Kapitel 7

Eigenwerte und Eigenvektoren reeller Matrizen

7.1 Einleitung: allgemeine und reguläre Eigenwertprobleme.

Im Kapitel 2 standen lineare *inhomogene* Probleme der Art

$$A \cdot \mathbf{x} = \mathbf{f} \qquad A\text{: reelle Matrix}$$

im Mittelpunkt. Im folgenden geht es um *homogene* Probleme, d.h. um solche, bei denen der Vektor $\mathbf{f}$ den *Nullvektor* darstellt:

$$A \cdot \mathbf{x} = 0 \tag{7.1}$$

Bei der Lösung eines solchen Problems ist laut Theorie zwischen zwei Fällen zu unterscheiden:

- Die Determinante der Koeffizientenmatrix A ist von Null verschieden: Dann ist der Lösungsvektor $\mathbf{x}$ einfach der Nullvektor.
- Die Determinante von A ist Null: Unter diesen Umständen gibt es neben der *trivialen* Lösung $\mathbf{x} = 0$ auch noch eine *nicht-triviale* Lösung $\mathbf{x} \neq 0$. So hat z.B. das homogene System

$$\begin{pmatrix} 1 & 2 & 3 \\ -1 & 13 & 2 \\ 0 & 3 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 0$$

  die nicht-triviale Lösung $\mathbf{x} = (-7, -1, +3)$. Es ist evident, daß dieser Lösungsvektor nur bis auf eine willkürliche Konstante bestimmt ist: mit $\mathbf{x}$ ist natürlich auch jeder Vektor $c \cdot \mathbf{x}$ Lösung des obigen Systems.

Es sind aber nicht Systeme vom Typus (7.1), um die es im folgenden geht. Viel wichtiger für die praktische Anwendung sind nämlich homogene Systeme der Art

$$A(\lambda) \cdot \mathbf{x} = 0 \qquad \text{mit} \qquad A(\lambda) = [a_{ij}(\lambda)] \quad , \tag{7.2}$$

bei denen die Matrixelemente von einer Variablen λ abhängen.
Die Lösungen von (7.2) sind:

- die triviale Lösung $\mathbf{x} = 0$.
- nicht-triviale Lösungen $\mathbf{x}_1, \mathbf{x}_2, \ldots$ immer dann, wenn λ einen der Werte $\lambda_1, \lambda_2, \ldots$ annimmt, die zu einer *Singularität* der Matrix A führen.

Die *Bedingungsgleichung für nicht-triviale Lösungen* lautet demnach:

$$det[A(\lambda)] = 0 \tag{7.3}$$

Die (i.a. komplexwertigen) Lösungen der Gleichung (7.3), $\lambda_1, \lambda_2, \ldots$ nennt man die *Eigenwerte*, und die entsprechenden Lösungsvektoren $\mathbf{x}_1, \mathbf{x}_2, \ldots$ nennt man die *Eigenvektoren* des Systems (7.2).
Die Ermittlung der Eigenwerte und -vektoren von (7.2) heißt Lösung des *allgemeinen Eigenwertproblems*. Für solche Probleme bleibt zumeist nur die aufwendige Methode, die Nullstellen der Gleichung (7.3) numerisch zu ermitteln, was die Berechnung von sehr vielen Determinanten erfordert und dementsprechend lange Rechenzeiten erfordert.

Ein sehr wichtiger Sonderfall des allgemeinen Eigenwertproblems (7.2) ist das *reguläre Eigenwertproblem* mit

$$A(\lambda) = A_0 - \lambda \cdot E \qquad \text{E ... Einheitsmatrix} \quad , \tag{7.4}$$

wobei die Matrix A_0 nicht von λ abhängt! Für derartige Probleme stehen effizientere numerische Methoden zur Verfügung.

Für reguläre Eigenwertprobleme

$$(A_0 - \lambda E)\mathbf{x} = 0 \qquad \text{bzw.} \qquad A_0\,\mathbf{x} = \lambda\,\mathbf{x} \tag{7.5}$$

lautet die Bedingungsgleichung für nicht-triviale Lösungen offenbar

$$det(A_0 - \lambda E) = 0 \quad . \tag{7.6}$$

Da im folgenden ausschließlich von regulären Eigenwertproblemen die Rede sein wird, wird ab nun auf die Indizierung von A_0 verzichtet.

Die Auswertung von (7.6) führt zu einem Polynom n-ten Grades von λ, wenn n die Ordnung des linearen Systems (7.5) ist. Man nennt dieses Polynom *das charakteristische Polynom der Matrix A*:

$$det(A - \lambda E) \equiv P_n(\lambda) = \lambda^n + \sum_{i=1}^{n} p_i \lambda^{n-i} \tag{7.7}$$

Die $p_1, \ldots, p_n$ sind die (reellen) Koeffizienten des charakteristischen Polynoms. Dieses Polynom hat genau n (nicht notwendig verschiedene) Nullstellen, die entweder reell sind oder konjugiert-komplexe Wertepaare darstellen. Diese Nullstellen sind wegen (7.6) die n Eigenwerte der Matrix A. Dabei kann es ohne weiteres vorkommen, daß das charakteristische Polynom *Mehrfach-Nullstellen* hat, daß also gilt:

$$P_n(\lambda) = 0 \qquad \text{für} \qquad \lambda : \lambda_1, \lambda_2, \ldots, \lambda_k = \lambda_{k+1}, \ldots = \lambda_{k+q-1}, \lambda_{k+q}, \ldots, \lambda_n \quad .$$

Man nennt den Eigenwert λ_k *q-fach entartet*.

Zu jedem Eigenwert λ_i gehört nun gemäß (7.5) ein Eigenvektor $\mathbf{x}_i$.

7.1.1 Eigenwerte und Eigenvektoren wichtiger Matrixformen

Wie bei den *inhomogenen* linearen Gleichungssystemen spielt auch bei den Eigenwertproblemen die konkrete Form der Matrix A eine wichtige Rolle. Im folgenden einige Fakten zu diesem Thema:

- Eine *reelle* Matrix A heißt *symmetrisch*, wenn gilt:

  $$A = A^T\,,$$

  wobei A^T die zu A transponierte Matrix mit den Komponenten

  $$(a_{i,j})^T = a_{j,i}$$

  darstellt.

- Eine *komplexwertige* Matrix A heißt *hermitesch*, wenn gilt:

  $$A = A^\dagger\,,$$

  wobei $A^\dagger$ die zu A hermitesch konjugierte Matrix mit den Komponenten

  $$(a_{i,j})^\dagger = a^*_{j,i}$$

  darstellt.

Es gilt:
Alle Eigenwerte einer symmetrischen oder hermiteschen Matrix sind reell.

- Eine reelle Matrix A heißt *orthogonal*, wenn gilt:

  $$A\,A^T = E \qquad \text{d. h.} \qquad A^T = A^{-1}\,,$$

  wobei E die Einheitsmatrix darstellt.

- Eine komplexwertige Matrix A heißt *unitär*, wenn gilt:

  $$A\,A^\dagger = E \qquad \text{d. h.} \qquad A^\dagger = A^{-1}\,.$$

- Eine reelle bzw. komplexwertige Matrix wird *normal* genannt, wenn sie mit der zu ihr transponierten bzw. hermitesch konjugierten Matrix vertauscht:

  $$A\,A^T = A^T\,A \qquad \text{bzw.} \qquad A\,A^\dagger = A^\dagger\,A\,.$$

  Es ist evident, daß sowohl symmetrische als auch hermitesche Matrizen zur Klasse der normalen Matrizen gehören.

Eine sehr wichtige Frage ist es auch, ob eine Matrix *diagonalisierbar* ist oder nicht.
Diagonalisierbar heißt, daß eine nicht-singuläre Matrix U existiert, welche die Matrix A in eine Diagonalmatrix D überführt:

$$U^{-1}\,A\,U = D\,. \tag{7.8}$$

Eine derartige Transformation ist eine *Ähnlichkeitsoperation*, bei der sich das Eigenwertspektrum der Matrix nicht ändert; es gilt also:

$$\text{Eigenwerte von } D = \text{Eigenwerte von } A\,.$$

- Da das Eigenwertproblem von D einfach zu lösen ist (die Eigenwerte sind die Diagonalelemente von D), ist eine Diagonalisierung gemäß (7.8) äquivalent mit der Bestimmung der Eigenwerte von A:

$$\lambda_i = d_{ii}\,. \tag{7.9}$$

- Man kann leicht zeigen, daß die Spaltenvektoren von U gleichzeitig die Eigenvektoren von A sind:

  Multipliziert man nämlich die Tranformationsvorschrift (7.8) von links mit U, so ergibt sich

$$U \cdot U^{-1}AU = U \cdot D \qquad \text{und}$$

$$A \cdot U = U \cdot D \quad . \tag{7.10}$$

  Nun kann aber die Matrix U auch *als ein System von n Spaltenvektoren betrachtet werden*:

$$U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{21} & u_{22} & \cdots & u_{2n} \\ . & . & & . \\ . & . & & . \\ u_{n1} & u_{n2} & \cdots & u_{nn} \end{pmatrix} \equiv (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n) \quad .$$

  Die komponentenweise Berechnung der rechten Seite von (7.10) ergibt:

$$(UD)_{ij} = \sum_{l=1}^{n} u_{il} d_{lj} = u_{ij} d_{jj} = \lambda_j u_{ij} \quad .$$

  Auch die Matrix UD kann unter Berücksichtigung von (7.9) als Vektoren-System geschrieben werden:

$$UD \equiv (\lambda_1 \mathbf{u}_1, \lambda_2 \mathbf{u}_2, \ldots, \lambda_n \mathbf{u}_n) \quad .$$

  Somit hat (7.10) die Form

$$A \cdot (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n) = (\lambda_1 \mathbf{u}_1, \lambda_2 \mathbf{u}_2, \ldots, \lambda_n \mathbf{u}_n) \quad .$$

  Damit ist aber für jeden Spaltenvektor $\mathbf{u}_j$ die Eigenwertbedingung erfüllt, und man kann sagen:

  Die j-te Spalte der Transformationsmatrix U stellt den j-ten Eigenvektor der gegebenen Matrix A dar.

Zusammenfassung: Wenn es gelingt, die Transformationsmatrix U zu finden, ist das vollständige Eigenwertproblem für die Matrix A gelöst!

Welche Matrizen sind diagonalisierbar?

- Alle Matrizen, welche ein *vollständiges, linear unabhängiges* Eigenvektor-System haben.
- Insbesondere kann gezeigt werden, daß alle *normalen* Matrizen ein solches Eigenvektor-System haben; darüber hinaus sind die Eigenvektoren solcher Matrizen auch noch *orthonormal*, d.h. sie gehorchen der Beziehung

  $$\mathbf{x}_i \cdot \mathbf{x}_j^* = \delta_{i,j} \,.$$

  Da diese Eigenvektoren die Spaltenvektoren der Transformationsmatrix U sind, haben diese Matrizen im Falle (reeller bzw. komplexwertiger) Matrizen A die Eigenschaft

  $$U\,U^T = E \qquad \text{bzw.} \qquad U\,U^\dagger = E \,,$$

  d.h. sie sind *orthogonal* bzw. *unitär*.

Für normale Matrizen gilt also die wichtige Aussage:
Normale Matrizen können mittels orthogonaler bzw. unitärer Transformationen auf Diagonalform gebracht werden:

$$U^T\,A\,U = D \qquad \text{bzw.} \qquad U^\dagger\,A\,U = D \,. \tag{7.11}$$

7.2 Numerische Behandlung regulärer Eigenwertprobleme.

7.2.1 Allgemeines

Die numerische Behandlung von regulären Eigenwertproblemen stellt ein außerordentlich wichtiges und deshalb sehr gut entwickeltes Gebiet der Numerischen Mathematik dar. Typisch für dieses Gebiet ist die starke Spezialisierung der einzelnen Methoden. Der Benutzer muß also genau wissen, was er will, um das für sein Problem richtige Verfahren auszuwählen.

Einen kleinen Überblick über das vielfältige Software-Angebot bzgl. der Lösung von Eigenwertproblemen s. in Abschnitt 7.6.

Im folgenden soll versucht werden, Ihnen an Hand einiger einfacher und dennoch sehr effektiver numerischer Methoden die Grundideen der wichtigsten Verfahren klarzumachen. Eine auch nur annähernd vollständige Beschreibung der in der Literatur enthaltenen Methoden ist im Rahmen dieser LV. unmöglich und wird daher nicht angestrebt.

Im folgenden beschränken wir uns auf *reelle* Matrizen und beschreiben

- das Verfahren von v. Mises (Verfahren der Vektoriteration);

- das Jacobi-Verfahren;
- die Anwendung des Hyman-Verfahrens auf Hessenberg-Matrizen.

7.3 Das Verfahren von v. Mises.

Dieses iterative Verfahren liefert den betragsgrößten (bzw., wie später gezeigt wird, auch den betragskleinsten) Eigenwert einer reellen Matrix A.
Voraussetzung für die Anwendbarkeit dieses Verfahrens ist, daß die gegebene Matrix diagonalisierbar ist (d.h., ein linear unabhängiges Eigenvektorsystem hat), und daß ein betragsmäßig dominanter Eigenwert existiert:

$$| \lambda_1 | > | \lambda_2| \geq | \lambda_3 | \geq \ldots \geq | \lambda_{n-1} | > | \lambda_n | \ . \tag{7.12}$$

In Fall linear unabhängiger Eigenvektoren kann jeder Vektor $\mathbf{v}^{(0)}$ (mit Ausnahme des Nullvektors) als Linearkombination der Eigenvektoren dargestellt werden:

$$\mathbf{v}^{(0)} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i \qquad \text{mit} \qquad | \alpha_1 | + | \alpha_2 | + \ldots + | \alpha_n | \neq 0 \quad .$$

Der beliebige [1] Vektor $\mathbf{v}^{(0)}$ ist der Anfangsvektor für die folgende Iteration:
$\mathbf{v}^{(0)}$ wird von links mit der Matrix A multipliziert:

$$A \cdot \mathbf{v}^{(0)} \equiv \mathbf{v}^{(1)} = \sum_{i=1}^{n} \alpha_i A \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i \lambda_i \mathbf{x}_i \quad ,$$

wobei die letzte Identität aus der Eigenwertgleichung (7.5) folgt. Nun wird der Vektor $\mathbf{v}^{(1)}$ seinerseits von links mit der Matrix A multipliziert (*fortgesetzte Vektormultiplikation*):

$$A \cdot \mathbf{v}^{(1)} \equiv \mathbf{v}^{(2)} = \sum_{i=1}^{n} \alpha_i \lambda_i A \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i \lambda_i^2 \mathbf{x}_i \quad .$$

Durch die ständige Wiederholung dieser Rechenschrittes erhält man eine Vektorfolge $\mathbf{v}^{(0)}$, $\mathbf{v}^{(1)}$, ..., $\mathbf{v}^{(t)}$, $\mathbf{v}^{(t+1)}$, ... mit

$$\mathbf{v}^{(t)} = \sum_{i=1}^{n} \alpha_i \lambda_i^t \mathbf{x}_i \qquad (t = 0, 1, \ldots) \quad . \tag{7.13}$$

Nun bedeutet jedoch die Gültigkeit von (7.12), daß mit Fortschreiten der Iteration der erste Summenterm in (7.13) wegen der immer höheren Potenzen von λ immer stärker dominiert. Für nicht zu kleine t-Werte gilt also in guter Näherung

$$\mathbf{v}^{(t)} \approx \alpha_1 \lambda_1^t \mathbf{x}_1 \qquad \text{und} \qquad \mathbf{v}^{(t+1)} \approx \alpha_1 \lambda_1^{t+1} \mathbf{x}_1 \quad . \tag{7.14}$$

Die obigen Gleichungen sind Vektorgleichungen, d.h. sie müssen für alle Komponenten gelten, so z.B. auch für die l-te Komponente. Es gilt demnach

$$\frac{v_l^{(t+1)}}{v_l^{(t)}} \approx \lambda_1 \qquad \text{für alle} \qquad l = 1, 2, \ldots, n \quad . \tag{7.15}$$

[1] vgl. aber Abschnitt 7.3.5.

D.h.: Im Falle der Konvergenz strebt das Verhältnis der gleichen Komponenten aufeinanderfolgender Vektoren $\mathbf{v}^{(t)}$ dem betragsgrößten Eigenwert zu:

$$\lim_{t\to\infty} \frac{v_l^{(t+1)}}{v_l^{(t)}} \to \lambda_1 \quad . \tag{7.16}$$

Ebenso ergibt sich unmittelbar aus (7.14), daß gilt:

$$\lim_{t\to\infty} \mathbf{v}^{(t)} \to \quad \text{prop.} \quad \mathbf{x}_1 \quad . \tag{7.17}$$

Im Prinzip könnte man ein beliebiges l auswählen und die Gleichung (7.15) auswerten. Nun ist jedoch nicht auszuschließen, daß während der Iteration ein $v_l^{(t)}$ Null bzw. sehr klein wird. Um die dadurch auftretenden Probleme zu vermeiden, berechnet man anstelle von (7.15) den Ausdruck

$$\frac{1}{n'} \sum_{\mu} \frac{v_\mu^{(t+1)}}{v_\mu^{(t)}} \approx \lambda_1 \quad , \tag{7.18}$$

wobei über alle Indizes μ summiert wird, für die gilt:

$$|v_\mu^{(t)}| > \epsilon \, . \tag{7.19}$$

n' ist die Anzahl der Summanden, die diese Bedingung erfüllen.

7.3.1 Die Berechnung des betragskleinsten Eigenwertes

einer reellen Matrix ist für viele praktische Anwendungen von größerem Interesse als die Kenntnis des betragsgrößten Eigenwertes. Das Verfahren von v. Mises ist in der Lage, auch dieses Problem zu lösen.

Ausgehend von der Eigenwertgleichung (7.5)

$$A \cdot \mathbf{x}_i = \lambda_i \cdot \mathbf{x}_i$$

erhält man durch Multiplikation mit der zu A inversen Matrix A^{-1}:

$$\mathbf{x}_i = \lambda_i \cdot A^{-1}\mathbf{x}_i \quad .$$

Die *Eigenwertgleichung für die inverse Matrix* lautet demnach einfach

$$A^{-1} \cdot \mathbf{x}_i = \frac{1}{\lambda_i} \cdot \mathbf{x}_i \quad . \tag{7.20}$$

Das heißt:

- Die Eigenwerte einer Matrix A^{-1} sind einfach die Kehrwerte der Eigenwerte von A.
- Die Matrizen A^{-1} und A haben dasselbe Eigenvektor-System.

Nachdem aber die Kehrwerte der betragskleinsten Eigenwerte von A natürlich die betragsgrößten Eigenwerte von A^{-1} sind, braucht man nur das v. Mises-Verfahren auf die Inverse der gegebenen Matrix anzuwenden, um den betragskleinsten Eigenwert von A zu erhalten.

Es gibt aber noch eine etwas andere Methode, die Mises-Iteration für den betragskleinsten Eigenwert durchzuführen. Zu diesem Zweck braucht man nur die Iterationsvorschrift für die *Inverse* von A umzuformen:

$$\mathbf{v}^{(t+1)} = A^{-1}\,\mathbf{v}^{(t)}\,.$$

Multipliziert man diese Gleichung von links mit A, so erhält man

$$A\,\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)}\,. \tag{7.21}$$

Dies kann aber als lineares Gleichungssystem mit der Koeffizientenmatrix A, dem inhomogenen Vektor $\mathbf{v}^{(t)}$ und dem Lösungsvektor $\mathbf{v}^{(t+1)}$ interpretiert werden. Wenn man zur Lösung dieses Systems die in Kapitel 2 beschriebene LU-Decomposition verwendet, hat man den Vorteil, daß man nur einmal (vor Beginn der Mises-Iteration) die zu A gehörende LU-Matrix berechnen muß, und danach während jedes Iterationsschrittes nur das Programm LUBKSB anwenden muß.
Auf diese Weise erhält man eine Serie von Vektoren mit der Eigenschaft

$$\lim_{t\to\infty} \frac{v_l^{(t)}}{v_l^{(t+1)}} = \lambda_n\,, \tag{7.22}$$

wobei λ_n der betragskleinste Eigenvektor von A ist.

7.3.2 Konvergenzsteigerung durch Spektralverschiebung

Auch ohne mathematische Beweisführung leuchtet ohne weiteres ein, daß die Konvergenz der Iteration (7.22) umso besser sein wird, je größer das Verhältnis

$$\frac{1}{\lambda_n} \Big/ \frac{1}{\lambda_{n-1}} = \frac{\lambda_{n-1}}{\lambda_n}$$

ist. Auf diesem Faktum beruht die folgende Idee:

Angenommen, man kennt mit λ_0 einen brauchbaren Schätzwert für den betragskleinsten Eigenwert λ_n. Nimmt man nun anstelle der gegebenen Matrix A die Matrix

$$A' = A - \lambda_0\,E\,,$$

so verschieben sich alle Eigenwerte um den Wert λ_0, also auch die Eigenwerte λ_{n-1} und λ_n, wobei gilt:

$$\frac{\lambda_{n-1} - \lambda_0}{\lambda_n - \lambda_0} > \frac{\lambda_{n-1}}{\lambda_n}\,.$$

Dementsprechend ist zu erwarten, daß der Grenzwert

$$\lim_{t\to\infty} \frac{v_l^{(t)}}{v_l^{(t+1)}} + \lambda_0 = \lambda_n \tag{7.23}$$

bedeutend rascher approximiert wird als der Grenzwert (7.22).

Die Überlegungen der Abschnitte 7.3.1 und 7.3.2 werden im folgenden Programm MISES realisiert.

7.3.3 Das Programm MISES

Quelle: [2], S.94ff, Programm S.331f mit Änderungen.

Das Programm MISES berechnet den betragskleinsten Eigenwert und den zugehörigen Eigenvektor einer reellen Matrix (inklusive Spektralverschiebung).

INPUT-Parameter:

A(,): reelle Matrix, deren betragskleinster Eigenwert zu bestimmen ist.

N: Ordnung der Matrix.

TMAX: maximale Anzahl von Iterationsschritten.

GEN: relative Genauigkeit für den Eigenwert.

V( ): Startvektor $\neq$ Nullvektor.

EIGW0: Schätzwert für den betragskleinsten Eigenwert.

OUTPUT-Parameter:

EIGW: Näherung für den betragskleinsten Eigenwert.

V( ): auf den Betrag 1 normierter, zu λ_1 gehörender Eigenvektor.

FEHLER: logische Fehlervariable: TRUE wenn keine Konvergenz innerhalb von TMAX ereicht wurde, sonst FALSE.

INTERNE Parameter:

W: Hilfsvektor.

EPS: Konstante s. Glg. (7.19).

VERWENDETE PROZEDUREN:

Das Programm MISES braucht die Prozeduren NORM sowie die Prozeduren LUDCMP und LUBKSB (s. Kapitel 2).

Struktogramm 21 — MISES(A,N,TMAX,GEN,V,EIGW0,EIGW,FEHLER)

EPS:=1.0E-6
FEHLER:=TRUE
EIGALT:=0.0

I=1(1)N

A(I,I):=A(I,I) - EIGW0

LUDCMP(A,N,INDX,D,KHAD)

T:=0

NORM(V,N)

LUBKSB(A,N,INDX,V,W)

SUM:=0.0
N1:=0

I=1(1)N

Y | W(I)| > EPS N

SUM:=SUM + V(I)/W(I)
N1:=N1 + 1

.....

EIGW:=SUM/N1

EIGW:=EIGW + EIGW0

Y |(EIGW-EIGALT)/EIGW| < GEN N

FEHLER:=FALSE

I=1(1)N

V(I):=W(I)

EIGALT:=EIGW

T:=T+1

T ≥ TMAX .or. notFEHLER

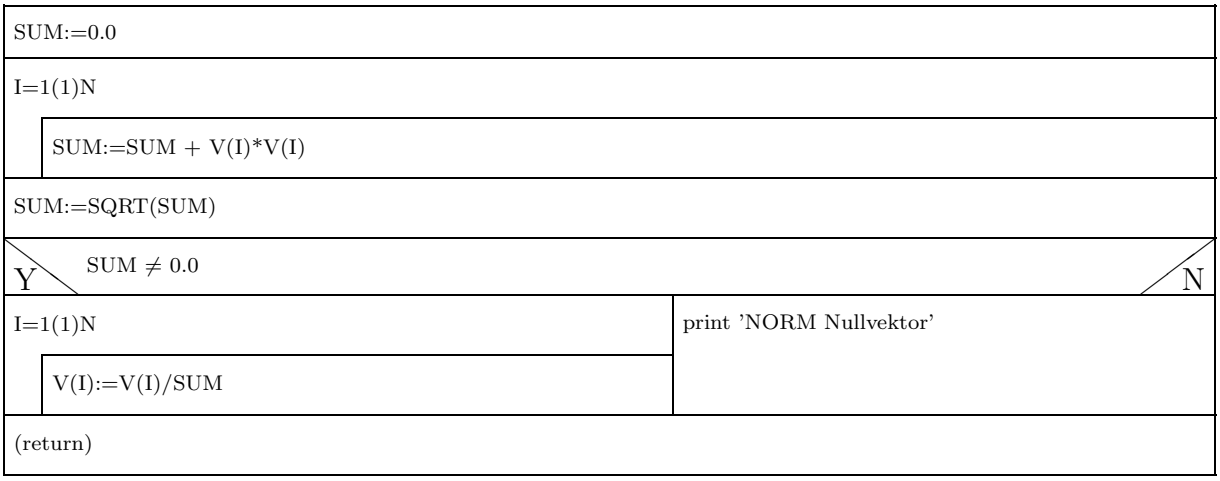
Y FEHLER N

print 'MISES1 keine Konvergenz'

NORM(V,N)

(return)

Struktogramm 21a — NORM(V,N)



7.3.4 v. Mises-Verfahren: Tests und Probleme.

Die folgenden Test-Beispiele sollen dazu dienen, die Zuverlässigkeit des Programms MISES zu überprüfen.

Die Tests beginnen mit der 4x4-Matrix

```
3.8000    1.8000   -2.0000   -0.6000
5.4000    6.2000   -7.2000   -1.0000
2.0000    2.4000   -2.0000    0.0000
1.8000    1.0000    0.0000    1.0000
```

Es soll der betragskleinste Eigenwert bzw. der zugehörige Eigenvektor berechnet werden. Das exakte Ergebnis lautet

$$\lambda_4 = 0.6 \qquad \text{und} \qquad \mathbf{x}_4 = \frac{1}{\sqrt{23}} \begin{pmatrix} 1 \\ -3 \\ -2 \\ 3 \end{pmatrix} = \begin{pmatrix} 0.2085144 \\ -0.6255432 \\ -0.4170288 \\ 0.6255432 \end{pmatrix} .$$

Es soll nun die Wirkung der Spektralverschiebung (s. Abschnitt 7.3.3) getestet werden. Der erste Test erfolgte *ohne* diese Technik, d. h. der Schätzwert für λ_4 wurde Null gesetzt:

```
.
MISES-Test:   n =    4   rel. Gen. =  1.0000000000E-06

Schaetzwert fuer EW =   0.000000

Matrix und Anfangsvektor:
```

```
    3.8000    1.8000   -2.0000   -0.6000       1.0000
    5.4000    6.2000   -7.2000   -1.0000       1.0000
    2.0000    2.4000   -2.0000    0.0000       1.0000
    1.8000    1.0000    0.0000    1.0000       1.0000

betragskleinster Eigenwert nach 20 Iterationen =   0.600000

                Eigenvektor:
       1     -0.208514
       2      0.625543
       3      0.417029
       4     -0.625543
```

Nimmt man als Schätzwert $\lambda_0 = 0.5$, so liefert das Programm MISES1:

```
.
MISES-Test:   n =    4   rel. Gen. =  1.0000000000E-06

Schaetzwert fuer EW =   0.500000

Matrix und Anfangsvektor:

    3.8000    1.8000   -2.0000   -0.6000       1.0000
    5.4000    6.2000   -7.2000   -1.0000       1.0000
    2.0000    2.4000   -2.0000    0.0000       1.0000
    1.8000    1.0000    0.0000    1.0000       1.0000

betragskleinster Eigenwert nach 8 Iterationen =   0.600000

                Eigenvektor:
       1     -0.208514
       2      0.625543
       3      0.417029
       4     -0.625543
```

Anmerkung:
Wie Sie sehen, gibt MISES1 nicht den theoretisch vorgegebenen Eigenvektor, sondern diesen Vektor mal dem Faktor (–1). Liegt hier ein Fehlverhalten des Programmes vor?

Zuletzt noch ein MISES1-Test für die 3x3-Matrix

```
    1.0000    0.0000   -1.0000
    1.0000    2.0000    1.0000
   -2.0000   -2.0000    2.0000
```

mit den entarteten Eigenwerten λ_1 und λ_2. Der dritte (und betragskleinste) Eigenwert bzw. sein Eigenvektor lauten:

$$\lambda_3 = 1.0 \qquad \text{und} \qquad \mathbf{x}_3 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.7071068 \\ -0.7071068 \\ 0.0000000 \end{pmatrix} .$$

```
.
MISES-Test:   n =    3   rel. Gen. =  1.0000000000E-06

Schaetzwert fuer EW =   0.000000

Matrix und Anfangsvektor:

    1.0000    0.0000   -1.0000       1.0000
    1.0000    2.0000    1.0000       0.0000
   -2.0000   -2.0000    2.0000       0.0000

betragskleinster Eigenwert nach 24 Iterationen =   1.000000

                Eigenvektor:
       1      0.707107
       2     -0.707107
       3      0.000000
```

7.4 Das Verfahren von Jacobi.

Dieses Verfahren ist geeignet, *das vollständige Eigenwertproblem symmetrischer Matrizen zu lösen.*
Gegeben ist also eine reelle Matrix

$$A = [a_{ij}] \qquad \text{mit} \qquad a_{ij} = a_{ji} \quad .$$

Es ist eine wesentliche allgemeine Eigenschaft reeller symmetrischer Matrizen, daß sie *ausschließlich reelle Eigenwerte* haben.

Das Prinzip des Jacobi-Verfahrens besteht aus einer orthogonalen Transformation der Matrix A in eine Diagonalmatrix D (vgl. Kap. 7.1.1):

$$U^T \cdot A \cdot U = D \tag{7.24}$$

Wie bereits ausführlich diskutiert, ist bei Kenntnis der Matrix U das Eigenwert- und Eigenvektor-Problem von A vollständig gelöst:

- Die Diagonalelemente von D sind die Eigenwerte von A:

$$\lambda_i = d_{ii} \qquad (i = 1, \ldots, n)\,.$$

- Die Spalten der Transformationsmatrix U sind die (auf Betrag Eins normierten) Eigenvektoren von A.

7.4.1 Eine iterative Annäherung an die Transformationsmatrix U.

Das Problem besteht nun darin, die Transformationsmatrix U zu ermitteln. Beim Jacobi-Verfahren wird die Transformation (7.24) durch eine sukzessive Folge von Ähnlichkeitsoperationen ersetzt. Selbstverständlich müssen auch die dabei verwendeten Transformationsmatrizen U_t $(t = 0, 1, 2, \ldots)$ orthogonal sein.

Anstelle von (7.24) gilt also:

$$U_0^T A U_0 \equiv A^{(1)}$$

$$U_1^T A^{(1)} U_1 \equiv A^{(2)} = U_1^T U_0^T A U_0 U_1$$

$$\vdots$$

$$U_t^T A^{(t)} U_t \equiv A^{(t+1)} = U_t^T U_{t-1}^T \cdots U_0^T A U_0 \cdots U_{t-1} U_t\,,$$

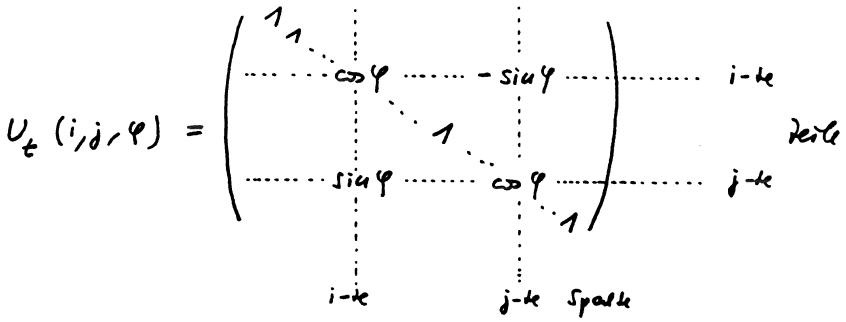
wobei die Einzel-Transformationen so beschaffen sein sollen, daß die Grenzwerte dieses Prozesses wie folgt lauten:

$$\lim_{t\to\infty} A^{(t)} \quad \to \quad D$$

$$\lim_{t\to\infty} U_0 U_1 \cdots U_{t-1} U_t \quad \to \quad U$$

Die beim Jacobi-Verfahren verwendeten Matrizen U_t haben die allgemeine Form einer *orthogonalen Drehmatrix*

. (7.25)



Wie man sofort sieht, ist U_t durch drei Angaben eindeutig bestimmt:
i und j geben jene Zeilen bzw. Spalten der Matrix an, an deren Schnittpunkten sich die Drehmatrix von einer Einheitsmatrix unterscheidet. φ ist der Drehwinkel. Es kann leicht gezeigt werden, daß $U(i,j,\varphi)$ – unabhängig von den Argumenten – stets orthogonal ist, d.h. daß gilt:

$$U_t^T(i,j,\varphi) \cdot U_t(i,j,\varphi) = E$$

Bevor nun über die optimale Wahl der Parameter i, j, φ für jeden Iterationsschritt gesprochen wird, soll im Detail untersucht werden, was bei einer Transformation

$$U^T(i_{t-1}, j_{t-1}, \varphi_{t-1}) \cdot A^{(t-1)} \cdot U(i_{t-1}, j_{t-1}, \varphi_{t-1}) \quad \rightarrow \quad A^{(t)} \tag{7.26}$$

geschieht. Komponentenweise ausgeschrieben, entspricht die Transformation dem Ausdruck

$$a_{kl}^{(t)} = \sum_{m=1}^{n} \sum_{m'=1}^{n} u_{mk} u_{m'l} a_{mm'}^{(t-1)} \quad . \tag{7.27}$$

Wie man der obigen Gleichung sofort entnehmen kann, bleibt die Symmetrie der Matrix $A^{(t-1)}$ bei dieser Transformation erhalten; es gilt also

$$a_{kl}^{(t)} = a_{lk}^{(t)} \quad .$$

Als nächstes soll (7.27) unter der Bedingung ausgewertet werden, daß *k und l weder i noch j sind*. Unter diesen Umständen sind die Komponenten der Drehmatrix (7.25) einfach 'Kronecker-Deltas', und man erhält

$$a_{kl}^{(t)} = \sum_{m=1}^{n} \sum_{m'=1}^{n} \delta_{mk} \delta_{m'l} a_{mm'}^{(t-1)} = a_{kl}^{(t-1)} \quad .$$

Das bedeutet: *Bei der Transformation bleiben alle Komponenten von A, die weder den Index i noch den Index j enthalten, unverändert.*

Die noch fehlenden Komponenten aus der i-ten und j-ten Zeile und Spalte können ebenfalls unter Verwendung von (7.27) bestimmt werden:

$$l = i \qquad k = 1, \ldots, n \quad \text{mit} \quad k \neq i, j \quad :$$

$$a_{ki}^{(t)} = a_{ik}^{(t)} = a_{ki}^{(t-1)} \cos\varphi + a_{kj}^{(t-1)} \sin\varphi \tag{7.28}$$

$$l = j \qquad k = 1, \ldots, n \quad \text{mit} \quad k \neq i, j \quad :$$

$$a_{kj}^{(t)} = a_{jk}^{(t)} = a_{kj}^{(t-1)} \cos\varphi - a_{ki}^{(t-1)} \sin\varphi \tag{7.29}$$

Nun fehlen nur noch die Komponenten

$$a_{ii}^{(t)} \qquad a_{jj}^{(t)} \qquad a_{ij}^{(t)} = a_{ji}^{(t)} \quad .$$

Für diese lauten die Transformationsgleichungen:

$$a_{ii}^{(t)} = a_{ii}^{(t-1)} \cos^2\varphi + 2a_{ij}^{(t-1)} \cos\varphi \sin\varphi + a_{jj}^{(t-1)} \sin^2\varphi \tag{7.30}$$

$$a_{jj}^{(t)} = a_{jj}^{(t-1)} \cos^2\varphi - 2a_{ij}^{(t-1)} \cos\varphi \sin\varphi + a_{ii}^{(t-1)} \sin^2\varphi \tag{7.31}$$

$$a_{ij}^{(t)} = a_{ij}^{(t-1)} (\cos^2\varphi - \sin^2\varphi) + (a_{jj}^{(t-1)} - a_{ii}^{(t-1)}) \cos\varphi \sin\varphi \tag{7.32}$$

7.4.2 Die richtige Wahl der Parameter i, j und φ.

Der Zweck jeder der orthogonalen Transformationen, der die Matrix A unterzogen wird, ist der, die entstehenden Matrizen

$$A^{(1)}, A^{(2)}, \ldots, A^{(t-1)}, A^{(t)}, A^{(t+1)}, \ldots$$

iterativ der gewünschten Diagonalform immer näher zu bringen.

Ein Maß dafür, inwieweit dies gelungen ist, ist offenbar *die Summe S der Quadrate der Nicht-Diagonalelemente der Matrix*:

$$S^{(t)} = 2 \sum_{m=1}^{n-1} \sum_{m'=m+1}^{n} \left(a_{mm'}^{(t)} \right)^2 \quad .$$

Wenn nun im Laufe der Jacobi-Iteration die Transformation

$$U_{t-1}^T \cdot A^{(t-1)} \cdot U_{t-1} \quad \rightarrow \quad A^{(t)}$$

durchgeführt wird, so ist diese dann als erfolgreich zu bezeichnen, wenn

$$S^{(t-1)} > S^{(t)} \quad ,$$

gilt, wobei $S^{(t-1)}$ und $S^{(t)}$ die Summen der Quadrate der Nicht-Diagonalelemente der Matrizen $A^{(t-1)}$ und $A^{(t)}$ sind.

Ohne Beweis: Es läßt sich zeigen, daß die Differenz von $S^{(t-1)}$ und $S^{(t)}$ einfach dem Ausdruck

$$S^{(t-1)} - S^{(t)} = 2 \left[\left(a_{ij}^{(t-1)} \right)^2 - \left(a_{ij}^{(t)} \right)^2 \right] \tag{7.33}$$

entspricht, also nur durch die Elemente a_{ij} *vor und nach* der Transformation bestimmt wird.

Die Gleichung (7.33) führt zur sinnvollsten Strategie bei der Auswahl der noch offenen Parameter i, j und φ für einen bestimmten Iterationsschritt: *Die Verkleinerung von S ist dann ein Maximum,*

- *wenn* $a_{ij}^{(t-1)}$ *dem Betrage nach möglichst groß ist und*
- *wenn* $a_{ij}^{(t)}$ *durch die Transformation zu Null wird.*

i und j sind also einfach die Indizes der betragsgrößten Nicht-Diagonalelemente der Matrix A vor dem jeweiligen Transformationsschritt[2], und die zweite Bedingung kann durch eine geeignete Wahl des Drehwinkels φ erfüllt werden. Durch das Nullsetzen von (7.32) erhält man[3]

$$\tan 2\varphi = \frac{2a_{ij}^{(t-1)}}{a_{ii}^{(t-1)} - a_{jj}^{(t-1)}} \quad . \tag{7.34}$$

Als Sonderfall ergibt sich für $a_{ii}^{(t-1)} = a_{jj}^{(t-1)}$ der Drehwinkel

$$\varphi = \frac{\pi}{4} \quad .$$

Wählt man nun vor jedem Iterationsschritt die Parameter der Drehmatrix (7.25) auf die soeben beschriebene Weise, ist eine *monotone* Abnahme von S

$$S^{(0)} > S^{(1)} > S^{(2)} > \cdots > S^{(t)} > \cdots$$

und dadurch eine immer stärkere Annäherung an die gewünschte Diagonalform garantiert.

Zusätzlich soll auch noch jene Matrix berechnet werden, die sich aus dem Produkt der einzelnen Transformationsmatrizen ergibt:

$$B^{(t-1)} = U_0 U_1 U_2 \cdots U_{t-2} U_{t-1} \quad , \tag{7.35}$$

weil die Matrix näherungsweise die gesuchten Eigenvektoren der Matrix A enthält. Wieder kann gezeigt werden, daß bei der Multiplikation von B mit einer Drehmatrix $U_t(i,j,\varphi)$ nur die Elemente der i-ten und j-ten Spalte von B verändert werden:

$$\Big(B^{(t-1)} U_t(i,j,\varphi)\Big)_{k,i} = b_{ki}\cos\varphi + b_{kj}\sin\varphi \tag{7.36}$$

$$\Big(B^{(t-1)} U_t(i,j,\varphi)\Big)_{k,j} = b_{kj}\cos\varphi - b_{ki}\sin\varphi \tag{7.37}$$

für $k = 1, \ldots, n$.

[2] Allerdings ist das Aufsuchen des betragsgrößten Nicht-Diagonalelementes vor jeder Matrix-Transformation sehr zeitintensiv. Aus diesem Grund wird in den meisten Programmen ein etwas einfacheres Auswahlverfahren angewendet, s. Abschnitt 7.4.3.

[3] Eine Variation dieser Vorgangsweise für C-Programme s. Abschnitt 7.4.4.

7.4.3 Das Programm JACOBI.

Das Programm JACOBI löst das vollständige Eigenwertproblem einer reellen, symmetrischen Matrix.

INPUT-Parameter:

A(,): die Komponenten einer reellen, symmetrischen Matrix. Da das Programm JACOBI so formuliert ist, daß immer nur auf die Matrixelemente auf und oberhalb der Hauptdiagonale zugegriffen wird, brauchen *nur diese Elemente vom aufrufenden Programm bereitgestellt werden.*

N: Zeilen und Spalten der Matrix.

TMAX: maximale Anzahl der Rechendurchläufe.

OUTPUT-Parameter:

A(,): Näherung für eine Diagonalmatrix mit demselben Eigenwertspektrum wie die ursprüngliche Matrix A. Die Diagonalelemente dieser Matrix sind somit Näherungswerte für die Eigenwerte.

EIGVEK(,): Diese Matrix enthält die Eigenvektoren der Matrix A, wobei die j-te Spalte von EIGVEK dem j-ten Eigenvektor von A entspricht.

Struktogramm 22 — JACOBI(A,N,TMAX,EIGVEK)

Struktogramm 23 — (Fortsetzung)

T:=0

T:=T+1

SUMME:=0.0

I=1(1)N-1

J=I+1(1)N

SUMME:=SUMME+2.0*A(I,J)*A(I,J)

IF(SUMME=0.0) ===> (return)

GRENZE:=SQRT(SUMME)/N/N

I=1(1)N-1

J=I+1(1)N

Y | A(I,J) | > GRENZE N

NENNER:=A(I,I)–A(J,J)

.....

Y | NENNER/A(I,J) | < EPS N

PHI:=PI/4.0

PHI:=0.5*ATAN(2.0*A(I,J)
/NENNER)

SINUS:=SIN(PHI)
COSIN:=COS(PHI)

K=I+1(1)J-1

SPEICH:=A(I,K)
A(I,K):=COSIN*A(I,K)+SINUS*A(K,J)
A(K,J):=COSIN*A(K,J)–SINUS*SPEICH

K=J+1(1)N

SPEICH:=A(I,K)
A(I,K):=COSIN*A(I,K)+SINUS*A(J,K)
A(J,K):=COSIN*A(J,K)–SINUS*SPEICH

K=1(1)I-1

SPEICH:=A(K,I)
A(K,I):=COSIN*A(K,I)+SINUS*A(K,J)
A(K,J):=COSIN*A(K,J)–SINUS*SPEICH

SPEICH:=A(I,I)
A(I,I):=COSIN**2*A(I,I)+2.0*COSIN*SINUS*A(I,J)+SINUS**2*A(J,J)
A(J,J):=COSIN**2*A(J,J)–
2.0*COSIN*SINUS*A(I,J)+SINUS**2*SPEICH
A(I,J):=0.0

K=1(1)N

SPEICH:=EIGVEK(K,J)
EIGVEK(K,J):=COSIN*EIGVEK(K,J)–SINUS*EIGVEK(K,I)
EIGVEK(K,I):=COSIN*EIGVEK(K,I)+SINUS*SPEICH

T > TMAX

print 'JACOBI Konvergenz nicht erreicht'
(return)

Programm-Struktur:

1. Nach der Definition der Konstanten PI und der 'Pseudonull' EPS=10^{-8} wird auf das Feld EIGVEK eine Einheitsmatrix abgespeichert.

2. UNTIL-Schleife für die (maximal) TMAX Rechendurchläufe:

   - Bestimmung der Summe der Quadrate der Nicht-Diagonalelemente sowie des 'Schwellenwertes' GRENZE.
   - Abfrage, ob diese Summe bereits Null ist.

     Die Konvergenz des Jacobi-Verfahrens ist zumeist so gut, daß man die Rechnung bis zum 'Underflow' von SUMME treiben kann. Die obige Abfrage auf die exakte Null funktioniert jedoch nur dann, wenn vom System her ein Underflow als Null interpretiert wird!

     Wenn SUMME=0.0, Return ins aufrufende Programm.

     Wenn SUMME>0.0, werden Matrix-Transformationen gemäß (7.28 - 7.32) bzgl. *all jener Nicht-Diagonalelemente durchgeführt, deren Beträge > GRENZE sind.*
   - Nach jeder Transformation wird die Matrix EIGVEK mit der bei diesem Iterationsschritt verwendeten orthogonalen Matrix U_t multipliziert (7.36 und 7.37).

3. Ist nach TMAX Rechendurchgängen SUMME immer noch größer als Null, erfolgt eine Fehlermitteilung und ein Return ins aufrufende Programm.

7.4.4 Variation des JACOBI-Algorithmus in C.

In FORTRAN-, PASCAL-, ... -Programmen kann die Berechnung des idealen Drehwinkels gemäß Glg. (7.34) erfolgen.
In Programmiersprachen ohne Arcustangens-Funktion wie z. B. in C kann die Berechnung von $\cos\varphi$ und $\sin\varphi$ auf die folgende Weise geschehen (mathematische Darstellung s. [10], S. 464f):

```
.
  g=100*fabs(a[i][j]);
  if(fabs(a[i][j]) > grenze) {
    h=a[i][i]-a[j][j];
// This if statement prevents an overflow of theta*theta in the following
// statement.
    if(fabs(h)+g == fabs(h)) tfac=a[i][j]/h;
    else {
      theta=h/2.0/a[i][j];
      tfac=1.0/(fabs(theta)+sqrt(1.0+theta*theta));
      if(theta<0.0)tfac=-tfac;
    }
    cosin=1.0/sqrt(tfac*tfac+1.0);
    sinus=tfac*cosin;
```

7.4.5 Zwei Testbeispiele für JACOBI.

Erstes Beispiel:

```
.
Matrix:   5.000   4.000   1.000   1.000
          4.000   5.000   1.000   1.000
          1.000   1.000   4.000   2.000
          1.000   1.000   2.000   4.000

exakte Loesung:
Eigenwert:                   Eigenvektor:

10  (2,2,1,1)/sqrt(10)   = (0.6324555, 0.6324555, 0.3162278, 0.3162278)
 1  (-1,1,0,0)/sqrt(2)   = (-0.7071068, 0.7071068, 0.0, 0.0)
 5  (-1,-1,2,2)/sqrt(10) = (-0.3162278, -0.3162278, 0.6324555, 0.6324555)
 2  (0,0,-1,1)/sqrt(2)   = (0.0, 0.0, -0.7071068, 0.7071068)

Jacobi-Loesung:
Eigenwert:                   Eigenvektor:

10.000000       0.632456    0.632456    0.316228    0.316228
 1.000000      -0.707107    0.707107   -0.000000   -0.000000
 5.000000      -0.316228   -0.316228    0.632456    0.632456
 2.000000      -0.000000   -0.000000   -0.707107    0.707107
```

Zweites Beispiel:

```
.
Matrix:   6.000   4.000   4.000   1.000
          4.000   6.000   1.000   4.000
          4.000   1.000   6.000   4.000
          1.000   4.000   4.000   6.000

exakte Loesung:
Eigenwert:                   Eigenvektor:
15              0.5         0.5         0.5         0.5
-1              0.5        -0.5        -0.5         0.5
 5             -0.5         0.5        -0.5         0.5
 5             -0.5        -0.5         0.5         0.5

Jacobi-Loesung:
Eigenwert:                   Eigenvektor:

15.000000       0.500000    0.500000    0.500000    0.500000
-1.000000      -0.500000    0.500000    0.500000   -0.500000
 5.000000       0.226248   -0.669934    0.669934   -0.226248
 5.000000      -0.669934   -0.226248    0.226248    0.669934
```

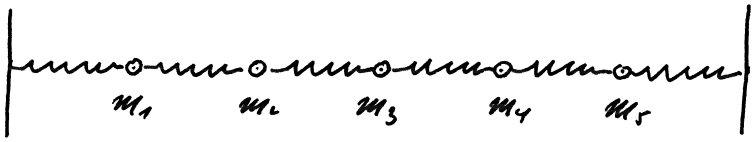
Zu diesen Ergebnissen noch zwei Anmerkungen:

1. Die normierten Eigenvektoren sind natürlich nur bis auf eine etwaige (multiplikative) Konstante (–1) bestimmt. Es ist daher *kein Fehler*, wenn das JACOBI-Programm den Eigenvektor zum Eigenwert -1 mit verändertem Vorzeichen ausgibt!

2. Noch stärker wirkt sich die Nicht-Eindeutigkeit von Eigenvektoren im Falle entarteter Eigenwerte aus. Im Fall des hier vorkommenden zweifach entarteten Eigenwertes 5 bedeutet das, daß jede Linearkombination der von JACOBI gelieferten Eigenvektoren wieder einen Eigenvektor zum Eigenwert 5 ergibt. Sie können sich ohne weiteres überzeugen, daß es möglich ist, die beiden Vektoren $(-0.5, 0.5, -0.5, 0.5)$ und $(-0.5, -0.5, 0.5, 0.5)$ so linear zu kombinieren, daß man die mittels JACOBI berechneten Eigenvektoren erhält.

7.4.6 Ein Anwendungsbeispiel für JACOBI.

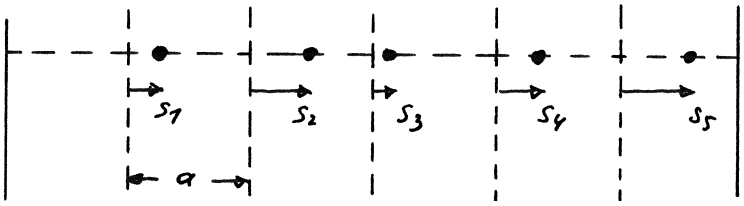
Gegeben ist ein System von n gekoppelten Federpendeln, wobei die schwingenden Körper die Massen m_i, $i = 1, \ldots, n$ haben:



Für die weitere Diskussion werden die folgenden Annahmen gemacht:

- Die Federkräfte seien so groß bzw. die Massen so klein, daß Gravitationskräfte vernachlässigt werden können.
- Die Schwingungsamplituden seien klein gegenüber den Abständen der Teilchen-Ruhepositionen.

Die momentanen Auslenkungen der Massenpunkte werden mit s_i bezeichnet, und die Kräfte zwischen den Massen werden im Sinne einer *harmonischen Näherung* als proportional zu ihren Abständen angenommen. Der Proportionalitätsfaktor zwischen Abstand und Kraft, die *Federkonstante* D, wird in cgs-Einheiten (dyn/cm) angegeben. a ist der Abstand zwischen 2 Massepunkten in der Ruhelage:



Wird ein solches System in Schwingungen versetzt und dann sich selbst überlassen, so läßt sich für jeden Massenpunkt eine Bewegungsgleichung der Art

$$m_i \ddot{s}_i + D(a - s_{i-1} + s_i) - D(a - s_i + s_{i+1}) = 0$$

aufstellen (m_i ist die Masse des i-ten Massenpunktes, und $s_i(t)$, $s_{i-1}(t)$, $s_{i+1}(t)$ sind die momentanen Auslenkungen des i-ten Punktes sowie seines linken und rechten Nachbarn). Man erhält schließlich ein *gekoppeltes System von n gewöhnlichen Differentialgleichungen zweiter Ordnung*:

$$\begin{aligned} m_1\ddot{s}_1 + 2Ds_1 - Ds_2 &= 0 \\ m_i\ddot{s}_i - Ds_{i-1} + 2Ds_i - Ds_{i+1} &= 0 \qquad i = 2, 3, \ldots, n-1 \\ m_n\ddot{s}_n - Ds_{n-1} + 2Ds_n &= 0 \end{aligned} \tag{7.38}$$

Beschränkt man sich bei der Lösung dieses Systems auf kleine Auslenkungen, so kann das System (7.38) unter Verwendung der Ansatzfunktion

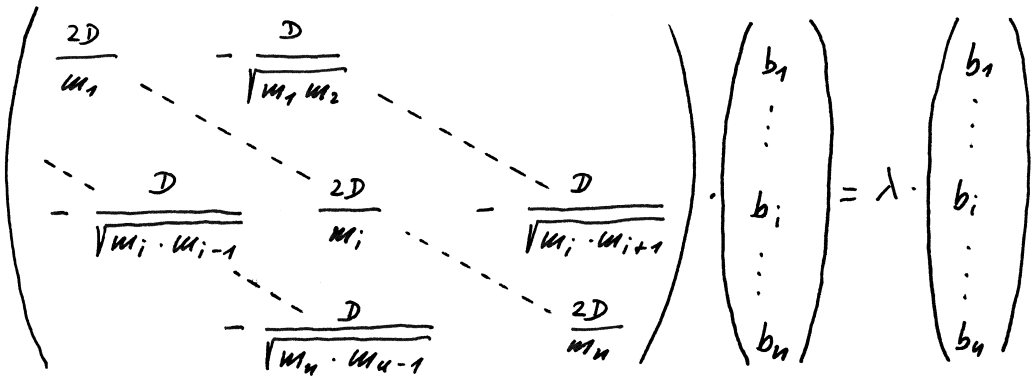
$$s_i(t) = \frac{b_i}{\sqrt{m_i}} \cdot e^{i\omega t}$$

in das *homogene, lineare Gleichungssystem*

$$\begin{aligned} \left(\frac{2D}{m_1} - \omega^2\right) b_1 - \frac{D}{\sqrt{m_1 m_2}} b_2 &= 0 \\ -\frac{D}{\sqrt{m_i m_{i-1}}} b_{i-1} + \left(\frac{2D}{m_i} - \omega^2\right) b_i - \frac{D}{\sqrt{m_i m_{i+1}}} b_{i+1} &= 0 \quad (i = 2, 3, \ldots, n) \\ -\frac{D}{\sqrt{m_n m_{n-1}}} b_{n-1} + \left(\frac{2D}{m_n} - \omega^2\right) b_n &= 0 \end{aligned}$$

übergeführt werden. Auf diese Weise erhält man das *reguläre Eigenwertproblem*

. (7.39)



wobei $\omega^2 = \lambda$ gesetzt wurde. Die (reellen) Eigenwerte der obigen *symmetrisch-tridiagonalen Matrix* sind also die Quadrate der Eigenfrequenzen des schwingenden Systems (kleinste Frequenz=Grundschwingung und $n-1$ Oberschwingungen).

```
Testbeispiel "Federpendel" fuer Jacobi:
=======================================

Federkonstante [dyn/cm] = 25.000
5 Massenpunkte:
  m1 = 3 g   m2 = 6 g   m3 = 9 g   m4 = 2 g   m5 = 6 g

JACOBI liefert das folgende Resultat:

   nr    lambda(1/s/s)     Kreisfrequenz(1/s)

   1      19.858498            4.456287
   2       1.135214            1.065464 (Grundschwingung)
   3       5.525477            2.350633
   4      29.036367            5.388540
   5       8.333333            2.886751
```

7.4.7 Erweiterte symmetrische Eigenwertprobleme

Viele interessante physikalisch-technische Problemstellungen führen nicht unmittelbar zu einem Eigenwertproblem vom Typus (7.5), sondern auf ein homogenes lineares Gleichungssystem in der Art

$$(A - \lambda S)\mathbf{x} = 0 \tag{7.40}$$

mit der symmetrischen Matrix A und der ebenfalls symmetrischen und zusätzlich noch *positiv-definiten* Matrix S (in manchen physikalischen Anwendungen wird S als *Strukturmatrix* bezeichnet).

Wie kann man nun das *erweiterte Eigenwertproblem* (7.40) auf die Form

$$(A' - \lambda E)\mathbf{x} = 0$$

bringen? Diese Frage scheint einfach zu lösen zu sein: man multipliziert die Glg. (7.40) von links mit der Inversen von S und erhält sofort

$$\left(S^{-1}A - \lambda E\right)\mathbf{x} = 0$$

mit der neuen Koeffizientenmatrix $A' = S^{-1}A$.

Leider ist diese simple Umformung nicht sehr brauchbar, weil nämlich die neue Koeffizentenmatrix A' die wichtige Eigenschaft der Symmetrie verloren hat:

$$A'_{i,j} = \sum_{k=1}^{n} s^{-1}_{i,k}\, a_{k,j}$$

ist im allgemeinen nicht identisch mit

$$A'_{j,i} = \sum_{k=1}^{n} s^{-1}_{j,k}\, a_{k,i}\,.$$

Eine bessere, allerdings etwas aufwändigere Methode der Umformung soll nun präsentiert werden:

Ausgangspunkt ist die sogenannte *Cholesky decomposition* der symmetrischen, positiv-definiten Matrix S:

$$S = L\,L^T \tag{7.41}$$

Beachten Sie die Ähnlichkeit dieser Aufspaltung mit der im Kapitel 2 behandelten LU *decomposition* (2.4): der Unterschied besteht nur darin, daß eine symmetrische, positiv definite Matrix als Produkt einer unteren Dreiecksmatrix L und deren um die Hauptachse transponierten oberen Dreieckmatrix L^T dargestellt werden kann.

Hat man die Cholesky-Prozedur erledigt, ist die weitere Vorgangsweise relativ einfach: Einsetzen von Glg. (7.41) in die erweiterte Eigenwertgleichung (7.40) gibt

$$\left(A - \lambda L L^T\right)\mathbf{x} = 0\,.$$

Hebt man (von rechts!) die Matrix L^T aus der Klammer heraus, folgt weiter

$$\left(A(L^T)^{-1} - \lambda L\right)\left(L^T\,\mathbf{x}\right) = 0$$

und

$$\left(\underbrace{L^{-1}A(L^T)^{-1}}_{C} - \lambda E\right)\left(L^T\mathbf{x}\right) = 0$$

bzw.[4]

$$(C - \lambda E)\mathbf{y} = 0 \qquad \text{mit} \quad C = L^{-1}A\left(L^{-1}\right)^T \qquad \text{und} \quad \mathbf{y} = L^T\mathbf{x}\,. \tag{7.42}$$

Die Eigenwerte der Matrix C (daß sie symmetrischi sind, ist leicht zu beweisen - probieren Sie es selbst) sind identisch mit den Eigenwerten von A in Glg. (7.40).

Aus den Eigenvektoren $\mathbf{y}$ des Systems (7.42) können mittels

$$\mathbf{x} = \left(L^{-1}\right)^T\,\mathbf{y} \tag{7.43}$$

die Eigenvektoren $\mathbf{x}$ von Glg. (7.40) bestimmt werden.

Die Formeln, die zur numerischen Ausführung der *Cholesky decomposition* erforderlich sind, können ohne Schwierigkeiten hergeleitet werden. Man geht dabei von der Komponentendarstellung von Glg. (7.41) aus:

$$s_{i,j} = \sum_{k=1}^{n} \ell_{ik}\ell_{jk} = \sum_{k=1}^{j} \ell_{ik}\ell_{jk}\,.$$

[4]Hier wird die Identität $\left(L^T\right)^{-1} = \left(L^{-1}\right)^T$ verwendet.

Die Berechnung der s_{ij} erfolgt nun spaltenweise (Index j), wobei wegen der Dreiecksform von L die Bedingung $i \leq j$ gilt.

Für die erste Spalte ($j = 1$) erhält man sofort

$$s_{11} = \ell_{11}^2 \rightarrow \ell_{11} = \sqrt{s_{11}}$$

sowie

$$s_{i1} = \ell_{i1}\ell_{11} \rightarrow \ell_{i1} = \frac{s_{i1}}{\ell_{11}} \quad \text{für} \quad i = 2, \ldots, n\,.$$

Analog erhält man für die zweite Spalte ($j = 2$)

$$s_{22} = \ell_{21}^2 + \ell_{22}^2 \rightarrow \ell_{22} = \sqrt{s_{22} - \ell_{21}^2}$$

und

$$s_{i2} = \ell i 1 \ell 21 + \ell_{i2}\ell_{22} \rightarrow \ell_{i2} = \frac{s_{i2} - \ell_{i1}\ell_{21}}{\ell_{22}} \quad \text{für} \quad i = 3, \ldots, n\,.$$

Aus diesen Gleichungen kann man ohne weiteres auf die allgemeinen Cholesky-Formeln schließen:

$$\ell_{jj} = \sqrt{s_{jj} - \sum_{t=1}^{j-1} \ell_{jt}^2}\,,$$

$$\ell_{i,j} = \frac{s_{ij} - \sum_{t=1}^{j-1} \ell_{it}\ell_{jt}}{\ell_{jj}} \quad \text{für} \quad (i = j+1, \ldots, n) \tag{7.44}$$

Zu diesem Gleichungen noch 2 Anmerkungen:

- Offensichtlich gibt es Probleme, wenn das Argument einer Quadratwurzel im Laufe der Cholesky-Rechnung Null oder negativ wird. Genau das kann man aber bei einer *positiv-definiten* Matrix S ausschließen. Im Programm muß also vor jeder Wurzelziehung abgefragt werden, ob

  $$\left(s_{jj} - \sum_{t=1}^{j-1} \ell_{jt}^2 \right) > 0\,.$$

  Ist dies nicht der Fall, ist die Matrix S nicht *positiv-definit* und die Rechnung muß abgebrochen werden!

- Eine genaue Analyse der Gleichungen (7.44) zeigt, daß die zur Berechnung der ℓ_{ij} erforderlichen s_{ij}-Werte jeweils nur einmal gebraucht werden und später nie wierder: das gibt die Möglichkeit, die Werte der L-Matrix (spaltenweise uns suzessive) auf den Speicherplätzen der S-Matrix abzuspeichern. Eine ähnlich platzsparende Vorgangsweise wurde auch im Programm LUDCMP realisiert.
  Nachteil: die ursprüngliche S-Matrix geht dabei verloren.

7.4.8 Das Programm CHOLESKY.

Die numerische Auswertung der Gleichungen (7.44) wird im folgenden Struktrogramm Nr. 23 CHOLESKY beschrieben. Um genau zu sein, im ersten (kleineren) Teil dieses Programms. Der zweite Teil von CHOLESKY beschreibt die numerische Auswertung der *neuen* Matrix C gemäß Glg. (7.42). Den entsprechenden Algorithmus-Teil im Struktogramm 23 habe ich dem Algol-Programm *reduc1* von Martin und Wilkinson[5] entnommen. Ich muß gestehen, daß ich bisher noch keine Zeit fand, diesen Algorithmus schlüssig aus der Gleichung (7.42) nachzuvollziehen.

INPUT-Parameter:

N: Zeilen und Spalten der Matrizen A uns S.

A(,): die Komponenten einer reellen, symmetrischen Matrix.

S(,): die Komponenten einer reellen, symmetrischen, *positiv-definiten* Matrix.

OUTPUT-Parameter:

S(,): die Komponenten der tridiagonalen Matrix L.

A(,): die Komponenten der symmetrischen Matrix C.

FEHLER: Fehlerdiagnostik: FEHLER=0 kein Fehler aufgetreten
FEHLER=1 Achtung:
Input-Matrix S ist nicht positiv-definit.

[5]Martin and Wilkinson, Num. Math. **11**, 99 (1968); *Handbook for Autom. Computing*, vol. II, p. 303 (1971).

Struktogramm 23 — CHOLESKY(N,A,S,FEHLER)

FEHLER:=0

I=1(1)N

J=I(1)N

X:=S(I,J)

K=1(1)I-1

X:=X-S(I,K)*S(J,K)

J ≠ I — Y / N

S(J,I):=X/Y

X ≤ 0.0 — Y / N

FEHLER:=1
print 'Matrix S nicht pos-def.'
(return)

.....

Y:=SQRT(X)
S(I,I):=Y

Anmerkung: Ende der Cholesky Decomposition.

Anmerkung: Nun beginnt die Berechnung der Matrix C (Achtung: Abspeicherung auf A):

I=1(1)N

Y:=B(I,I)

J=I(1)N

X:=A(I,J)

K=1(1)I-1

X:=X-S(I,K)*A(J,K)

A(J,I):=X/Y

J=1(1)N

I=J(1)N

X:=A(I,J)

K=J(1)I-1

X:=X-A(K,J)*S(I,K)

K=1(1)J-1

X:=X-A(J,K)*S(I,K)

A(I,J):=X/S(I,I)

I=1(1)N-1

J=I+1(1)N

A(I,J):=A(J,I)
S(I,J):=0.0

```
TESTBEISPIEL:
=============

Numerische Loesung des erweiterten Eigenwertproblems

       [A - lambda . B] x  =  0

Matrix A (symmetrisch):
 5.0000   4.0000   1.0000   1.0000
 4.0000   5.0000   1.0000   1.0000
 1.0000   1.0000   4.0000   2.0000
 1.0000   1.0000   2.0000   4.0000

Matrix B (symmetrisch und positiv-definit):
 5.0000   7.0000   6.0000   5.0000
 7.0000  10.0000   8.0000   7.0000
 6.0000   8.0000  10.0000   9.0000
 5.0000   7.0000   9.0000  10.0000

Das Programm CHOLESKY gewinnt aus den gegebenen Matrizen
A und B mittels Cholesky-Aufspaltung von B:

               B = L . L^T

die Matrix

C = L^(-1) . A . (L^(-1))^T

Die untere Dreiecksmatrix  L  lautet:

   2.2361     0.0000     0.0000     0.0000
   3.1305     0.4472     0.0000     0.0000
   2.6833    -0.8944     1.4142     0.0000
   2.2361    -0.0000     2.1213     0.7071

Diese Matrix  C  ist symmetrisch und hat dasselbe
Eigenwertspektrum wie das gegebene erweiterte Eigenwertproblem:

   1.0000    -3.0000    -3.4785     7.9057
  -3.0000    18.0000    16.4438   -41.1096
  -3.4785    16.4438    18.0000   -43.0000
   7.9057   -41.1096   -43.0000   110.0000

Mit dem Jacobi-Programm koennen die 4 Eigenwerte der
Matrix  C  ohne weiteres bestimmt werden.

Die Eigenwerte lauten: 0.2623  2.3078  1.1530  143.2769
```

7.4.9 'More advanced programs'

Das Jacobi-Verfahren ist im Prinzip für alle reellen symmetrischen Matrizen ausgezeichnet geeignet. Dennoch bieten die meisten 'Profi-Bibliotheken' noch weitere leistungsfähige Programme an, die für Matrizen höherer Ordnung ($>$ 20) deutlich schneller sind als das Jacobi-Verfahren[6].
Die Grundidee dieser Programme ist ein *Zwei-Stufen-Prozess*:

1. Die gegebene symmetrische Matrix A wird mittels eines *endlichen* (d.h. nicht-iterativen) Rechenprozesses auf eine einfachere Form gebracht, und zwar im konkreten Fall auf *tridiagonal-symmetrische* Form. Dazu wird häufig der *Householder*-Algorithmus verwendet.

2. Aus dieser Tridiagonalmatrix werden dann mittels des sog. QR- oder QL-Algorithmus die Eigenwerte und Eigenvektoren von A berechnet.

Sowohl die Householder- als auch die QR- (QL-)Methode kann im Rahmen dieser Lehrveranstaltung nicht im Detail erörtert werden. Eine sehr gute und kompakte Darstellung der Theorie sowie die entsprechenden Programme TRED2 (Householder) und TQLI (QL-Algorithmus) finden Sie in [9] und in [10].

Genauere Informationen über diese modernen Methoden erhalten Sie auch in meiner weiterführenden Lehrveranstaltung

Ausgewählte Kapitel aus 'Numerische Methoden in der Physik'
SS 2003 2 VO 2 UE (515.433 / 515.434)

7.5 Eigenwerte allgemeiner reeller Matrizen

Dieses Problem wird in der Numerik i. a. ähnlich angegangen wie bei den symmetrischen Matrizen, nämlich mittels eines zweistufigen Prozesses. Dabei wird (1) die Matrix A durch einen nicht-iterativen Algorithmus in eine einfachere Form gebracht (s. u.), und (2) werden die Eigenwerte der so erhaltenen Matrix bestimmt.

Die Grundidee basiert auf den folgenden Aussagen aus der Matrizentheorie:
Jede Matrix kann durch einen nicht-iterativen Ähnlichkeits-Prozess auf eine Upper Hessenberg Form (UHF) gebracht werden.

und

Bei einer Upper-Hessenberg-Matrix sind alle Matrix-Komponenten unterhalb der ersten unteren Nebendiagonale Null.

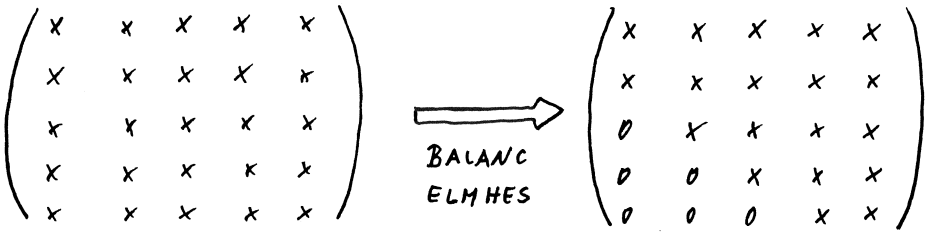


Abbildung 7.1: Reduktion einer allgemeinen Matrix auf die Upper-Hessenberg-Form.

7.5.1 Reduktion einer Matrix auf die Upper-Hessenberg-Form.

Die Umformung einer allgemeinen reellen Matrix auf eine UHF geschieht mit einem Algorithmus, der dem Gauss'schen Eliminationsverfahren (s. Kap. 2, LU-Decomposition) ähnlich ist, nur mit dem wichtigen Unterschied, daß im gegebenen Fall jede Umwandlung der Matrix eine Ähnlichkeitsoperation sein muß. Im folgenden soll dieser Algorithmus kurz erläutert werden:

Angenommen, eine 7x7-Matrix soll in die UHF gebracht werden, und der Algorithmus habe bereits einen Teil der Arbeit erledigt d.h. er habe (z. B.) bereits die ersten drei Spalten der Matrix auf Upper-Hessenberg gebracht. Dann sieht die Matrix so aus:

```
x   x   x   x   x   x   x

x   x   x   x   x   x   x

0   x   x   x   x   x   x

0   0   x   x   x   x   x

0   0   0   a   x   x   x

0   0   0   b   x   x   x

0   0   0   c   x   x   x
```

Im nun folgenden Schritt soll die vierte Spalte auf UHF gebracht werden, d.h. die letzten beiden Werte dieser Spalte sollen Null werden. Dies erreicht man dadurch, daß man die drittletzte Zeile der Matrix mit den Faktoren (b/a) bzw. (c/a) multipliziert und diese zur vorletzten bzw. letzten Zeile addiert.

[6]In diesem Zusammenhang ein interessantes Detail: Die hochangesehene Programmbibliothek LAPACK hatte eine Zeitlang überhaupt keine Jacobi-Routine im Repertoire, bis man im *User's Guide* von 1995, S. 18, den folgenden Satz findet: *In the future LAPACK will include routines based on the Jacobi algorithm ..., which are slower than the above routines (QR etc) but can be significantly more accurate.*

Wie Sie sich erinnern (Kap. 2, LU-Decomposition), ist es bei solchen Rechnungen wichtig, darauf zu achten, daß die Multiplikatoren dem Betrage nach so klein als möglich gehalten werden: dies erreicht man durch *Pivotisierung*, d.h. dadurch, daß man die in Diskussion stehenden Zeilen (im konkreten Fall diejenigen, welche a, b und c enthalten) so vertauscht, daß an der Stelle 'a' die betragsgrößte der Zahlen a, b, c zu stehen kommt.
In einem wichtigen Punkt unterscheidet sich das hier zu diskutierende Verfahren von der LU-Decomposition: *um zu gewährleisten, daß die Umformung der gegebenen Matrix A in eine UHF eine Ähnlichkeitsoperation ist, müssen alle Zeilenvertauschungen bei der Pivotisierung und alle Zeilenadditionen zur Erreichung der 'Hessenberg-Nullen' durch analoge Operationen bzgl. der Spalten ergänzt werden.*

7.5.2 Das Programm ELMHES.

Das Programm ELMHES reduziert eine allgemeine reelle Matrix in die entsprechende 'Upper-Hessenberg-Matrix'.

Quelle: [9], S. 752; [10], S. 485f.

INPUT-Parameter:

A(,): die Komponenten einer reellen Matrix.

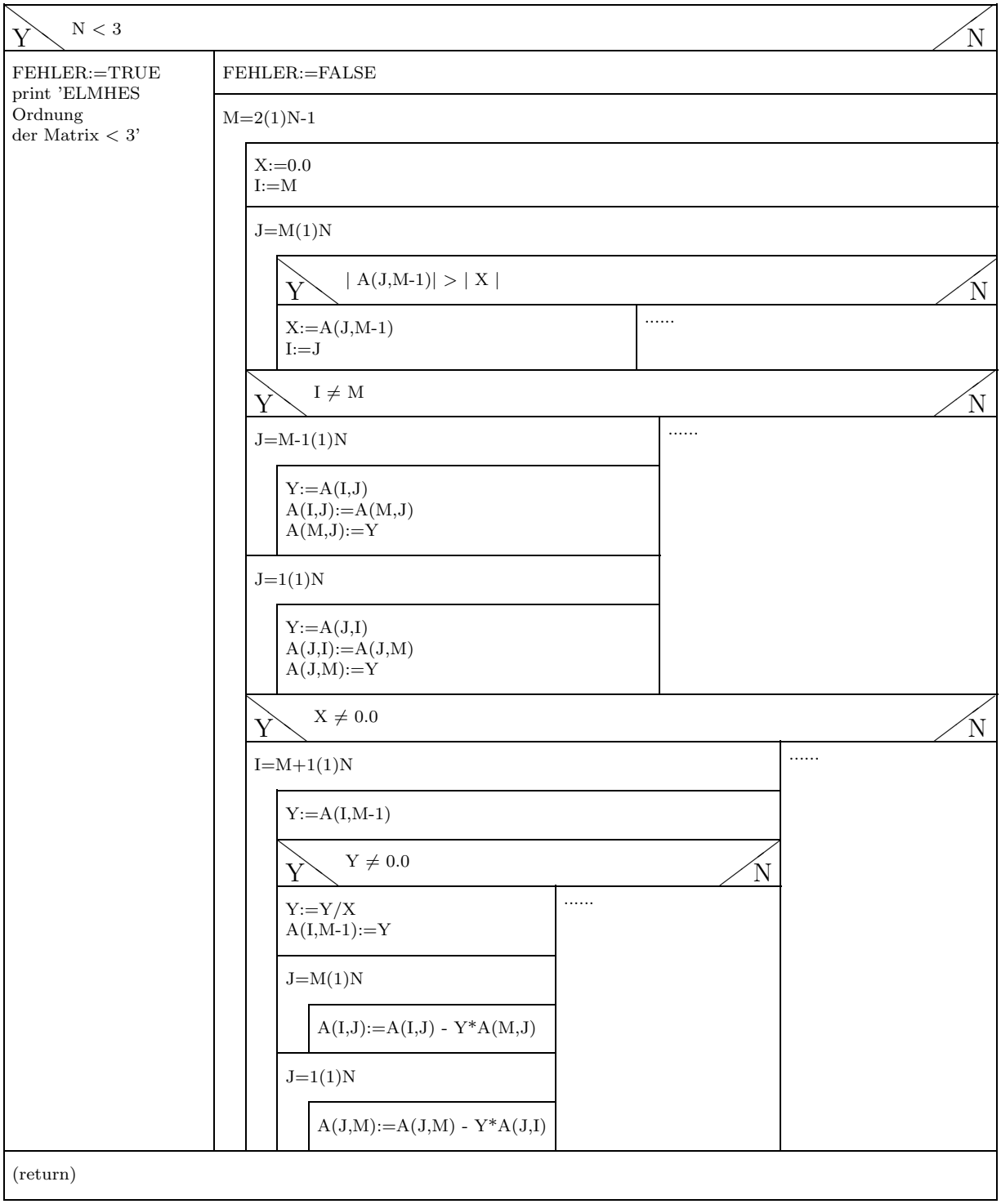
N: Zeilen und Spalten der Matrix.

OUTPUT-Parameter:

A(,): 'Upper-Hessenberg-Matrix' mit denselben Eigenwerten wie die ursprüngliche Matrix.

FEHLER: TRUE wenn Ordnung der Matrix kleiner als 3, sonst FALSE.

Struktogramm 24 — ELMHES(A,N,FEHLER)



7.5.3 Bestimmung der Eigenwerte einer Matrix in UHF.

Um die reellen und konjugiert-komplexen Eigenwerte einer reellen Upper-Hessenberg-Matrix zu berechnen, kann wieder der bereits im Abschnitt 7.4.9 erwähnte QR-Algorithmus zum Einsatz kommen. Ein Beispiel für ein derarti-

ges Programm finden Sie in den Numerical Recipes (Theorie und FORTRAN-Programm in [9], S. 374f; PASCAL-Programm in [9], S. 753f; Theorie und C-Programm in [10], S. 486ff.
Weitere einschlägige Programme siehe auch im Abschnitt 7.6.

Auch über den QR-Algorithmus können Sie in meiner weiterführenden Lehrveranstaltung

Ausgewählte Kapitel aus 'Numerische Methoden in der Physik'
SS 2003 2 VO 2 UE (515.433 / 515.434)

mehr erfahren.

Wenn man sich mit den *reellen* Eigenwerten und Eigenvektoren einer Matrix in UHF begnügt, gibt es auch eine viel einfachere Methode als QR; diese soll im folgenden beschrieben werden:

Wie Sie dem Abschnitt 7.1 entnehmen können, sind die Eigenwerte einer Matrix zugleich die Nullstellen ihres *charakteristischen Polynoms*

$$P_n(\lambda) = \lambda^n + \sum_{i=1}^{n} p_i \, \lambda^{n-i} \, .$$

Es gibt nun Algorithmen, mit denen man die Koeffizienten p_i eines solchen Polynoms bestimmen kann. Kennt man die p_i, kann man die Nullstellen des Polynoms mit einem geeigneten Nullstellen-Suchprogramm ermitteln. Dieses Verfahren ist jedoch sehr umständlich und vor allem sehr rundungsfehleranfällig und wird daher in der Praxis kaum angewendet.
Man kann jedoch im Falle einer Upper-Hessenberg-Matrix das charakteristische Polynom auswerten, *ohne explizite dessen Koeffizienten zu kennen*; dies wird im folgenden Abschnitt erläutert.

7.5.4 Die Methode von Hyman.

Hyman konnte zeigen, daß für Hessenberg-Matrizen das Polynom $P_n(\lambda)$ die Form

$$P_n(\lambda) = (-1)^{n+1} \cdot a_{21} a_{32} \cdots a_{n,n-1} \cdot H(\lambda)$$

annimmt. Natürlich sind die Polynom-Nullstellen mit den Nullstellen der Funktion $H(\lambda)$ identisch. Die Funktionswerte von $H(\lambda)$ können nun aus dem folgenden einfach strukturierten (und numerisch gut konditionierten) linearen Gleichungssystem ermittelt werden:

$$(A - \lambda \cdot E) \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_{n-1} \\ 1 \end{pmatrix} = H(\lambda) \begin{pmatrix} 1 \\ 0 \\ . \\ . \\ . \\ . \\ 0 \end{pmatrix} \quad . \tag{7.45}$$

Dabei ist A die gegebene Hessenberg-Matrix, und $(x_1, x_2, \ldots, x_{n-1}, 1)$ ist ein Hilfsvektor.

Eine einfache Rechnung zeigt nun, daß die $n-1$ unbekannten Koeffizienten des Vektors $\mathbf{x}$ mittels

$$x_n = 1$$

$$x_i = \frac{1}{a_{i+1,i}} \left[\lambda\, x_{i+1} - \sum_{l=0}^{n-i-1} a_{i+1,n-l} x_{n-l} \right] \qquad i = n-1, n-2, \ldots, 2, 1 \quad (7.46)$$

ausgewertet werden können. Aus der ersten Gleichung von (7.45) folgt nun

$$H(\lambda) = \sum_{i=1}^{n} a_{1,i} x_i - \lambda x_1 \quad . \qquad (7.47)$$

Diese Art und Weise der Berechnung der Funktionswerte von $H(\lambda)$ kann sehr effizient mit einem leistungsfähigen numerischen Nullstellen-Suchprogramm (z.B. mit INTSCH, s. Kapitel 5) kombiniert werden.

Die Funktion HYMAN

INPUT-PARAMETER:

UHM: reelle Upper-Hessenberg-Matrix

N: Zeilen und Spalten der Matrix

LAM: Funktionsargument λ

Struktogramm 25 — FUNCTION HYMAN(UHM,N,LAM)

```
X(N):=1.0
I=N-1(-1)1
    SUM:= LAM*X(I+1)
    L=0(1)N-I-1
        SUM:=SUM - UHM(I+1,N-L)*X(N-L)
    X(I):= SUM/UHM(I+1,I)
SUM:=-LAM*X(1)
I=1(1)N
    SUM:=SUM + UHM(1,I)*X(I)
HYMAN:=SUM
(return)
```

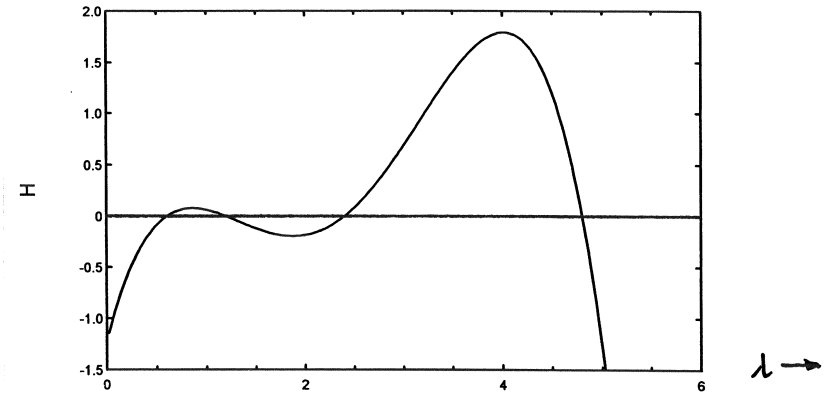


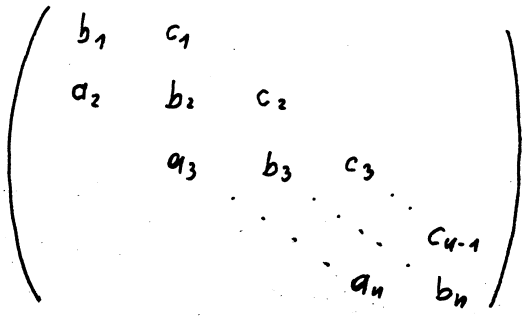
Abbildung 7.2: Die Hyman-Funktion $H(\lambda)$ der 4x4-Matrix von Seite 207.

Ein Testbeispiel

Wenn man die gegebene Matrix mittels ELMHES auf die UHF bringt, und diese Matrix als Input von HYMAN nimmt, so erhält man schnell und genau die Funktionswerte von $H(\lambda)$. Die Nullstellen dieser Kurve sind bekanntlich die (reellen) Eigenwerte der gegebenen Matrix. Um diese Nullstellen numerisch zu ermitteln, müssen also die Programme ELMHES, HYMAN und (z. B.) INTSCH geeignet kombiniert werden.
Die Abb. 7.2 zeigt die Funktion $H(\lambda)$ für die 4x4-Matrix von Seite 207. Die vier Eigenwerte lauten: 0.6, 1.2, 2.4 und 4.8.

7.5.5 Eigenwerte tridiagonaler Matrizen.

Die Hyman'sche Methode kann besonders einfach auf tridiagonale Matrizen der Form



angewendet werden, also auf Matrizen, die durch die drei Diagonalvektoren **a**, **b** und **c** bestimmt werden. Man erhält durch Spezialisierung von (7.46) und (7.47) die Formeln

$$x_n = 1$$

$$x_{n-1} = -\frac{1}{a_n}(b_n - \lambda)$$

$$x_{n-2} = -\frac{1}{a_{n-1}}\left(c_{n-1} - \lambda x_{n-1} + b_{n-1}x_{n-1}\right)$$

$$x_{i-1} = -\frac{1}{a_i}\left((b_i - \lambda)x_i + c_i x_{i+1}\right) \qquad i = n-2, n-1, \ldots, 3, 2$$

$$H(\lambda) = (b_1 - \lambda)\, x_1 + c_1 x_2\,. \tag{7.48}$$

Es folgt nun ein Programmiervorschlag (in C) für die numerische Bestimmung der reellen Eigenwerte allgemeiner reeller tridiagonaler Matrizen:

Programm-Struktur C

```
.
#include <stdio.h>
#include <math.h>
#include "nrutil.c"

int n;
double *a,*b,*c;


double hymtri(double lam)
// Diese Routine dient zur Auswertung der 'Hyman-Funktion' fuer
// tridiagonale Matrizen gemaess Glg. (7.43).
// Die Groesse der Matrix (n) und die Vektoren a, b und c sind
// global definiert.
{
  int i;
  double x1,x2,x;

  if(n <= 2) {
    printf("Grad zu klein\n");
    return 0.0;
  }

  else {
    x2=-(b[n]-lam)/a[n];
    x1=-((b[n-1]-lam)*x2 + c[n-1])/a[n-1];
    for(i=n-2;i>1;i--){
      x= -((b[i]-lam)*x1 + c[i]*x2)/a[i];
      x2=x1;
      x1=x;
    }
    return (b[1]-lam)*x1 + c[1]*x2;
  }
}


#include "intsch.c"
```

```
/********************** main program ******************/
void main()
{
  int anzmax=100;
  double *nullst;
   .
   .
  n=....;      //Groesse der Matrix

  a=dvector(1,n);
  b=dvector(1,n);
  c=dvector(1,n);
  nullst=dvector(1,n);

// Einlesen der Vektoren  a  b  c   der tridiagonalen Matrix:
   .
   .

// Eingabe der INTSCH-Parameter:
  anf=....;     // Beginn des Grobsuch-Bereichs
  aend=....;    // Ende des Grobsuch-Bereichs
  h=....;       // Schrittweite im Grobsuch-Bereich
  gen=....;     // rel. Genauigkeit der Eigenwerte


  intsch(&hymtri,anf,aend,h,gen,anzmax,nullst,&anz);

// Ausgabe der Nullstellen = Eigenwerte der tridiag. Matrix:
   .
   .

  free_dvector(a,1,n);
  free_dvector(b,1,n);
  free_dvector(c,1,n);
  free_dvector(nullst,1,n);
}
```

Anmerkung: Das Nullstellen-Suchprogramm INTSCH wird im Abschnitt 5.5.2 genau erklärt. Der im obigen Beispiel angegebene Aufruf von INTSCH enthält jedoch noch einen zusätzlichen Parameter, nämlich den aktuellen Namen der Funktion, deren Nullstellen von INTSCH eruiert werden sollen. Im konkreten Fall ist das die 'Hyman-Funktion' für tridiagonale Matrizen mit dem Namen 'hymtri'.
Eine solche Möglichkeit, auch Namen von Routinen über Parameterlisten

zu übergeben, bieten viele Programmiersprachen (C, F90, Matlab, ...). Im Falle eines C-Programms müßten Sie die Headline von INTSCH wie folgt schreiben:

```
void intsch(double (*fct)(double),double anf, double aend, double h,
            double gen, int anzmax, double nullst[], int *anz)
```

Zusätzlich müßten Sie im Programm INTSCH jeden Aufruf der Funktion anpassen:

```
anstatt  ... fct(x) ...          ... (*fct)(x) ...
```

7.6 Software-Angebot.

EISPACK (in www.netlib.org) stellte viele Jahre den 'De-facto-Standard' in bezug auf Eigenwert-Programme dar[7]. Es bietet Fortran-Unterprogramme zur Berechnung von Eigenwerten und Eigenvektoren für die folgenden Klassen von Matrizen: komplex, Hermitesch komplex, reell, symmetrisch reell, symmetrisch reell mit Bandstruktur, symmetrisch tridiagonal reell, 'special' tridiagonal reell, allgemein reell, allgemein symmetrisch reell. Außerdem sind manche Programme spezialisiert auf Teillösungen (nur Eigenwerte, alle Eigenwerte und ausgewählte Eigenvektoren, einige Eigenwerte, einige Eigenwerte und zugehörige Eigenvektoren).
1992 ist EISPACK zusammen mit LINPACK zu dem Programmsystem LAPACK zusammengefaßt worden. Einige Anmerkungen zu LAPACK s. Abschnitt 2.8 dieses Skriptums.

Numerical Recipes: Die folgenden einschlägigen Programme in F77, F90 und C in den Büchern sowie in

```
   /usr/local/numeric/numrec/c  (oder fortran-77 oder fortran-90);

                          reelle Matrix    C/F77/F90
-------------------------------------------------------
Jacobi-Verfahren              symm.         jacobi

Householder                   symm.         tred2

QL                          symm.,tridiag.  tqli

Balancing                     general       balanc

Reduction to Hessenberg       general       elmhes

Eigenvalues                   Hessenb.      hqr

(Anmerkung: die Programme 'tred2', 'tql1', 'balanc',
 'elmhes' und 'hqr' entstammen der EISPACK-Bibliothek)
-------------------------------------------------------
```

Numerical Algorithms with C or FORTRAN: In diesem Werk [6] sind zwei Programme zur Lösung von Eigenwertproblemen allgemeiner reeller Matrizen enthalten:

```
Mises-Verfahren (betragsgroesster Eigenwert und -vektor:
       fmises.c   eigval.f90   eigval.for
```

[7]Smith et al, *EISPACK Guide*, in: *Lecture Notes in Computational Science* (G. Goos, Ed.), Springer Berlin 1974.

```
Verfahren von Martin, Parlett, Peters, Reinsch, Wilkinson
                (alle Eigenwerte und -vektoren, QR-Methode):
       feigen.c   eigen.f90    eigen.for

/usr/local/numeric/num_alg/c/ansic/ansic/07  ('07' bedeutet: Kap. 7)
/usr/local/numeric/num_alg/f77/f77/kap07
/usr/local/numeric/num_alg/f77/f90/kap07
```

MATLAB.6: Alles Wissenswerte über die Eigenwert-Routinen von Matlab können Sie online erfahren, wenn Sie den folgenden Weg gehen:

```
MATLAB Help -- MATLAB Function Listed by Category -- Mathematics --
Linear Algebra -- Eigenvalues and Singular Values
```

Sie erhalten dann die folgende Liste von Funktionen:

Eigenvalues and Singular Values

balance	Improve accuracy of computed eigenvalues
cdf2rdf	Convert complex diagonal form to real block diagonal form
condeig	Condition number with respect to eigenvalues
eig	Eigenvalues and eigenvectors
eigs	Eigenvalues and eigenvectors of sparse matrix
gsvd	Generalized singular value decomposition
hess	Hessenberg form of matrix
poly	Polynomial with specified roots
polyeig	Polynomial eigenvalue problem
qz	QZ factorization for generalized eigenvalues
rsf2csf	Convert real Schur form to complex Schur form
schur	Schur decomposition
svd	Singular value decomposition
svds	Singular values and vectors of sparse matrix

Alle in dieser Liste enthaltenen Funktionen verwenden LAPACK-Programme! Durch Anklicken der entsprechenden Namen (besonders wichtig sind die Funktionen *eig* und *eigs*) erhalten Sie genaue Informationen, insbesondere über die Parameterliste.

NAG: Auch in dieser Library gibt es zahlreiche einschlägige Programme, wie der folgende *decision tree* zeigt: