MATRIZEN

Version 10. (c) Herbert Paukert**

In dieser Einführung werden die wichtigsten theoretischen Konzepte des Rechnens mit Matrizen kurz dargestellt. Dieser Text stammt aus dem Rechenprogramm MATRIZEN.EXE des Autors, das von www.paukert.at heruntergeladen werden kann. Im Anhang ab Seite 14 findet man den Programmcode für die wichtigsten Matrizen-Operationen in der Programmiersprache DELPHI.

(1) MATRIZEN

Eine Matrix A ist eine Anordnung von Elementen a(i,j) in n Zeilen und m Spalten. Dabei ist i der Zeilenindex (1 <= i <= n) und j ist der Spaltenindex (1 <= j <= m).

```
a(1,1) a(1,2) . . . a(1,m)
a(2,1) a(2,2) . . . a(2,m)
. . . . . . . . . . . . .
. . . . . . . . . . . . .
a(n,1) a(n,2) . . . a(n,m)
```

Matrizen mit n Zeilen und n Spalten heißen quadratisch. Matrizen mit n Zeilen und 1 Spalte, oder 1 Zeile und n Spalten heißen Vektoren. Sie werden klein geschrieben.

Beispiel eine Vektors x´ = [a(1) a(2) . . . a(n)]

i-ter Zeilenvektor der Matrix: [a(i,1) a(i,2) . . .]
j-ter Spaltenvektor der Matrix: [a(1,j) a(2,j) . . .]

Hauptdiagonale: a(1,1) a(2,2) . . . a(n,m)
Nebendiagonale: a(1,m) a(2,m-1) . . a(n,1)

Einheitsmatrix E: a(i,j) = 1 für alle i = j
**   a(i,j) = 0 für alle i <> j**
Nullmatrix O: a(i,j) = 0 für alle i und alle j

Eine Dreiecksmatrix ist eine quadratische Matrix, deren Elemente unter- oder oberhalb der Hauptdiagonale alle Null sind. Eine Diagonalmatrix ist eine Matrix, bei der alle Elemente außerhalb der Hauptdiagonale Null sind.

Matrizen-Addition C = A + B: c(i,j) = a(i,j) + b(i,j).
Multiplikation mit Zahl ß, C = ß*A: c(i,j) = ß*a(i,j).

(2) VEKTOREN

Gegeben sind zwei Vektoren x und y.

```
x´ = [x(1) x(2) . . . x(n)]
y´ = [y(1) y(2) . . . y(n)]
```

Für das Skalarprodukt zweier Vektoren s = x´ * y´ gilt:
s = x(1)*y(1) + x(2)*y(2) + . . . + x(n)*y(n)

Zwei Vektoren heißen orthogonal, wenn ihr Skalarprodukt Null ergibt.

Für den Betrag (Länge) |x´| eines Vektors x´ gilt:
|x´| = sqrt(x(1)² + x(2)² + ... + x(n)²)
Ein Vektor x´ heißt normiert, wenn |x´| = 1.

Gegeben ist eine Menge von Vektoren: w´, x´, y´, z´.
Vektor w´ ist eine Linearkombination (Vielfachensumme) der Vektoren x´, y´, z´, wenn es reelle Zahlen a, b, c gibt, sodass gilt: w´ = a*x´ + b*y´ + c*z´.

Vektoren heißen voneinander linear abhängig (l.a.), wenn mindestens einer von ihnen eine Linearkombination der restlichen anderen Vektoren ist.
Vektoren heißen voneinander linear unabhängig (l.u.a.), wenn keiner von ihnen eine Linearkombination der restlichen anderen Vektoren ist.

---

*(3) DAS PRODUKT VON MATRIZEN C = A * B*

A ist eine (n,m)-Matrix und B ist eine (m,k)-Matrix:

```
a(1,1) a(1,2) . . . a(1,m)
a(2,1) a(2,2) . . . a(2,m)
. . . . . . . . . . . . .
. . . . . . . . . . . . .
a(i,1) a(i,2) . . . a(i,m)
. . . . . . . . . . . . .
. . . . . . . . . . . . .
a(n,1) a(n,2) . . . a(n,m)


b(1,1) b(1,2) . . b(1,j). . b(1,k)
b(2,1) b(2,2) . . b(2,j). . b(2,k)
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
b(m,1) b(m,2) . . b(m,j). . b(m,k)
```

Das Matrizenprodukt C = A * B ist eine (n,k)-Matrix in der jedes Element c(i,j) das Skalarprodukt des i-ten Zeilenvektors von A mit dem j-ten Spaltenvektor von B ist:

c(i,j) = a(i,1)*b(1,j)+a(i,2)*b(2,j)+...+a(i,m)*b(m,j)

Ein wichtiger Spezialfall ist die Multiplikation einer quadratischen Matrix A mit einem Vektor x´. Das ergibt dann einen neuen Vektor y´ = A * x´.

```
a(1,1) a(1,2) . . . a(1,n)
a(2,1) a(2,2) . . . a(2,n)
. . . . . . . . . . . . .
. . . . . . . . . . . . .
a(i,1) a(i,2) . . . a(i,n)
. . . . . . . . . . . . .
. . . . . . . . . . . . .
a(n,1) a(n,2) . . . a(n,n)
```

Vektor x´ = [x(1) x(2) . . . x(n)]

y´ = A * x´ = [y(1) y(2) . . y(n)] mit folgenden Werten:
y(i) = a(i,1)*x(1) + a(i,2)*x(2) + . . . + a(i,n)*x(n)

Für die Matrizen-Multiplikation gelten folgende Regeln:

A * E = E * A = A (mit Einheitsmatrix E)
A * B <> B * A (antikommunikativ)
(A*B)*C = A*(B*C) (assoziativ)
A*(B+C) = A*B + A*C (distributiv)
(B+C)*A = B*A + C*A (distributiv)

Vertauscht man in einer Matrix A ihre Zeilen mit ihren Spalten, dann erhält man die transponierte Matrix At. Die Transponierte eines Produktes: (A * B)t = Bt * At. Eine Matrix heißt symmetrisch, wenn A = At.

(4) DER RANG EINER MATRIX, rank(A)

Unter dem Rang einer Matrix "rank(A)" versteht man die maximale Anzahl von linear unabhängigen Zeilenvektoren der Matrix A. Zeilenrang und Spaltenrang stimmen immer überein. Also ist rank(A) = rank(At).

Rang einer quadratischen (n,n)-Matrix kann maximal n sein. Eine (n,n)-Matrix mit Rang n heißt regulär. Andernfalls heißt sie singulär, wenn rank(A) < n.

(5) DIE DETERMINANTE EINER MATRIX, det(A)

Wenn man in einer Matrix A die i-te Zeile und auch die j-te Spalte streicht, dann erhält man die so genannte Streichungsmatrix A|ij.

Für die Determinante det(A) einer (n,n)-Matrix gilt:

det(A) = Summe((-1)^(i+j) * a(i,j) * det(A|ij))
für laufenden Zeilenindex i von 1 bis n,
für einen beliebigen festen Spaltenindex j.

Beispielsweise mit Spalte j = 1 gilt dann: det(A) = a(1,1)*det(A|11)-a(2,1)*det(A|21)+a(3,1)*det(A|31) .....

Anstatt die Determinante nach den Zeilen zu entwickeln, kann sie auch nach den Spalten entwickelt werden. Das Ergebnis ist immer gleich (Entwicklungssatz von Laplace).

Beispiel einer (2,2)-Determinante:

a11 a12
a21 a22

det(A) = a11*a22 - a21*a12

Geometrisch ist det(A) die Fläche des Parallelogramms in der Ebene, das von den zwei Zeilenvektoren x´= [a11 a12] und y´ = [a21 a22] aufgespannt wird.

Beispiel einer (3,3)-Determinante:

```
a11 a12 a13
a21 a22 a23
a31 a32 a33

det(A) = a11*det(A|11) - a21*det(A|21) + a31*det(A|31)

det(A) = a11 * (a22*a33 - a32*a23) -
         a21 * (a12*a33 - a32*a13) +
         a31 * (a12*a23 - a22*a13)

       = a11*a22*a33 + a21*a32*a13 + a31*a12*a23
         -(a11*a32*a23 + a21*a12*a33 + a31*a22*a13)
```

Geometrisch ist det(A) das Volumen des schiefen Quaders im Raum (Parallelepipeds), der von den 3 Zeilenvektoren aufgespannt wird. Dieses Volumen heißt auch Spatprodukt.

Es gelten folgende vier wichtige Determinantenregeln, welche sich direkt aus der Definition ergeben:

[D1] Vertauscht man zwei Zeilen (Spalten) der Matrix, dann ändert sich nur das Vorzeichen, aber nicht der Betrag der Determinante.

[D2] Addiert man zu einer Zeile (Spalte) der Matrix ein Vielfaches einer anderen Zeile (Spalte), dann ändert sich die Determinante nicht.

[D3] Die Determinante einer Dreiecksmatrix oder einer Diagonalmatrix ist das Produkt der Elemente der Hauptdiagonale.

[D4] Sind die Zeilen (Spalten) einer (n,n)-Matrix A linear abhängig (rank(A) < n, A singulär), dann ist die Determinante gleich Null.

(6) DIE INVERSE EINER MATRIX (Ai)

Die inverse Matrix Ai zu einer (n,n)-Matrix A ist eine Matrix mit folgender Eigenschaft:

A * Ai = Ai * A = E (E = Einheitsmatrix)

Nur zu quadratischen, nicht singulären Matrizen gibt es eindeutig bestimmte inverse Matrizen.

Für die inverse Matrix eines Produktes gilt:

```
(A * B)i * (A * B) = E
(A * B)i * A = E * Bi
(A * B)i = E * Bi * Ai

(A * B)i = Bi * Ai
```

Die Inversion einer Matrix spielt eine wesentliche Rolle bei der Auflösung von linearen Gleichungssystemen, usw.

Die Ermittlung der inversen Matrix erfolgt nach einem Verfahren von Gauß & Jordan. Dieser Algorithmus wird weiter unten an Hand eines einfachen Beispiels erklärt.

(7) LINEARE GLEICHUNGSSYSTEME

Beispiel einer einfachen linearen Gleichung in einer Variablen x: 2*x - 3 = 5

Um die Gleichung zu lösen, addieren wir zuerst auf beiden Gleichungsseiten die Zahl 3 und dividieren dann beide Gleichungsseiten durch 2. So erhalten wir die Lösung x = 4.

Umformungen, welche die Lösungsmenge einer Gleichung nicht verändern, heißen Äquivalenz-Umformungen. Das ist immer dann der Fall, wenn man auf beiden Seiten der Gleichung dieselben Rechenoperationen durchführt.

Ein lineares Gleichungssystem in N Variablen:

```
a11 * x1  + a12 * x2 + . . . + a1N * xN  =  b1
a21 * x1  + a22 * x2 + . . . + a2N * xN  =  b2
. . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . .
aN1 * x1  + aN2 * x2 + . . . + aNN * xN  =  bN
```

Die Systemmatrix A ist quadratisch:

```
a11 a12 . . . a1N
a11 a12 . . . a1N
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
aN1 aN2 . . . aNN
```

Die erweiterte Matrix Ae ist eine (N,N+1)-Matrix:

```
a11 a12 . . . a1N  b1
a11 a12 . . . a1N  b2
. . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
aN1 aN2 . . . aNN  bN
```

Vektor b´ = [b1 b2 b3 . . . bN]
Vektor x´ = [x1 x2 x3 . . . xN]

Das lineare Gleichungssystem in Matrix-Schreibweise: A * x´ = b´. Falls die Systemmatrix A regulär ist, dann gibt es dazu eine inverse Matrix Ai und es gilt: A * x´ = b´, Ai * A * x´ = Ai * b´, E * x´ = Ai * b´. Somit gilt: x´ = Ai * b´.

Der Lösungsvektor x´ wird mit dem Eliminationsverfahren von Gauß ermittelt. Dabei werden entsprechende Vielfache von geeigneten Zeilenvektoren Z(i) der Matrix Ae von den restlichen Zeilenvektoren so lange subtrahiert bis die Systemmatrix A eine Dreiecksform hat. Weil es sich um Äquivalenz-Umformungen handelt, wird die Lösung dabei nicht verändert. Beginnend mit der letzten Zeile können dann die einzelnen Koeffizienten des Lösungsvektors x schrittweise durch Rückwärtseinsetzen berechnet werden.

Beispiel mit allen Eliminationsschritten.

Zuerst mit dem Gauß-Verfahren bis zur Dreiecksmatrix und dann weiter mit dem Gauß-Jordan-Verfahren bis zur Diagonalmatrix.

Alle Zeilen-Hinweise rechts beziehen sich auf die jeweils vorangehende Matrix.

```
3*x1 + 5*x2 + 1*x3 = 14
1*x1 + 2*x2 + 2*x3 = 10
2*x1 + 4*x2 + 5*x3 = 23

3*x1 +   5*x2 +    1*x3 =   14, Z(1) bleibt unverändert
   0 + 1/3*x2 +  5/3*x3 = 16/3, Z(2) = Z(2) - (1/3)*Z(1)
   0 + 2/3*x2 + 13/3*x3 = 41/3, Z(3) = Z(2) - (2/3)*Z(1)

3*x1 + 5*x2 +  1*x3 = 14,  Z(1) bleibt unverändert
   0 + 1*x2 +  5*x3 = 16,  Z(2) = Z(2) * 3
   0 + 2*x2 + 13*x3 = 41,  Z(3) = Z(3) * 3

3*x1 + 5*x2 +  1*x3 = 14,  Z(1) bleibt unverändert
   0 + 1*x2 +  5*x3 = 16,  Z(2) bleibt unverändert
   0 +    0 +  3*x3 =  9,  Z(3) = Z(3) - (2/1)*Z(2)
```

Durch diese Umformungen ist eine Systemmatrix A in Dreiecksform entstanden. Aus der letzten Gleichung folgt x3 = 3. Durch Einsetzen in die zweite Gleichung erhält man x2 = 1. Einsetzen in die erste Gleichung ergibt dann x1 = 2. Damit ist das Eliminationsverfahren nach Gauß beendet.

Beim Verfahren von Gauß-Jordan wird der Eliminationsprozess nicht nur auf alle, der Eliminationszeile Z(i) NACHFOLGENDEN Zeilen angewendet, sondern auf ALLE Gleichungszeilen der Matrix, ausgenommen der Zeile Z(i).

Nach dem Erreichen der Dreiecksform werden die Umformungen so lange fortgesetzt bis schließlich die Systemmatrix eine Diagonalform aufweist, mit lauter 1 in der Hauptdiagonale (Einheitsmatrix).

```
3*x1 +    0 - 24*x3 = -66,  Z(1) = Z(1) - (5/1)*Z(2)
   0 + 1*x2 +  5*x3 =  16,  Z(2) bleibt unverändert
   0 +    0 +  1*x3 =   3,  Z(3) = Z(3)/3

1*x1 +    0 -  8*x3 = -22,  Z(1) = Z(1)/3
   0 + 1*x2 +  5*x3 =  16,  Z(2) bleibt unverändert
   0 +    0 +  1*x3 =   3,  Z(3) bleibt unverändert
```

```
1*x1 +    0 -    0 =    2,  Z(1) = Z(1) - (-8/1)*Z(3)
   0 + 1*x2 +    0 =    1,  Z(2) = Z(2) - (5/1)*Z(3)
   0 +    0 + 1*x3 =    3,  Z(3) bleibt unverändert
```

Nach den letzten Äquivalenz-Umformungen sind die Lösungen aus der Diagonalform der Matrix direkt ablesbar. Damit ist das Eliminationsverfahren von Gauß-Jordan beendet.

(8) BERECHNUNG DER INVERSEN MATRIX

Ai ist die inverse Matrix zur quadratischen Matrix A.

Es gilt A * Ai = E.

Wir verwandeln die Matrix A mit dem Eliminations-verfahren von Gauß-Jordan in die Einheitsmatrix E. Alle dabei ausgeführten Umformungen wenden wir nun auch auf E an. Dadurch entsteht eine neue Matrix B. Diese Matrix B ist die gesuchte inverse Matrix Ai, weil A * B = E ist, und daher B = Ai sein muss.

Zur Demonstration verwenden wir die Systemmatrix A des ersten Beispiels und schreiben daneben auch die entsprechend umgeformte Einheitsmatrix E.

Alle Zeilen-Hinweise rechts beziehen sich auf die je-weils vorangehende Matrix.

```
3  5  1     |    1   0   0
1  2  2     |    0   1   0
2  4  5     |    0   0   1

3   5    1  |    1   0   0,  Z(1) bleibt unverändert
0 1/3  5/3  | -1/3   1   0,  Z(2) = Z(2) - (1/3)*Z(1)
0 2/3 13/3  | -2/3   0   1,  Z(3) = Z(3) - (2/3)*Z(1)

3  5  1     |    1   0   0,  Z(1) bleibt unverändert
0  1  5     |   -1   3   0,  Z(2) = Z(2) * 3
0  2 13     |   -2   0   3,  Z(3) = Z(3) * 3

3  5  1     |    1   0   0,  Z(1) bleibt unverändert
0  1  5     |   -1   3   0,  Z(2) bleibt unverändert
0  0  3     |    0  -6   3,  Z(3) = Z(3) - (2/1)*Z(2)

3  0 -24    |    6 -15   0,  Z(1) = Z(1) - (5/1)*Z(2)
0  1   5    |   -1   3   0,  Z(2) bleibt unverändert
0  0   1    |    0  -2   1,  Z(3) = Z(3)/3

1  0 -8     |    2  -5   0,  Z(1) = Z(1)/3
0  1  5     |   -1   3   0,  Z(2) bleibt unverändert
0  0  1     |    0  -2   1,  Z(3) bleibt unverändert

1  0  0     |    2 -21   8,  Z(1) = Z(1) - (-8/1)*Z(3)
0  1  0     |   -1  13  -5,  Z(2) = Z(2) - (5/1)*Z(3)
0  0  1     |    0  -2   1,  Z(3) bleibt unverändert
```

Damit gilt für die inverse Matrix Ai von Matrix A:

```
 2  -21   8
-1   13  -5
 0   -2   1
```

Zur Überprüfung kann noch A * Ai = E gerechnet werden.

Gleichungen mit einer unbekannten Matrix heißt Matrizen-Gleichungen: A * X = B. Wenn die Matrizen quadratisch und invertierbar sind, dann gilt X = Ai * B, wobei Ai die inverse Matrix zu A ist, die von links multipliziert wird.
Beispiel: A + B*X = C*X, Lösung: X = (C-B)i * A

(9) WEITERE BEISPIELE ZUM ÜBEN

Zwischen den Elementen der Matrizen wird im Folgenden immer der Strichpunkt als Separator-Zeichen geschrieben.

BSP 1: A * x´ = b´, x´ = Ai * b´

```
 3;  2;  6;  1
 1;  1;  3;  0
-3; -2; -5; -2
```

Lösungen: x1 = 1, x2 = 2, x3 = -1
Inverse Systemmatrix Ai:

```
 1; -2;  0
-4;  3; -3
 1;  0;  1
```

BSP 2: A * x´ = b´, x´ = Ai * b´

```
 1;  1;  1;  0
 4;  2;  1;  1
 9;  3;  1;  3
```

Lösungen: x1 = 0.5, x2 = -0.5, x3 = 0
Inverse Systemmatrix Ai:

```
 0.50; -1.00;  0.50
-2.50;  4.00; -1.50
 3.00; -3.00;  1.00
```

BSP 3: A * x´ = b´, x´ = Ai * b´

```
 2; -2; -1;  2
 1;  3; -2;  4
 1; -3;  1;  2
```

```
Lösungen: x1 = 4.67, x2 = 2.00, x3 = 3.33
Inverse Systemmatrix Ai:

-0.50;  0.83;  1.17
-0.50;  0.50;  0.50
-1.00;  0.67;  1.33

--------------------------------
BSP 4:  A * x´ = b´, x´ = Ai * b´
--------------------------------

 3;  2;  1;  7
 0;  1;  2;  8
 3;  1;  1;  3

Lösungen: x1 = -1, x2 = 4, x3 = 2
Inverse Systemmatrix Ai:

-0.17; -0.17;  0.50
 1.00;  0.00; -1.00
-0.50;  0.50;  0.50


BEISPIEL einer QUADRATISCHEN REGRESSION: 5 Messpaare (x;y)
sind gegeben: (x1;y1), (x2;y2), (x3;y3), (x4;y4), (x5;y5).
Gesucht ist eine Polynomfunktion f(x) = a0 + a1*x + a2*x^2,
welche sich optimal an die Messpunkte anpasst. Zur Lösung
wird die Methode der kleinsten Fehlerquadrate angewendet.

Diese Aufgabe wird mit Hilfe der Matrizenrechnung gelöst.
Im Menü <Bearbeiten> sind zur Ermittlung der inversen Matrix
acht Nachkommastellen zu wählen. Es gilt folgende Theorie:

1;  x1;  x1^2    Die links stehende Systemmatrix sei M.
1;  x2;  x2^2    (y1; y2; y3; y4; y5) sei der Vektor y´.
1;  x3;  x3^2    (a0; a1; a2) sei der Vektor a´. Es gilt
1;  x4;  x4^2    dann: M * a´ = y´. Das ist ein lineares
1;  x5;  x5^2    Gleichungssystem mit 5 Gleichungen und
                 den 3 Unbekannten a0, a1, a2. Das System
                 ist daher überbestimmt.

Weiters gelten folgende Umformungen: Mt * M * a´ = Mt * y´
und a´ = (Mt * M)i * Mt * y´. Dabei ist Mt die Transponierte
von M, und (Mt * M)i ist die Inverse Matrix von (Mt * M).
Die Umformungen sind sinnvoll, weil (Mt * M) eine quadratische
Matrix ist. Die Umformungen liefern das sog. Normalsystem NS.

Beispiel: 5 Messpaare: (-1;14), (0;5), (1;4), (2;-1), (8;26)

System-Matrix: (5,3)-Matrix M

1; -1;  1
1;  0;  0
1;  1;  1
1;  2;  4
1;  8; 64

Vektor y´= (14; 5; 4; -1; 26)
```

Transponierte Matrix: (3,5)-Matrix Mt

```
 1; 1; 1; 1;  1
-1; 0; 1; 2;  8
 1; 0; 1; 4; 64
```

Matrizenprodukt: (3,3)-Matrix Mt * M

```
 5;  10;   70
10;  70;  520
70; 520; 4114
```

Inverse Matrix (Mt * M)i:

```
 0.28585366; -0.07707317;  0.00487805
-0.07707317;  0.25479675; -0.03089431
 0.00487805; -0.03089431;  0.00406504
```

Matrizenprodukt (Mt * M)i * Mt:

```
 0.36780488;  0.28585366;  0.21365854;  0.15121952; -0.01853650
-0.36276423; -0.07707317;  0.14682927;  0.30894309; -0.01593501
 0.03983740;  0.00487805; -0.02195122; -0.04065041;  0.01788613
```

Lösungsvektor a´ = (a0; a1; a2) = (Mt * M)i * Mt * y´
6.80000000; -5.60000000; 1.00000000

Regressionsfunktion f(x): f(x) = 6.8 - 5.6 * x + x^2

---

(10) QR-Zerlegung einer Matrix

Für jede Matrix A gibt es eine so genannte QR-Zerlegung

A = Q * R

Dabei ist Q eine orthogonale Matrix und R eine obere Dreiecksmatrix. In einer orthogonalen Matrix Q sind alle Spaltenvektoren paarweise orthogonal, und zusätzlich sind sie noch normiert. Daher gilt: Q * Qt = E (Einheitsmatrix), d.h. die transponierte Matrix ist auch die inverse Matrix. In der oberen Dreiecksmatrix R sind alle Matrixelemente unterhalb der Hauptdiagonale gleich Null. Die QR-Zerlegung kann mit Hilfe verschiedener Verfahren durchgeführt werden, beispielsweise mit dem GIVENS-Verfahren.

Mit Hilfe der QR-Zerlegung können lineare Gleichungssysteme folgendermaßen gelöst werden.

Beispiel:
3*x1 + 5*x2 + 1*x3 = 14
1*x1 + 2*x2 + 2*x3 = 10
2*x1 + 4*x2 + 5*x3 = 23

Systemmatrix A:

```
3;  5;  1
1;  2;  2
2;  4;  5
```

Vektor x´= [x1; x2; x3]
Vektor b´= [14; 10; 23]

Das lineare Gleichungssystem A * x´ = b´ wird zuerst zum Normalsystem NS umgeformt, weil dadurch die inverse Matrix von einer quadratischen Matrix (At * A) gebildet werden kann. Das gilt auch, wenn die Matrix A selbst keine quadratische Matrix ist, wie beispielsweise bei den Regressionsaufgaben.

A * x´ = b´
At * A * x´ = At * b´
x´ = (At * A)i * At * b´ (Normalsystem NS)

A = Q * R (QR-Zerlegung von A)
At = Rt * Qt
x´ = (At * A)i * At * y´
x´ = (Rt * Qt * Q * R)i * Rt * Qt * y´
x´ = (Rt * E * R)i * Rt * Qt * y´
x´ = (Rt * R)i * Rt * Qt * y´
x´ = Ri * (Rt)i * Rt * Qt * y´
x´ = Ri * E * Qt * y´
x´ = Ri * Qt * y´

Die Inverse Ri zur oberen Dreiecksmatrix R wird mit dem Gauß-Jordan-Verfahren ohne einen Zeilentausch berechnet, weil alle Elemente in der Diagonale von R nicht Null sind.

Das Beispiel kann als Übung mit dem Gauß-Verfahren, mit dem Normalsystem NS und mit der QR-Zerlegung durchgerechnet werden. Als Lösungsvektor x´ erhält man in allen Fällen x´ = [2; 1; 3]. Mit Hilfe der QR-Zerlegung der Matrix A kann auch die Inverse Ai berechnet werden: Ai = Ri * Qt, weil Qi = Qt.

---

(11) Eigenwerte und Eigenvektoren

Es sei A eine quadratische Matrix. Die Zahl k und ein dazugehöriger Vektor x´ heißen Eigenwert und Eigenvektor der Matrix, wenn folgende Gleichung gilt:

A * x´ = k*x´

Die Multiplikation des Eigenvektors x´ mit der Matrix A ergibt das k-Fache des Eigenvektors. Falls die Matrix symmetrisch ist, sind alle Eigenwerte k reelle Zahlen.

A * x´ = k*x´
A * x´ - k*x´ = 0´ (Nullvektor 0´)
(A - k*E) * x´ = 0´ (Einheitsmatrix E)

```
a(1,1)-k  a(1,2)    a(1,3)  . . . . . a(1,n)
a(2,1)    a(2,2)-k  a(2,3)  . . . . . a(2,n)
a(3,1)    a(3,2)    a(3,3)-k  . . . . a(3,n)
. . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . .
a(n,1)    a(n,2)    a(n,3)  . . . . . a(n,n)-k
```

Falls es zu dieser Matrix (A - k*E) eine inverse Matrix gibt, dann gilt:

(A - k*E) * x´ = 0´
x´ = (A - k*E)i * 0´ = 0´

Der Eigenvektor x´ ist hier der Nullvektor. Das ist aber sinnlos. Daher darf (A - k*E) nicht invertierbar sein. Dann ist diese Matrix nicht regulär, d.h. ihre Zeilenvektoren sind linear abhängig und ihre Determinante ist gleich Null, det(A - k*E) = 0.

Rechnet man die Determinante der Matrix aus, so erhält man ein charakteristisches Polynom n-ten Grades in den Eigenwerten k, welches den Wert Null annimmt. Pn(k) = 0. Somit sind die Eigenwerte die Nullstellen des charakteristischen Polynoms.

Die zu den ermittelten Eigenwerten k dazugehörigen Eigenvektoren x´ erhält man, wenn man das lineare Gleichungssystem (A - k*E) * x´ = 0´ auflöst. Das ergibt dann den Lösungsvektor x´ = [x(1); x(2); x(3); . . . . . x(n)].

Beispiel: (2*2)-Matrix A

1; -2
1; 4

1-k; -2
1;   4-k

det(A - k*E) = P2(k) = (1-k)*(4-k) - 1*(-2) = 0

Das liefert die quadratische Gleichung: k^2 - 5*k + 6 = 0.
Die zwei Lösungen davon sind: k1 = 2 und k2 = 3.

Den ersten Eigenvektor x´ = [x(1), x(2)] zum Eigenwert k = 2 erhält man durch Einsetzen in (A - 2*E) * x´ = 0´. Das ergibt ein lineares Gleichungssystem mit linear abhängigen Zeilen:

-1*x(1) - 2*x(2) = 0
1*x(1) + 2*x(2) = 0

x(1) = -2*x(2). Wir wählen für x(2) = 1. Dann ist x(1) = -2.
Zum Eigenwert 2 existiert ein Eigenvektor x´ = [-2; 1].

In analoger Weise findet man den zweiten Eigenvektor y´ zum Eigenwert k = 3: (A - 3*E) * y´ = 0´.

-2*y(1) - 2*y(2) = 0
1*y(1) + 1*y(2) = 0

y(1) = -1*y(2). Wir wählen für y(2) = 1. Dann ist y(1) = -1.
Zum Eigenwert 3 existiert ein Eigenvektor y´ = [-1; 1].

Mittels Menüeintrag <Eigenwerte (2,3)> können die Eigenwerte und Eigenvektoren von beliebigen quadratischen Matrizen, deren Zeilenanzahl 2 oder 3 ist, ermittelt werden. Dabei erfolgen die Berechnungen durch Auflösung des charakteristischen Polynoms.

Mittels Menüeintrag <Eigenwerte (n)> werden die Eigenwerte und Eigenvektoren von symmetrischen (n,n)-Matrizen ermittelt. Die Berechnungen erfolgen mit Hilfe des so genannten JACOBI-Näherungsverfahrens. Es werden hier nur symmetrische Matrizen behandelt, weil diese Matrizen in der Praxis wichtig sind.

Beim JACOBI-Verfahren wird die symmetrische Matrix A in eine Diagonalmatrix Ad umgeformt, in deren Diagonale nur mehr die Eigenwerte von A stehen.

Die diagonalisierte Matrix Ad erhält man durch Ad = Ut * A * U. Dabei ist die Matrix U eine orthogonale Matrix, die selbst ein Produkt von orthogonalen Matrizen Uk (Jacobi-Rotationen) ist, welche die Eigenwerte von A nicht ändern: U = U1 * U2 * ... * Uk.

Die Rotationen werden nun so lange ausgeführt bis man eine gute Näherung für die Diagonalmatrix erhält. In der Diagonale dieser Matrix Ad stehen dann die Eigenwerte von A. Das Verfahren liefert auch noch die zugehörigen Eigenvektoren, die dann in den Spalten der Matrix U stehen (bzw. in den Zeilen von Ut).

Beispiel:

Symmetrische (4*4)-Matrix A:

```
 4; 1; -2;  2
 1; 2;  0;  1
-2; 0;  3; -2
 2; 1; -2; -1
```

Diagonal-Matrix Ad:

```
 6.8446;  0.0000;  0.0000; -0.0000
 0.0000;  1.0844; -0.0000;  0.0000
 0.0000;  0.0000;  2.2685;  0.0000
 0.0000;  0.0000;  0.0000; -2.1975
```

Eigenwerte:

```
 6.8446;  1.0844;  2.2685; -2.1975
```

Eigenvektoren (Zeilen):

```
 0.7180;  0.2212; -0.5574; 0.3534
-0.6423;  0.5442; -0.5202; 0.1440
 0.2017;  0.7894;  0.5796; 0.0103
-0.1767; -0.1781;  0.2877; 0.9243
```

Hinweis:

Ausführliche Herleitungen des Gauß-Jordan-Verfahrens, der QR-Zerlegung, des Givens- und des Jacobi-Verfahrens findet man in den einschlägigen Fachbüchern über numerische Mathematik und lineare Algebra.

In dem Programm MATRIZEN.EXE wurde zu den genannten Verfahren ein entsprechender Programmcode in DELPHI erstellt. Diesen findet man im anschließenden Anhang.

Anhang: Programmierung von MATRIZEN

Programmcode von 16 Routinen zum Rechnen mit Matrizen aus dem DELPHI-Programm MATRIZEN.EXE von Herbert Paukert. Homepage: www.paukert.at. Die Arbeitsfläche des Programms ist ein einfacher Texteditor, wo alle Zahlen, Vektoren und Matrizen editiert und mit der Maus markiert werden. Dann wird in einem Menü die entsprechende Matrizen-Rechnung angeklickt. Dadurch erfolgt die Ausführung, und das Rechenergebnis wird in der Arbeitsfläche ausgegeben. Nach neuerlicher Markierung kann weiter-gerechnet werden. So sind Rechnungen mit Matrizen bequem ausführbar. Über das Menü <MATRIZEN> werden folgende elf Operationen erreicht:

[1] Lineare Gleichungssysteme (Gauß-Verfahren)
[2] Matrix-Determinante (Gauß-Verfahren)
[3] Matrix-Transposition
[4] Matrix-Addition
[5] Matrix MAL Zahl
[6] Matrix-Multiplikation
[7] Matrix-Dreiecksform (Gauß-Verfahren)
[8] Matrix-Inversion (QR-Verfahren und Gauß-Jordan-Verfahren)
[9] Matrix-QR-Zerlegung (Givens-Verfahren)
[10] Eigenwerte von (2*2)-Matrizen und (3*3)-Matrizen
[11] Eigenwerte von symmetrischen (n*n)-Matrizen (Jacobi-Verfahren)

```
{
       Sechs globale Matrix-Variable:
       ERROR: Boolean;                          //  allgemeine Fehlervariable
       SEP: String;                             //  Separatorzeichen
       RANK : Integer;                          //  Matrix-Rang
       DETER: Real;                             //  Matrix-Determinante
       KOMPLEX: Boolean;                        //  Komplex oder Reell
       CPOLY: String;                           //  Charakteristisches Polynom

       Alle Matrizen sind vom Typ  TMatrix = array[0..MAX,0..MAX] of Real.
       MAX ist der maximale Index. Derzeit ist MAX = 10. Es sind dabei:
       A[0,0] = Zeilenanzahl n, und A[0,1] = Spaltenanzahl m der Matrix.
       Die Vektoren sind vom Typ  TVektor = array[0..MAX] of Real oder als
       einzeilige bzw. einspaltige Matrizen definiert. Vektoren können
       als Zeilen oder als Spalten geschrieben werden.
}
```

function IDE(n: Integer): TMatrix;

```
// Erzeugt die Einheitsmatrix
var i,j: Integer;
    H: TMatrix;
begin
  H[0,0] := n;
  H[0,1] := n;
  for i := 1 to n do begin
      for j := 1 to n do
          if i = j then H[i,j] := 1
                   else H[i,j] := 0;
  end;
  Result := H;
end;
```

function DUPL(var X: TMatrix): TMatrix;

```
// Kopiert eine Matrix
var l,r,i,j: Integer;
    H: TMatrix;
begin
  l := Round(X[0,0]);
  r := Round(X[0,1]);
  for i := 1 to l do
      for j := 1 to r do H[i,j] := X[i,j];
  H[0,0] := l;
  H[0,1] := r;
  Result := H;
end;
```

function EQU(var X,Y: TMatrix): Boolean;

```
// testet die Gleicheit zweier Matrizen
var n,m,l,r,i,j: Integer;
    H: TMatrix;
begin
  Result := True;
  n := Round(X[0,0]);
  m := Round(X[0,1]);
  l := Round(Y[0,0]);
  r := Round(Y[0,1]);
  if (n <> m) or (l <> r) or (n <> l) then begin
     Result := False;
     Exit;
  end;
  for i := 1 to n do
    for j := 1 to n do
        if X[i,j] <> Y[i,j] then Result := False;
end;
```

function TRANS(var X: TMatrix): TMatrix;

```
// erzeugt eine transponierte Matrix
var l,r,i,j: Integer;
    H: TMatrix;
begin
  l := Round(X[0,0]);
  r := Round(X[0,1]);
  for i := 1 to l do begin
      for j := 1 to r do begin
          H[j,i] := X[i,j];
      end;
  end;
  H[0,0] := r;
  H[0,1] := l;
  Result := H;
end;
```

function ADD(var X,Y: TMatrix; OP: Integer): TMatrix;

```
// Matrizen-Addition, Addition (OP = +1), Subtraktion (OP = -1)
var n,m,l,r,i,j,k: Integer;
    H: TMatrix;
begin
     n := Round(X[0,0]);
     m := Round(X[0,1]);
     l := Round(Y[0,0]);
     r := Round(Y[0,1]);
     Error := False;
     if (n <> l) or (m <> r) then begin
        Error := True;
        Exit;
     end;
     for i := 1 to n do begin
         for j := 1 to m do begin
             H[i,j] := X[i,j] + OP * Y[i,j];
         end;
     end;
     H[0,0] := n;
     H[0,1] := m;
     Result := H;
end;
```

function MATNUM(var X: TMatrix; z: Real): TMatrix;

```
// Matrix MAL Zahl, bei Division wird 1/z statt z übergeben
var n,m,i,j: Integer;
    H: TMatrix;
begin
     n := Round(X[0,0]);
     m := Round(X[0,1]);
     for i := 1 to n do begin
         for j := 1 to m do begin
             H[i,j] := z * X[i,j];
         end;
     end;
     H[0,0] := n;
     H[0,1] := m;
     Result := H;
end;
```

function MULT(var X,Y: TMatrix): TMatrix;

```
// Matrix MAL Matrix:
//(n,m) MAL (l,r) = (n,r)-Matrix, wobei m = l ist
// Matrix MAL Vektor:
//(Vektor = Matrix mit 1 Zeile ODER 1 Spalte)
// Vektor MAL Vektor:
//(Skalarprodukt = Matrix mit 1 Zeile UND 1 Spalte = Reelle Zahl)
var i,j,k,n,m,l,r: Integer;
    z: Real;
    H: TMatrix;
begin
  n := Round(X[0,0]);
  m := Round(X[0,1]);
  l := Round(Y[0,0]);
  r := Round(Y[0,1]);
  ERROR := False;
  if (l > 1) and (m <> l) then begin
     ERROR := True;
     Exit;
  end;
  if (l = 1) and (m <> r) then begin
     ERROR := True;
     Exit;
  end;
  if (l = 1) and (m = r) then Y := TRANS(Y);
  // Inkonsequent aber bequem können auch Zeilen-Vektoren multipliziert werden
  // ohne sie vorher transponieren zu müssen

  for i := 1 to n do begin
    for j := 1 to r do begin
        z := 0;
        for k := 1 to m do z := z + X[i,k] * Y[k,j];
        H[i,j] := z;
    end;
  end;

  H[0,0] := n;
  H[0,1] := r;
  if (l = 1) and (m = r) then begin
      H := TRANS(H);
      H[0,0] := l;
      H[0,1] := n;
  end;

  Result := H;
end;
```

function DET(var A: TMatrix): Real;

```
// Berechnung der Determinante
var N,D,I,J,K,M: Integer;
    R,S,EPS: Real;
    H: TMatrix;
begin
  N := Round(A[0,0]);
  H := DUPL(A);
  ERROR := False;
  EPS := 1.e-8;
  D := 1;
  R := 0;
  Result := R;

  // AUFSUCHEN DES GRÖSSTEN ANFANGS-KOEFFIZIENTEN
  for I:= 1 to N-1 do begin
      M:= I;
      for K:= I+1 to N do
          if Abs(H[K,I]) > Abs(H[M,I]) then M:= K;

  // DIE I-TE ZEILE MIT DER M-TEN ZEILE TAUSCHEN
      if I<>M then begin
         D := -D;
         for J:= I to N do begin
             S:= H[I,J];
             H[I,J]:= H[M,J];
             H[M,J]:= S;
         end;
      end;
```

```
  // ELIMINATION VON X[I] AUS DER K-TEN GLEICHUNG
      for K:= I+1 to N do begin
         if H[I,I] = 0 then begin ERROR := True; Exit; end;
         S:= H[K,I]/H[I,I];
         for J:= I+1 to N do H[K,J]:= H[K,J]-H[I,J]*S
      end;
   end;

  // EIGENTLICHE BERECHNUNG DER DETERMINANTE
  // UND DES RANGES DER MATRIX
   RANK := N;
   for I:= N Downto 1 do
     if abs(H[I,I]) < EPS then RANK := RANK - 1;

   R := D;
   for I:= N DownTo 1 do R := R * H[I,I];
   DETER  := R;
   Result := R;
end;
```

procedure LINGLEI(var A: TMatrix; var X: TVektor);

```
// Lösung eines linearen Gleichungsystems in N Variablen
// A = Erweiterte Koeffizienten-Matrix, X = Lösungs-Vektor
var N,D,I,J,K,M: Integer;
    S: Real;
begin
  N := Round(A[0,0]);
  ERROR := False;
  D := 1;

  // AUFSUCHEN DES GRÖSSTEN ANFANGS-KOEFFIZIENTEN
  for I:= 1 to N-1 do begin
      M:= I;
      for K:= I+1 to N do
          if Abs(A[K,I]) > Abs(A[M,I]) then M:= K;

  // DIE I-TE ZEILE MIT DER M-TEN ZEILE TAUSCHEN
      if I<>M then begin
         D := -D;
         for J:= I to N+1 do begin
             S:= A[I,J];
             A[I,J]:= A[M,J];
             A[M,J]:= S;
         end;
      end;

  // ELIMINATION VON X[I] AUS DER K-TEN GLEICHUNG
      for K:= I+1 to N do begin
         if A[I,I] = 0 then begin ERROR := True; Exit; end;
         S:= A[K,I]/A[I,I];
         for J:= I+1 to N+1 do A[K,J]:= A[K,J]-A[I,J]*S
      end;
   end;

  // EIGENTLICHE BERECHNUNG DER LÖSUNGEN
   for I:= N DownTo 1 do begin
      S:= A[I,N+1];
      for J:= I+1 to N do S:= S-A[I,J]*X[J];
      if A[I,I]=0 then begin ERROR := True; Exit; end;
      X[I]:= S/A[I,I];
   end;

  // ZUSÄTZLICHE BERECHNUNG DER DETERMINANTE
   DETER := D;
   for I:= N DownTo 1 do DETER := DETER * A[I,I];
end;
```

procedure DREI(var A: TMatrix);

```
// Matrix in Dreiecksform
var N,D,I,J,K,M: Integer;
    S,EPS: Real;
begin
  N := Round(A[0,0]);
  DETER := 0;
  ERROR := False;
  EPS := 1.e-8;
  D := 1;

  for I:= 1 to N do begin
      M := I;
      for K:= I+1 to N do
          if Abs(A[K,I]) > Abs(A[M,I]) then M:= K;
      if I<>M then begin
         D := -D;
         for J:= I to N do begin
             S:= A[I,J];
             A[I,J]:= A[M,J];
             A[M,J]:= S;
         end;
      end;
      for K:= I+1 to N do begin
         if A[I,I] = 0 then begin ERROR := True; Exit; end;
         S:= A[K,I]/A[I,I];
         for J:= 1 to N do A[K,J]:= A[K,J]-A[I,J]*S;
      end;
  end;

  RANK := N;
  for I:= N Downto 1 do
     if abs(A[I,I]) < EPS then RANK := RANK - 1;
  DETER := D;
  for I:= N DownTo 1 do DETER := DETER * A[I,I];
end;
```

procedure JACOBI(tmax: Integer; var A,EIGVEK: TMatrix; var B: TVektor);

```
// JACOBI-Algorithmus, A = gegebene symmetrische Matrix, tmax = Anzahl der Rotationen
// EIGVEK = Matrix der Eigenvektoren, B = Vektor der Eigenwerte
var   n, i, j, k, t: Integer;
      pi, eps: Real;
      summe, grenze, nenner, speich: Real;
      phi, sinus, cosin: Real;
begin
  Error := False;
  n := Round(A[0,0]);
  pi := 3.14159265;
  eps := 1.e-8;
  EIGVEK := IDE(n);
  for j := 1 to n do
    for i := j+1 to n do
      A[i,j] := 0;
  t := 0;
  repeat
    t := t + 1;
    summe := 0.0;
    for i := 1 to n-1 do begin
      for j := i+1 to n do begin
        summe := summe + 2.0*A[i,j]*A[i,j];
      end;
    end;
    if (summe = 0.0) then begin
       Exit;
    end;
    grenze := sqrt(summe)/(n*n);
    for i := 1 to n-1 do begin
      for j := i+1 to n do begin
        if abs(A[i,j]) > grenze then begin
           nenner := A[i,i] - A[j,j];
           if abs(nenner/A[i,j]) < eps  then begin
              phi := pi / 4.0;
           end
           else begin
              phi := 0.5*arctan(2.0*A[i,j]/nenner);
           end;
```

```
          sinus := sin(phi);
          cosin := cos(phi);
          for k := i+1 to j-1 do begin
            speich := A[i,k];
            A[i,k] := cosin*A[i,k] + sinus*A[k,j];
            A[k,j] := cosin*A[k,j] - sinus*speich;
          end;
          for k := j+1 to n do begin
            speich := A[i,k];
            A[i,k] := cosin*A[i,k] + sinus*A[j,k];
            A[j,k] := cosin*A[j,k] - sinus*speich;
          end;
          for k := 1 to i-1 do begin
            speich := A[k,i];
            A[k,i] := cosin*A[k,i] + sinus*A[k,j];
            A[k,j] := cosin*A[k,j] - sinus*speich;
          end;
          speich := A[i,i];
          A[i,i] := sqr(cosin)*A[i,i]+2.0*cosin*sinus*A[i,j]+ sqr(sinus)*A[j,j];
          A[j,j] := sqr(cosin)*A[j,j]-2.0*cosin*sinus*A[i,j]+ sqr(sinus)*speich;
          A[i,j] := 0.0;
          B[0] := n;
          for k := 1 to n do begin
            B[k] := A[k,k];
            speich := EIGVEK[k,j];
            EIGVEK[k,j] := cosin*EIGVEK[k,j]-sinus*EIGVEK[k,i];
            EIGVEK[k,i] := cosin*EIGVEK[k,i]+sinus*speich;
          end;
        end;
      end;
    end;
  until t > tmax;
  Error := true;
end;
```

procedure Cardano(a,b,c,d: Real; var x,y,z: Real);

```
// Reelle Lösungen einer Gleichung dritten Grades
var disc: Real;
    p,q,s,t,u,v,w: Real;
    eps: Real;
    S0,SA,SB,SC,SD: String;
begin
     eps := 1.e-8;
     p := c - b*b/3;
     q := 2*b*b*b/27 - b*c/3 + d;

     disc := (q/2)*(q/2) + (p/3)*(p/3)*(p/3);

     str(a:12:4,SA); SA := Trim(SA);
     str(b:12:4,SB); SB := Trim(SB);
     str(c:12:4,SC); SC := Trim(SC);
     str(d:12:4,SD); SD := Trim(SD);
     S0 := '(' + SA + ')*x^3+(' + SB + ')*x^2+(' + SC + ')*x+' + '(' + SD +')';
     CPOLY := S0;

     if disc > 0 then begin
        komplex := true;
        Exit;
     end;

     if abs(disc) < eps then begin
        komplex := false;
        u :=  3*q/p;
        v := -3*q/(2*p);
        w :=  y;

        x := u - b/3;
        y := v - b/3;
        z := w - b/3;
        Exit;
     end;
```

```
     if disc < 0 then begin
        komplex := false;
        s := sqrt(-4*p/3);
        t := arccos(-sqrt(-27/(p*p*p))*q/2)/3;

        u := -s * cos(t + pi/3);
        v :=  s * cos(t);
        w := -s * cos(t - pi/3);

        x := u - b/3;
        y := v - b/3;
        z := w - b/3;
     end;
end;
```

procedure Quad(a,b,c: Real; var x,y: Real);

```
// Lösungen einer Gleichung zweiten Grades
var disc,d: Real;
    S0,SA,SB,SC: String;
begin
  disc := b*b - 4*a*c;
  d := sqrt(abs(disc))/(2*a);
  str(a:12:4,SA); SA := Trim(SA);
  str(b:12:4,SB); SB := Trim(SB);
  str(c:12:4,SC); SC := Trim(SC);

  S0 := '(' + SA + ')*x^2+(' + SB + ')*x+(' + SC + ')';
  CPOLY := S0;
  komplex := False;
  if disc < 0 then begin
     komplex := true;
     x := -b/(2*a);
     y := d;
   end
   else begin
     komplex := false;
     x := -b/(2*a) + d;
     y := -b/(2*a) - d;
   end;
end;
```

procedure Eigen23(var M,M1: TMatrix; var B: TVektor);

```
// Eigenwerte und Eigenvektoren von (2,2)- oder (3,3)-Matrizen M
// M1 = Matrix der Eigenvektoren, B = Vektor der Eigenwerte
var a,b,c,d,x,y,z: Real;
    disc: Real;
    p,q,s,t,u,v,w: Real;
    n,e,f: Integer;
    H: TMatrix;
begin
  n := Round(M[0,0]);
  if n = 2 then begin
     M1 := IDE(2);
     a := 1;
     b := -(M[1,1] + M[2,2]);
     c := M[1,1]*M[2,2] - M[1,2]*M[2,1];

     quad(a,b,c,x,y);

     B[0] := 2;
     B[1] := x;
     B[2] := y;

     if komplex then Exit;

     if M[1,2] <> 0 then begin
        u := M[1,2];
        v := -(M[1,1] - x);
     end
     else begin
        u := M[2,2] - x;
        v := -M[2,1];
     end;
     M1[1,1] := u;
     M1[1,2] := v;
```

```
    if M[1,2] <> 0 then begin
       u := M[1,2];
       v := -(M[1,1] - y);
    end
    else begin
       u := M[2,2] - y;
       v := -M[2,1];
    end;
    M1[2,1] := u;
    M1[2,2] := v;
  end;

  if n = 3 then begin
    a := 1;
    b := -(M[1,1] + M[2,2] + M[3,3]);
    c := M[1,1]*M[2,2] + M[1,1]*M[3,3] + M[2,2]*M[3,3] -
         M[1,2]*M[2,1] - M[1,3]*M[3,1] - M[2,3]*M[3,2];
    d := M[1,3]*M[3,1]*M[2,2] + M[1,1]*M[2,3]*M[3,2] + M[1,2]*M[2,1]*M[3,3] -
         M[1,1]*M[2,2]*M[3,3] - M[1,2]*M[2,3]*M[3,1] - M[1,3]*M[2,1]*M[3,2];

    Cardano(a,b,c,d,x,y,z);

    B[0] := 3;
    B[1] := x;
    B[2] := y;
    B[3] := z;

    if komplex then Exit;

    M1 := IDE(3);

    H := DUPL(M);
    H[1,1] := H[1,1] - x;
    H[2,2] := H[2,2] - x;
    H[3,3] := H[3,3] - x;

    u :=  (H[2,2]*H[3,3] - H[2,3]*H[3,2]);
    v := -(H[2,1]*H[3,3] - H[2,3]*H[3,1]);
    w := -(H[3,1]*u + H[3,2]*v) / H[3,3];

    M1[1,1] := u;
    M1[1,2] := v;
    M1[1,3] := w;

    H := DUPL(M);
    H[1,1] := H[1,1] - y;
    H[2,2] := H[2,2] - y;
    H[3,3] := H[3,3] - y;

    u :=  (H[2,2]*H[3,3] - H[2,3]*H[3,2]);
    v := -(H[2,1]*H[3,3] - H[2,3]*H[3,1]);
    w := -(H[3,1]*u + H[3,2]*v) / H[3,3];

    M1[2,1] := u;
    M1[2,2] := v;
    M1[2,3] := w;

    H := DUPL(M);
    H[1,1] := H[1,1] - z;
    H[2,2] := H[2,2] - z;
    H[3,3] := H[3,3] - z;

    u :=  (H[2,2]*H[3,3] - H[2,3]*H[3,2]);
    v := -(H[2,1]*H[3,3] - H[2,3]*H[3,1]);
    w := -(H[3,1]*u + H[3,2]*v) / H[3,3];

    M1[3,1] := u;
    M1[3,2] := v;
    M1[3,3] := w;
  end;
end;
```

procedure QR(var A,Q,R: TMatrix);

```
// QR-Zerlegung einer Matrix: A = Q * R
// Q = orthogonale Matrix
// R = obere Dreiecksmatrix
// Methode: GIVENS-Rotation
var p,c,s,z: Real;
    m,n,i,j: Integer;
    X,Xk,Qk: TMatrix;
begin
  n  := Round(A[0,0]);
  m  := Round(A[0,1]);
  X  := DUPL(A);
  Q  := IDE(n);

  Qk[0,0] := n;
  Qk[0,1] := n;
  Xk[0,0] := n;
  Xk[0,1] := n;

  for j := 1 to n-1 do begin
    for i := j+1 to m do begin
      z := abs(X[j,j]) + abs(X[i,j]);
      p := z * sqrt(sqr((X[j,j]/z)) + sqr((X[i,j])/z));
      if p = 0 then p := 1;
      c := X[j,j] / p;
      s := X[i,j] / p;

      Qk := IDE(n);
      Qk[j,j] := c;
      Qk[i,i] := c;
      Qk[i,j] := -s;
      Qk[j,i] := s;

      Q  := MULT(Qk,Q);
      Xk := MULT(Qk,X);
      X  := DUPL(Xk);
    end;
  end;

  Q  := TRANS(Q);
  R  := DUPL(X);
end;
```

procedure INVERS(var A,C: TMatrix);

```
// Allgemeine Routine zur Ermittlung der inversen Matrix
// A = gegebene quadratische Matrix, C = inverse Matrix

var Ai,Q,Qt,R,Ri: TMatrix;
    d: Real;

    procedure INV(var A,C: TMatrix);
    // Hilfs-Routine zur Ermittlung der inversen Matrix
    // von oberen Dreiecks-Matrizen, welche durch eine
    // QR-Zerlegung der Matrix A erzeugt worden sind.
    // Methode: Gauß-Jordan-Verfahren ohne Zeilentausch,
    // der bei diesen Matrizen nicht notwendig ist, weil
    // alle Elemente in der Diagonale ungleich Null sind.

    var N,I,J,K: Integer;
        z: Real;
    begin
      ERROR := False;
      N := Round(A[0,0]);
      C := IDE(N);

      for I:= 1 to N do begin
        for K:= 1 to N do begin
          if (K <> I) then begin
            if A[I,I] = 0 then begin ERROR := True; Exit; end;
            z:= A[K,I]/A[I,I];
            for J:= 1 to N do A[K,J]:= A[K,J]-A[I,J]*z;
            for J:= 1 to N do C[K,J]:= C[K,J]-C[I,J]*z;
          end;
        end;
      end;
```

```
      for I := 1 to N do begin
        z := A[I,I];
        for J := 1 to N do begin
          if z <> 0 then C[I,J] := C[I,J] / z;
        end;
      end;
    end;


// Hauptprogramm
begin
  ERROR := False;
  d := DET(A);
  if d = 0 then begin
     ERROR := True;
     Exit;
  end;
  Ai := DUPL(A);
  QR(Ai,Q,R);
  Qt := TRANS(Q);
  INV(R,Ri);
  if ERROR then Exit;
  C := MULT(Ri,Qt);
end;
```

---------- ENDE ----------