

# Appendix A

## Ein Beispiel: Das Jacobi-Verfahren

### Vorgehensweise

Wir schildern im folgenden das Jacobi-Verfahren zur Berechnung aller Eigenwerte einer symmetrischen Matrix  $A$ ; hierbei folgen wir der Darstellung bei [Toe79], § 10. Das Ziel des Verfahrens besteht in der Berechnung einer orthogonalen Matrix  $T$ , so daß  $D := T^t A T$  eine Diagonalmatrix ist. Die Eigenwerte von  $A$  stehen dann als Elemente der Diagonalen in  $D$ . Hierbei wird  $T$  iterativ bestimmt. Man setzt

$$\begin{aligned} A_0 &:= A \\ A_{k+1} &:= T_{k+1}^T A_k T_{k+1} \end{aligned}$$

Hierbei sind die Matrizen  $T_k$  orthogonal, und es genügt offenbar, diese Matrizen zu konstruieren. Setzt man  $A_k = (a_{i,j}^{(k)})_{1 \leq i,j \leq n}$ , so geht man wie folgt vor:

1. man wählt außerhalb der Diagonalen von  $A_k$  ein betragsmäßig maximales Element  $a_{r,s}^{(k)}$ , d.h. es gilt

$$|a_{r,s}^{(k)}| = \max \{a_{i,j}^{(k)} \mid 1 \leq i,j \leq n, i \neq j\}$$

2. für  $r < s$  setzt man

$$\varphi_{k+1} := \begin{cases} \frac{1}{2} \arctan\left(\frac{2a_{r,s}^{(k)}}{a_{s,s}^{(k)} - a_{r,r}^{(k)}}\right), & \text{falls } a_{s,s}^{(k)} \neq a_{r,r}^{(k)} \\ \text{sign}(a_{r,s}^{(k)}) \cdot \frac{\pi}{4}, & \text{falls } a_{s,s}^{(k)} = a_{r,r}^{(k)} \end{cases}$$

(hierbei soll  $\varphi_{k+1}$  so bestimmt werden, daß  $-\frac{\pi}{4} \leq \varphi_{k+1} \leq \frac{\pi}{4}$  gilt), und weiter

$$\begin{aligned} t_{r,r}^{(k+1)} &:= t_{s,s}^{(k+1)} := \cos(\varphi_{k+1}) \\ t_{r,s}^{(k+1)} &:= \sin(\varphi_{k+1}), \quad t_{s,r}^{(k+1)} := -t_{r,s}^{(k+1)} \end{aligned}$$

Alle Diagonalelemente  $t_{i,i}^{(k+1)}$  mit  $i \notin \{r, s\}$  setze man auf 1, alle anderen Elemente  $t_{i,j}^{(k+1)}$  auf 0. Daraus ergibt sich dann die gesuchte orthogonale Matrix  $T_{k+1} := (t_{i,j}^{(k+1)})$ .

3. Bei  $r > s$  vertausche man  $r$  und  $s$ .

Für die so eingeführte Matrix-Folge  $(A_k)_{k \geq 0}$  kann man nun zeigen, daß

$$\lim_{k \rightarrow \infty} A_k = D$$

gilt ([Toe79], Satz 10.1 – 1).

Wir geben i.f. ein *LA*-Programm für das Jacobi-Verfahren an. *LA* hat weder die trigonometrischen Funktion *sin* und *cos* noch die Arcus-Funktion *arctan* als vordefinierte Funktionen zur Verfügung, daher müssen diese Funktionen explizit berechnet werden. Für die Arcus-Funktion machen wir uns die bekannte Darstellung

$$\arctan(z) = \frac{z}{1+z^2} \left( 1 + \frac{2}{3} \cdot \frac{z}{1+z^2} + \frac{2}{3} \cdot \frac{4}{5} \left( \frac{z}{1+z^2} \right)^2 + \dots \right)$$

(für  $z^2 \neq -1$ ) zunutze, für die *sin*- und *cos*-Funktion benutzen wir die im komplexen gültige Beziehung

$$\begin{aligned} \sin(z) &= \frac{1}{2i}(e^{iz} - e^{-iz}) \\ \cos(z) &= \frac{1}{2}(e^{iz} + e^{-iz}). \end{aligned}$$

Hierbei nutzen wir aus, daß wir die komplexe Zahl  $x + iy$  als Matrix darstellen können.

$$x + iy \equiv \begin{pmatrix} x & -y \\ y & x \end{pmatrix},$$

und daß wir die Exponentialfunktion zur Verfügung haben. Wir geben den Code für diese Funktionen nicht explizit an.

## *LA* –Programm

```

program
-- Programm zur Berechnung der Eigenwerte symmetrischer
-- Matrizen nach dem Jacobi-Verfahren.
--
-- Die Programmidee stammt aus [Toe79], p. 29 ff.
--
-- Programm-Autor: M. Ebigt
-- Programm-Datum: 8. März 1990
--

const
  pi = 3.141582;

var
  weiteriterieren: boolean,
  i, j, n, zeile, spalte: integer,
  eps, elem, maximum, phi, hilfsgang: real;

```

```

put("Berechnung der Eigenwerte einer symmetrischen");
put("Matrix nach dem Jacobiverfahren");
loop
  put("Ördnung der Matrix: ");
  get(n);
  if n < 2 then
    put(" Die Ordnung der Matrix muss >= 2 sein! ");
  else
    quit;
  fi;
end loop;
var
  a, t, einheitsmatrix: mat[n, n],
  eigenwert: mat[n, 1];

put("Genauigkeitsschranke epsilon: ");
get(eps);
eps := abs(eps);
put(" Im folgenden werden die Koeffizienten der");
put(" oberen Dreiecksmatrix erwartet.");
i := 1;
loop
  j := i;
  loop
    get(a[ i, j ]); a[ j, i ] := a[ i, j ];
    j := j + 1;
    if j > n then quit; fi;
  end loop;
  einheitsmatrix[ i, i ] := 1;
  i := i + 1;
  if i > n then quit; fi;
end loop;
loop
  maximum := 0.0; zeile := 0; spalte := 0;
  i := 1;
  loop
    j := 1;
    loop
      if i ≠ j then
        if abs(a[ i, j ]) > maximum then
          zeile := i; spalte := j; maximum := abs(a[ i, j ]);
        fi;
      fi;
      j := j + 1;
      if j > n then
        quit;
      fi;
    end loop;
    i := i + 1;
    if i > n then quit; fi;
  end loop;
end loop;

```

```

if maximum > 0.0 then
  -- Suche das betragsmäßig größte Element,
  -- das nicht Diagonal-Element ist
  if a[ zeile, zeile ] = a[ spalte, spalte ] then
    phi := sgn(a[ zeile, spalte ]) * pi / 4.0;
  else
    hilfsarg := 2.0 * a[ zeile, spalte ] / (a[ spalte, spalte ] - a[ zeile, zeile ]);
    phi := atan(hilfsarg) / 2.0;
  fi;
  t := einheitsmatrix;
  t[ zeile, zeile ] := cos(phi); t[ spalte, spalte ] := cos(phi);
  if zeile < spalte then
    t[ zeile, spalte ] := sin(phi); t[ spalte, zeile ] := -sin(phi);
  else
    t[ zeile, spalte ] := -sin(phi); t[ spalte, zeile ] := sin(phi);
  fi;
  i := 1;
  loop
    eigenwert[ i, 1 ] := a[ i, i ];
    i := i + 1;
    if i > n then quit; fi;
  end loop;
  -- Iterationsschritt
  a := trans(t) * a * t;
  weiteriterieren := false;
  i := 1;
  loop
    if abs(eigenwert[ i, 1 ] - a[ i, i ]) > eps then
      -- Die Differenz ist noch zu groß
      weiteriterieren := true;
      quit;
    fi;
    i := i + 1;
    if i > n then quit; fi;
  end loop;
  if not weiteriterieren then quit; fi;
else
  quit;
fi;
end loop;
put("Die Eigenwerte sind:");
put(eigenwert);

function sgn (x: real) return real is

  -- Signum-Funktion

end sgn;

```

```
function sin (x: real) return real is  
    -- Sinus  
end sin;  
function cos (x: real) return real is  
    -- Der andere alte Römer  
end cos;  
function atan (x: real) return real is  
    -- arc tan  
end atan;  
end program.
```

## Appendix B

# Grundbegriffe der Linearen Algebra

Wir erinnern hier an einige Grundtatsachen aus der Theorie der Vektoren und Matrizen; als gründliche Einführungen in dieses Gebiet seien empfohlen [KS89] und [Lin69].

### Algebraische Struktur

$\mathcal{R}^n$  bezeichnet im folgenden die Menge aller reellen Vektoren  $(x_1, \dots, x_n)$  mit  $n$  Komponenten,  $\mathcal{M}(n, k)$  die Menge aller  $n \times k$ -Matrizen

$$\begin{pmatrix} x_{1,1} & \dots & x_{1,k} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \dots & x_{n,k} \end{pmatrix}$$

mit reellen Komponenten. Insbesondere kann  $\mathcal{R}^n$  mit  $\mathcal{M}(1, n)$  identifiziert werden.  $\mathcal{M}(n, k)$  ist mit komponentenweiser Addition und skalarer Multiplikation ein  $n \cdot k$ -dimensionaler Vektorraum über den reellen Zahlen.

Für  $\mathbf{A} = (a_{i,j})_{1 \leq i \leq n, 1 \leq j \leq k} \in \mathcal{M}(n, k)$  und  $\mathbf{B} = (b_{s,t})_{1 \leq s \leq k, 1 \leq t \leq l}$  setzt man

$$\mathbf{AB} := \left( \sum_{j=1}^k a_{i,j} b_{j,t} \right)_{1 \leq i \leq n, 1 \leq t \leq l}$$

als Produkt von  $\mathbf{A}$  und  $\mathbf{B}$ , das Produkt bildet also  $\mathcal{M}(n, k) \times \mathcal{M}(k, l)$  in  $\mathcal{M}(n, l)$  ab. Ist  $n = k$ , so bildet  $\mathcal{M}(n, n)$  einen nichtkommutativen Ring unter dieser Multiplikation. Als Spezialfälle der Multiplikation ergeben sich

$$(x_1, \dots, x_n) \begin{pmatrix} y_{1,1} & \dots & y_{1,k} \\ \vdots & \ddots & \vdots \\ y_{n,1} & \dots & y_{n,k} \end{pmatrix} = \left( \sum_{i=1}^n x_i y_{i,1}, \dots, \sum_{i=1}^n x_i y_{i,k} \right)$$

und

$$\begin{pmatrix} y_{1,1} & \dots & y_{1,k} \\ \vdots & \ddots & \vdots \\ y_{n,1} & \dots & y_{n,k} \end{pmatrix} \begin{pmatrix} z_1 \\ \vdots \\ z_k \end{pmatrix} = \left( \sum_{j=1}^k y_{1,j} z_j, \dots, \sum_{j=1}^k y_{n,j} z_j \right)$$

Vertauscht man Zeilen und Spalten einer Matrix, so entsteht die transponierte Matrix, symbolisch

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,k} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,k} \end{pmatrix}^T = \begin{pmatrix} x_{1,1} & \cdots & x_{n,1} \\ \vdots & \ddots & \vdots \\ x_{1,k} & \cdots & x_{n,k} \end{pmatrix}$$

Sei  $\mathbf{F} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\} \subset \mathcal{R}^n$  eine endliche Menge von Vektoren, so heißt  $\mathbf{F}$  *linear unabhängig*, falls

$$\forall \alpha_1, \dots, \alpha_m \in \mathcal{R} : \sum_{i=1}^m \alpha_i \mathbf{x}_i = \mathbf{0} \implies \alpha_1 = \dots = \alpha_m = 0$$

Der *Rang* einer Matrix ist die maximale Anzahl linear unabhängiger Zeilen oder, was auf das Gleiche herauskommt, die maximale Anzahl linear unabhängiger Spalten.

Sei  $\mathbf{A} \in \mathcal{M}(n, n)$  eine quadratische Matrix, dann heißt  $\mathbf{A}$  *regulär*, wenn ihr Rang gleich  $n$  ist. Eine Matrix ist genau dann regulär, wenn ihre Determinante von Null verschieden ist.

## Operationen

Die *Determinante* von  $\mathbf{A} = (x_{i,j})_{1 \leq i, j \leq n}$  ist definiert als

$$\det(\mathbf{A}) := \sum_{\pi \in \mathcal{S}_n} \operatorname{sgn}(\pi) \prod_{i=1}^n x_{i, \pi(i)}$$

Hierbei ist  $\mathcal{S}_n$  die Menge aller Permutationen von  $\{1, \dots, n\}$ , und  $\operatorname{sgn}(\pi)$  ist das Signum der Permutation,

$$\operatorname{sgn}(\pi) = \begin{cases} -1, & \text{falls } \pi \text{ ungerade ist} \\ +1, & \text{falls } \pi \text{ gerade ist} \end{cases}$$

(Eine Permutation ist genau dann gerade, wenn sie als das Produkt einer geraden Anzahl von Transpositionen  $(ij)$  geschrieben werden kann, sonst ist sie ungerade).

Für  $\mathbf{A} \in \mathcal{M}(n, k)$  ist die Menge aller Urbilder des Nullvektors

$$\operatorname{Kern}(\mathbf{A}) := \{\mathbf{x} \in \mathcal{R}^k : \mathbf{A}\mathbf{x}^T = \mathbf{0}\}$$

der *Kern* von  $\mathbf{A}$ . Man sieht, daß der Kern ein linearer Unterraum von  $\mathcal{R}^k$  ist. Als linearer Unterraum hat der Kern eine Basis  $\{\mathbf{x}_1, \dots, \mathbf{x}_l\}$ , die ohne Einschränkung als *orthonormal* angenommen werden kann, d.h. es gilt

$$\begin{aligned} \forall i \neq j : \langle \mathbf{x}_i, \mathbf{x}_j \rangle &= 0 \\ \forall i : \|\mathbf{x}_i\| &= 1 \end{aligned}$$

Mit Hilfe einer solchen Basis kann der Kern in Matrixform

$$\begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_l \end{pmatrix}$$

dargestellt werden.

Bezeichne  $\mathbf{E} = (\delta_{i,j})_{1 \leq i,j \leq n}$  die  $n$ -dimensionale Einheitsmatrix, (wobei

$$\delta_{i,j} := \text{if } i = j \text{ then } 1 \text{ else } 0 \text{ fi}$$

Kroneckers  $\delta$  ist), so ist jeder Wert  $\lambda \in \mathcal{R}$  mit

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

ein *Eigenwert* von  $\mathbf{A}$ ;  $\det(\mathbf{A} - \lambda \mathbf{E})$  heißt das *charakteristische Polynom* von  $\mathbf{A}$  und ist vom Grad  $n$ . Jede Lösung  $\mathbf{x} = \mathbf{0}$  der Gleichung

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

heißt *Eigenvektor* zum Eigenwert  $\lambda$ .

Analog zur Determinante von  $\mathbf{A} \in \mathcal{M}(n, n)$  ist die *Permanente* von  $\mathbf{A}$  definiert:

$$\text{perm}(\mathbf{A}) = \sum_{\pi \in \mathcal{S}_n} \prod_{i=1}^n x_{i,\pi(i)}$$

Eine Matrix  $\mathbf{A} \in \mathcal{M}(n, n)$  ist genau dann regulär, wenn eine Matrix  $\mathbf{B} \in \mathcal{M}(n, n)$  so existiert, daß

$$\mathbf{AB} = \mathbf{BA} = \mathbf{E}$$

gilt.  $\mathbf{B}$  heißt die *Inverse* zu  $\mathbf{A}$ . Eine Inverse existiert genau dann, wenn die Gleichung

$$\mathbf{xA} = \mathbf{0}$$

nur die triviale Lösung  $\mathbf{x} = \mathbf{0}$  besitzt. Es sei allgemeiner das Gleichungssystem

$$(*) \quad \mathbf{Ay} = \mathbf{b}$$

gegeben, wobei  $\mathbf{A} \in \mathcal{M}(n, k)$ ,  $\mathbf{y} \in \mathcal{M}(k, 1)$  und  $\mathbf{b} \in \mathcal{M}(n, 1)$  sein mögen. Ist  $\tilde{\mathbf{y}}$  eine Lösung, so auch  $\tilde{\mathbf{y}} + \tilde{\mathbf{z}}$ , wobei  $\tilde{\mathbf{z}}$  eine Lösung des zugehörigen homogenen Gleichungssystems

$$\mathbf{Az} = \mathbf{0}$$

ist. Man kann zeigen, daß sich jede Lösung von  $(*)$  so darstellen läßt,  $\tilde{\mathbf{y}}$  heißt eine *partikuläre Lösung* des Systems.

Das *innere Produkt*  $\langle \mathbf{x}, \mathbf{y} \rangle$  zweier Vektoren  $\mathbf{x}, \mathbf{y} \in \mathcal{R}^n$  ist definiert durch

$$\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{i=1}^n x_i y_i$$

und

$$\|\mathbf{x}\| := \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$$

ist die Norm des Vektors  $\mathbf{x}$ , also sein (euklidischer) Abstand zum Ursprung. In analoger Weise definiert man für die Matrix



$$\mathbf{A} = (x_{i,j})_{1 \leq i \leq n, 1 \leq j \leq k}$$

die Norm

$$\|\mathbf{A}\| := \sqrt{\sum_{i=1}^n \sum_{j=1}^k x_{i,j}^2}$$

Mit dieser Norm ist  $\mathcal{M}(n, k)$  ein Banach-Raum,  $\mathcal{M}(n, n)$  ist sogar eine Banach-Algebra, und man zeigt für  $\mathbf{A} \in \mathcal{M}(n, n)$  wie im reellen oder komplexen, daß die unendliche Reihe

$$e^{\mathbf{A}} := \sum_{n \geq 0} \frac{\mathbf{A}^n}{n!}$$

bezüglich dieser Norm in  $\mathcal{M}(n, n)$  konvergiert.  $\mathbf{A} \rightarrow e^{\mathbf{A}}$  ist die Exponentialfunktion; auf ähnliche Weise lassen sich andere transzendente Funktionen über verallgemeinerte Taylor-Reihen definieren.

# Literaturverzeichnis

- [AJ74] A.V. Aho, S. C. Johnson. LR – Parsing. *ACM Computing Surveys*, 6(2):99 – 124, 1974.
- [AM85] M. P. Atkinson, R. Morrision. Procedures as persistent data objects. *ACM Trans. Prog. Lang. Syst.*, 7(4):539 – 559, 1985.
- [ASU86] A. V. Aho, R. Sethi, J. D. Ullman. *Compilers — Principles, Techniques, Tools*. Addison-Wesley, Reading, Mass., 1986.
- [AU72] A.V. Aho, J.D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume 1: Parsing. Prentice-Hall, Englewood Cliffs, NJ, 1972.
- [AU77] A.V. Aho, J.D. Ullman. *Principles of Compiler Design*. Addison-Wesley, Reading, Mass, 1977.
- [CH87] J. Cohen, T. Hickely. Parsing and compiling using PROLOG. *ACM Trans. Prog. Lang. Syst.*, 9(2):125 – 163, 1987.
- [DF77] E. Denert, R. Franck. *Datenstrukturen*. Reihe Informatik 22. Wissenschaftsverlag im Bibliographischen Institut, Mannheim, 1977.
- [DF89] E.-E. Doberkat, D. Fox. *Software Prototyping mit SETL*. Leitfäden und Monographien der Informatik. Teubner-Verlag, Stuttgart, 1989.
- [Dob89] E.-E. Doberkat. Zur Wiederaufbereitung von Software. *Informatik — Forschung und Entwicklung*, 4:14 – 24, 1989.
- [Eng84] J. Engelfriet. Attribute grammars: Attribute evaluation methods. In B. Lorho (Hrsg.): *Methods and Tools for Compiler Construction*, 103 – 138. Cambridge University Press, Cambridge, UK, 1984.
- [GL83] G.H. Golub, C.F. van Loan. *Matrix Computations*. North Oxford Academic Press, Oxford, 1983.
- [Gut90] U. Gutenbeil. Implementation von LA mit dem Werkzeug Eli — eine vergleichende Studie. Technischer Bericht 90-03, Universität — Gesamthochschule — Essen, Mai 1990.
- [Hec77] M.S. Hecht. *Flow Analysis of Computer Programs*. North Holland Publishing Co., New York, 1977.
- [HU79] J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass., 1979.

- [Kas80] U. Kastens. Ordered attribute grammars. *Acta Informatica*, 13:229 – 256, 1980.
- [Kas84] U. Kastens. The GAG - system – a tool for compiler construction. In B. Lorho (Hrsg.): *Methods and Tools for Compiler Construction*, 165 – 182. Cambridge University Press, Cambridge, UK, 1984.
- [KHZ82] U. Kastens, B. Hutt, E. Zimmermann. *GAG: A Practical Compiler Generator*. Lecture Notes in Computer Science 141. Springer-Verlag, Berlin, 1982.
- [Knu73] D.E. Knuth. *The Art of Computer Programming*, volume 1, Fundamental Algorithms. Addison-Wesley, Reading, Mass, 2. Auflage, 1973.
- [Knu81] D.E. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison-Wesley, Reading, Mass, 2. Auflage, 1981.
- [KP84] B.P. Kernighan, R. Pyke. *The UNIX Programming Environment*. Prentice-Hall, Englewood Cliffs, N. J., 1984.
- [KS88] A. Kielbasinski, H. Schwetlick. *Numerische lineare Algebra-eine computerorientierte Einführung*. Harri Deutsch Verlag, Frankfurt a. M., 1988.
- [KS89] K.H. Kiyek, F. Schwarz. *Mathematik für Informatiker*. Leitfäden und Monographien der Informatik. Teubner-Verlag, Stuttgart, 1989.
- [Lan65] S. Lang. *Algebra*. Addison-Wesley, Reading, Mass, 1965.
- [Lin69] R. Lingenberg. *Lineare Algebra*. Bibliographisches Institut, Mannheim, 1969.
- [LM81] H. Ledgard, M. Marcotty. *The Programming Language Landscape*. Science Research Associates, Chicago, 1981.
- [Lor84] B. Lorho, editor. *Methods and Tools for Compiler Construction*. Cambridge University Press, Cambridge, UK, 1984.
- [Os60] E.E. Osborne. On pre-conditioning of matrices. *Journal of the ACM*, 7:338 – 345., 1960.
- [RM85] P. Rechenberg, H. Mössenböck. *Ein Compiler-Generator für Mikrocomputer*. Carl Hanser Verlag, München und Wien, 1985.
- [Sto72] J. Stoer. *Einführung in die Numerische Mathematik I*. Springer-Verlag, Heidelberg, 1972.
- [Toe79] W. Törnig. *Numerische Mathematik für Ingenieure und Physiker*, Band II: Eigenwertprobleme und numerische Methoden der Analysis. Springer-Verlag, Berlin, 1979.
- [WG84] W.M. Waite, G. Goos. *Compiler Construction*. Springer-Verlag, New York, 1984.
- [WR71] J.H. Wilkinson, C. Reinsch. *Linear Algebra*. Springer-Verlag, New York, 1971.
- [Zim82] H. Zima. *Compilerbau I*. Reihe Informatik 36. Wissenschaftsverlag im Bibliographischen Institut, Mannheim, 1982.
- [Zim83] H. Zima. *Compilerbau II*. Reihe Informatik 37. Wissenschaftsverlag im Bibliographischen Institut, Mannheim, 1983.

# Index

$\Rightarrow^*$  16  
 $\Rightarrow$  16  
 $\Rightarrow_L$  18  
 $\Rightarrow_R$  18

## A

Abhängigkeit, direkte 26  
Abhängigkeit, induzierte 26  
Ableitungsrelation 16  
Abschluß 36  
Abstieg, rekursiver 20  
activation record 34, 108  
Adjunkte 57  
Adresse 118  
Adreßlücke 117  
Anweisung, bedingte 43  
Anweisung, fallgesteuerte 44  
Anweisungsfolge 43  
Arithmetik-Stack 108, 110  
Assemblercode 118  
Assoziativität 70  
Attribut, geerbtes 24  
Attribut, intrinsisches 24  
Attribut, synthetisiertes 24  
Attributierung 23  
Attributierungsregel 24  
Ausdruck, arithmetischer 42  
Ausdruck, Boolescher 42  
Ausdruck, regulärer 15  
Ausdruck, relationaler 42  
Ausschnitt aus Matrizen 96  
Auswertungsstrategie 26, 27, 28  
Automat, endlicher 15

## B

baseglob 109  
baselok 109  
Basis, orthormierte 56  
Bedingung, semantische 23  
Befehlssegment 119

Benutzung 31  
Bezeichner 30, 69  
Bindung, dynamische 32  
Bindung, lexikalische 32  
Bindungsregel 32  
Block 32  
Block, lebender 37  
Block, toter 37

## C

call by name 33  
call by reference 33, 45, 114  
call by value 33, 45, 114  
call by value/result 33  
Coco 22  
Code-Generator 14  
Code-Segment 108  
Code-Tafel 29

## D

data counter 118  
Daten, lokale 34  
Daten-Stack 108  
Daten-Segment 119  
Deklaration 31, 41  
Determinante 47  
Diagonale 47  
Disjunktion 46  
Display 36  
Division 46  
Drei-Adreß-Code 14, 67  
Dreiecksform, obere 52

## E

Eigenvektor 59  
Eigenwert 47, 59  
Exponentialfunktion 60  
Extraktion 111

## F

Fehleraktion 20  
 Fehlermatrix 61  
 Fehlermeldung 84  
 Feld, dynamisches 35  
 Feld-Deklaration 25  
 Feld-Deskriptor 35  
 FIRST( $\alpha$ ) 18  
 FOLLOW(A) 18  
 Funktions-Vereinbarung 45

## G

garbage collection 37  
 Gaußscher Algorithmus 55, 57  
 Generator 18  
 Gliederung von Programmen 46  
 Gram-Schmidt-Algorithmus 57  
 Grammatik, attributierte 23  
 Grammatik, kontextfreie 16  
 Gültigkeitsbereich 31, 77, 108

## H

Hash-Tafel 31, 65  
 Hauptdiagonale 54  
 Hauptprogramm 46  
 Heap 36, 108  
 Hessenberg-Form, obere 59  
 Hierarchie 43  
 Householder Orthogonalisierung 57  
 Hülle 21

## I

Instruktionszeiger 32

## J

Jensen's device 33

## K

Kern 47  
 Kollektion, kanonische 21  
 Konjunktion 46  
 Konstante 41, 67, 90  
 Kontext-Bedingung 23  
 Konturenmodell 32  
 Korrektheit, semantische 23

## L

l-Wert 33  
 LALR(1) 22

lex 8, 16, 69  
 Links-Ableitung 18  
 LL(1)-Parser 19  
 LL(1)-Tabelle 19  
 LL(1) 18  
 Logarithmus 47  
 LR(1) 18, 20

## M

Maschinenbefehl 117  
 Maschineninstruktion 117  
 Matrix, inverse 57  
 Matrix-Selektion 42  
 Matrix-Umformung 51  
 Matrizen-Parameter 45  
 Maximum 47  
 Minor 57  
 most closely nested rule 31

## N

Namens-Analyse 23  
 natürliche Zahl 54  
 Norm 47  
 Norm, euklidische 53

## O

Objekte, zusammengesetzte 41  
 Operator, überladener 23  
 Optimierungsphase 14  
 Option 120

## P

Padé-Funktion 61  
 partikuläre Lösung 48  
 Permanente 47, 54  
 Permutationsmatrix 58  
 Pivotisierung 51  
 Portabilität 107  
 Potenz 46  
 preorder 92  
 pretty printing 92  
 Produkt, inneres 47  
 program counter 34, 108, 118  
 Programm-Listing 84  
 Prozedur-Aufruf 35  
 Prozedur-Parameter 32  
 Prozedur-Vereinbarung 45  
 Pseudo-Operation 118

## Q

QR-Algorithmus 59  
Quadrat 48  
Quadratwurzel 48  
Quellprogramm 14

## R

r-Wert 33  
Rang 48, 54  
Rechts-Ableitung 18  
Reduktion 18  
Referenzierbarkeit 86  
Register-Inhalt 34  
Reservierung von Speicherplatz 118  
Resultat-Parameter 43  
Rückgabewert 34

## S

S-Attributierung 26  
Scanner 69  
Schleife, benannte 44  
Schleifenname 44  
Schleifenumgebung 79  
Schlüsselwort 30  
Scope 67  
Segment, .data 91, 95  
Segment, .text 92  
Segmentadresse 119  
short circuit evaluation 42  
Sichtbarkeit 46, 108  
SLR(1) 22  
Software Prototyping 8  
Sortierung, topologische 26  
Spalte 48  
Speicherallokation 36  
Speicherbereinigung 37  
Sprungziel 91  
Stabilität, numerische 51  
Stack-Maschine 107  
Stack 36  
Standard-Ausgabe 48  
Standard-Eingabe 47  
Standard-Funktion 86  
String-Speicher 30  
String-Konstante 41, 69  
Symbol-Tabelle 30, 65  
Syntaxbaum 14, 67

## T

Teilmatrix 42, 111  
Token-String-Field 66  
Tokenklasse 69  
Token 13  
Typ, primitiver 41  
Typ-Analyse 23  
Typ-Identifikation 79  
Typen-Stack 108, 109  
Typvertäglichkeit 86  
Typwandlung 111

## U

Übergangsfunktion 21  
Umgebungszeiger 32  
Umgebung 31

## V

Variable, temporäre 30  
Variablen-Deklaration 41  
Verschattungsprobleme 79  
Vorgänger, dynamischer 32, 34  
Vorgänger, statischer 32, 34

## W

Wert, temporärer 34  
Werte-Parameter 43  
Wort, reserviertes 69

## Y

yacc 8, 22, 65, 69

## Z

Zeile 48  
Zeilenelement 120  
Zeilensummennorm 60  
Zielmaschine 117  
Zustand 31  
Zustandsabbildung 31  
Zuweisung 43  
Zwischencode 14  
Zwischendarstellung 65, 67

Der LA-Compiler  
 Quell-Code und ablauffähiges Programm  
 Stand: 2.7.1990

Die beigelegte Diskette enthält in den drei Unterverzeichnissen

*assemble*, *maschine*, und *compiler*

den Quellcode des LA-Compilers,  
 im Verzeichnis

*beispiel*

das Beispielprogramm aus Anhang A  
 und im Verzeichnis

*programm*

eine ausführbare Version des Compilers für MS-DOS.

Systemanforderung: AT-kompatibler MS-DOS-Rechner mit mindestens 640 KB internem Speicher.

Der Compiler wurde unter UNIX auf SUN-Workstations mit dem Motorola 68020 Prozessor entwickelt. Er ist deshalb einfach auf Personal Computer mit dem gleichen Prozessortyp (etwa Apple MacIntosh oder Atari) portierbar; hierzu müssen lediglich

- die Speicheranforderungen für Assembler und Maschine reduziert und
- die Variablen vom Typ *int* in solche vom Typ *long* umgewandelt werden.

Die Portierung auf Rechner mit Intel-Prozessoren ist ein weitaus schwierigeres Unterfangen, von dem eigentlich abzuraten ist.

Die im Verzeichnis *programm* stehenden ablauffähigen Programme resultieren aus einer Portierung nach C, Version 1.5, auf einem IBM-AT (mit 640 KB Hauptspeicher — mit weniger läuft Turbo C nicht!). Die Transparenz und Lesbarkeit des C-Codes ist bei der Portierung stark in Mitleidenschaft gezogen worden; deshalb haben wir uns entschieden, die ursprünglichen UNIX-Quelldateien auf diese Diskette zu geben. Interessenten können jedoch auch die Turbo C-Quelldateien von uns erhalten, wenn sie eine Leerdiskette (5,25", HD) senden an

Dr. Dietmar Fox  
 Universität Hildesheim  
 Institut für Informatik  
 Marienburger Platz 22  
 D-3200 Hildesheim.