierte Sachverhalte methodisch geschickt und didaktisch wirksam für Präsentationen vor einem Hörerkreis aufzubereiten.
Der Komfort des Werkzeugs Computer einschließlich der Software kann sich aber nur “entfalten“ durch den an Ideen reichen und kreativen Nutzer in der Einheit mit seiner gezielten Nutzung, gründlichen Analyse und ständigen Weiterentwicklung.

Literaturverzeichnis

Maple

[1] Blachman, N. und Mossinghoff M. J.: *Maple griffbereit*. Vieweg Verlag Braunschweig 1996.

[2] Heal, K. M. et al.: *Handbuch Waterloo Maple*. Maple V - Learning Guide, Springer-Verlag New York 1998.

[3] Heck, A.: *Introduction to Maple*. 2nd Ed. Springer-Verlag New York 1996.

[4] Krawietz, A.: *Maple V für das Ingenieurstudium*. Springer-Verlag New York 1997.

[5] Klimek, G. und M. Klimek: *Discovering Curves and Surfaces with Maple*. Springer-Verlag New York 1997.

[6] Kofler, M.: *Maple V Release 4*. Addison Wesley Bonn 1996.

[7] Monagan, M.: *Programming in Maple: The Basics*. Institut für Wissenschaftliches Rechnen ETH-Zentrum, CH-8092 Zürich.

[8] Monagan, M. et al.: *Maple V Programming Guide*. Springer-Verlag New York 1996.

[9] Neundorf, W.: *Programming in Maple V Release 5*. Extended Basics. Preprint M 07/99 IfMath der TU Ilmenau, Februar 1999.

[10] Neundorf, W. und B. Walther: *Grafik, Animation und Ausgabeformate in Maple V Release 5*. Preprint M 12/00 IfMath der TU Ilmenau, Juni 2000.

[11] Nicolaides, R. A. und N. J. Walkington: *Maple: A Comprehensive Introduction*. Cambridge University Press 1996.

[12] Walz, A.: *Maple V. Rechnen und Programmieren mit Release 4*. R. Oldenbourg Verlag München 1998.

[13] Werner, W.: *Mathematik lernen mit Maple*. Bd. 1,2. Ein Lehr-und Arbeitsbuch für das Grundstudium. dpunkt Heidelberg 1996, 1998 (CD).

[14] Westermann, T.: *Mathematik für Ingenieure mit Maple*. Springer-Verlag 1996.

[15] Neundorf, W.: *Lösungsmethoden mit Maple. Betrachtung von Fehlern und Kondition sowie Darstellungsmöglichkeiten*. Preprint M 08/03 IfMath der TU Ilmenau, April 2003.

Matlab

[16] Gramlich, G. und W. Werner: *Numerische Mathematik mit Matlab*. 1. Auflage. dpunkt.verlag Heidelberg 2000.

[17] Neundorf, W.: *MATLAB - Teil I: - Vektoren, Matrizen, lineare Gleichungssysteme*. Preprint M 20/99 IfMath der TU Ilmenau, Juli 1999.

[18] Neundorf, W.: *MATLAB - Teil II: - Speicheraspekte, spezielle LGS, SDV, EWP, Graphik, NLG, NLGS*. Preprint M 23/99 TUI, September 1999.

[19] Neundorf, W.: *MATLAB - Teil III: - Komplexe LGS, Interpolation, Splines*. Preprint M 10/00 IfMath der TU Ilmenau, Mai 2000.

[20] Neundorf, W.: *MATLAB - Teil IV: - Approximation, Numerische Intergration*. Preprint M 11/00 IfMath der TU Ilmenau, Mai 2000.

Numerik

[21] Deuflhard, P. und H. Hohmann: *Numerische Mathematik*. 1: Eine algorithmisch orientierte Einführung. 3. überarbeitete und erweiterte Auflage, Lehrbuch. Walter de Gruyter Berlin 2002.

[22] Hanke-Bourgeois, M.: *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*. Mathematische Leitfäden. B. G. Teubner GmbH, Stuttgart 2002.

[23] Hermann, M.: *Numerische Mathematik*. R. Oldenbourg Verlag München 2001.

[24] Stoer, J. und R. Bulirsch: *Numerical mathematics 2*. An Introduction - under consideration of lectures by F. L. Bauer. 4. neu bearbeitete und erweiterte Auflage. Springer-Verlag Berlin 2000.

[25] Kiełbasiński, A. und H. Schwetlick: *Numerische lineare Algebra*. Mathematik für Naturwissenschaft und Technik Band 18, DVW, Berlin 1988.

[26] Hackbusch, W.: *Iterative Lösung großer schwach besetzter Gleichungssysteme*. Leitfäden der angewandten Mathematik und Mechanik Band 69. B. G. Teubner Stuttgart 1991.

[27] Maess, G.: *Vorlesungen über numerische Mathematik*. Band 1, 2. Akademie-Verlag Berlin 1984, 1988.

[28] Meister, A.: *Numerik linearer Gleichungssysteme*. Eine Einführung in moderne Verfahren. Friedr. Vieweg & Sohn VG mbH, Braunschweig 1999.

[29] Schwarz, H. R.: *Numerische Mathematik*. B. G. Teubner Stuttgart 1988.

[30] Schaback, R. und H. Werner: *Numerische Mathematik*. Springer-Verlag, Berlin 1992.

[31] Süli, E. und D. Mayers *An Introduction to Numerical Analysis*. Cambridge Textbooks. Cambridge University Press 2003.

[32] Neundorf, W.: *Numerische Mathematik*. Vorlesungen, Übungen, Algorithmen und Programme. Berichte aus der Mathematik. Shaker Verlag Aachen 2002.

[33] Neundorf, W.: *Wissenschaftliches Rechnen - Matrizen und lineare Gleichungssysteme*. Vorlesungsskript IfMath der TU Ilmenau, August 2002.

[34] Neundorf, W.: *Kondition eines Problems und angepaßte Lösungsmethoden*. Preprint M 09/95 IfMath der TU Ilmenau, April 1995.

Anschrift:

Dr. rer. nat. habil. Werner Neundorf
Technische Universität Ilmenau, Institut für Mathematik
PF 10 05 65
D - 98684 Ilmenau

E-mail: werner.neundorf@tu-ilmenau.de
Homepage: http://www.mathematik.tu-ilmenau.de/∼neundorf/index_de.html