FORTRAN - SC

FORTRAN for Scientific Computation

Institute for Applied Mathematics
Prof. Dr. U. Kulisch
University of Karlsruhe
Karlsruhe, West-Germany

FORTRAN-SC

FORTRAN for Scientific Computation

FORTRAN is the abbreviation of **FORmula TRANslator**. Modern numerical applications require formulas containing vectors, matrices and intervals. Furthermore, all standard arithmetic operations, especially the vector/matrix operators, should always deliver a result of highest accuracy. Until now FORTRAN compilers have been unable to satisfy these requirements. For this reason, a new language called FORTRAN-SC was developed. FORTRAN-SC is an extension of FORTRAN 77 intended for engineering and scientific computation. It is particularly suitable for the development of numerical algorithms which deliver highly accurate and automatically verified results.

The FORTRAN-SC compiler and runtime system have been developed at the Institute for Applied Mathematics at the University of Karlsruhe in collaboration with the IBM Research and Development Laboratory, Böblingen. FORTRAN-SC is implemented under VM/SP on the IBM system /370 architecture. Extensive programming experiences have demonstrated the usefulness and effectiveness of the language and the reliability of the implementation.

The most important new concepts in FORTRAN-SC are:

- vector/matrix operations
- dynamic arrays
- subarrays
- functions with array result
- user-defined operators
- interval arithmetic
- dot product expressions

These concepts are briefly explained and their advantages are highlighted. Subsequently, their use is illustrated in sample programs for typical applications of FORTRAN-SC.

Vector/Matrix Operations

The traditional mathematical vector/matrix operations are provided as standard arithmetical operators. Type conversion functions are also available for vectors and matrices.
The advantages of vector/matrix operations are:

- "natural", mathematical notation of array expressions
- functions and operators instead of subroutine calls
- no long sequences of subroutine calls
- no unnecessary loops and indexing

Dynamic Arrays

The use of vectors and matrices is simplified through dynamic arrays. The size and the index ranges of dynamic arrays are determined at execution time, not at compile time. Thus dynamic arrays reduce runtime storage (see examples 2, 3, 4 and 5).
The advantages of dynamic arrays are:

- memory allocation only as required
- size and index range modifiable during execution
- no recompilation necessary for problems of varying size
- argument type and index checking
- full compatibility with static arrays

Subarrays

Rows and columns of matrices and in general arbitrary "rectangular" parts of arrays may be accessed via a special notation for subarrays (see example 3).

Functions with Array Result

In standard FORTRAN a function must have a scalar result. FORTRAN-SC also allows functions with array result, e.g. the result may be an interval matrix.

User-Defined Operators

The user can declare specific operators on the intrinsic types of FORTRAN-SC (see example 5). Moreover, it is possible to declare new data types, e.g. POLYNOMIAL, and operators for these types. User-defined operators can be used to mimic the mathematical notation of a problem with operations which are not predefined. They enhance the readability of complicated expressions.

Interval Arithmetic

FORTRAN-SC offers the additional data types INTERVAL and COMPLEX INTERVAL which are supported by numerous standard operators and functions. A special notation for interval constants and interval I/O guarantees correct rounding of decimal data. Arithmetic expressions in higher numerical spaces - e.g. expressions involving interval matrices - may be written in mathematical notation using operator symbols like +, -, *, /. Additional operators like .IS. for the intersection of two intervals or .SB. for the subset relation are available (see example 1).
The advantages of interval arithmetic are:

- control of rounding errors
- verification of the solution by inclusion of the exact result
- stability and sensitivity analysis
- treatment of problems with imprecise data

Dot Product Expressions

Dot product expressions are sums of real or complex constants, variables, vectors, matrices and single products of these. They frequently occur in defect correction and iterative refinement methods where the elimination of cancellation is crucial.

Dot product expressions can be evaluated without error. Their exact evaluation is an important tool in many numerical applications. Their result may be stored to full accuracy in a variable of type DOT PRECISION or rounded to one of the adjacent floating-point numbers or enclosed in an interval of maximum accuracy (see example 4).

Easy Access to ACRITH

The ACRITH Subroutine Library (ACRITH is a program product of IBM, refer to [6], [7] and [8]) is a collection of problem solving routines for standard problems of numerical analysis, for example:

- evaluation of arithmetic expressions
- matrix inversion, linear systems (dense and sparse)
- eigenvalues, eigenvectors
- systems of nonlinear equations
- linear programming
- evaluation and zeros of polynomials

All ACRITH routines compute verified bounds of high accuracy for the exact solution. In FORTRAN-SC, many of these subroutines are accessible as functions. The argument lists of all functions and subroutines are simplified. The basic routines, e.g. for vector/matrix and interval arithmetic, are available as predefined operators (see example 1).

Additional Features

- standard functions of high accuracy for all arithmetic types
- standard operators and constants with directed roundings
- input/output data conversion with controlled rounding of highest accuracy
- identifiers with up to 31 characters
- lower case letters
- WHILE and REPEAT loops

List of Sample Programs

The examples demonstrate various concepts of FORTRAN-SC.

1. Interval Newton Method

    - data type INTERVAL
    - interval operators
    - interval standard functions
    - REPEAT - UNTIL loop

2. Runge-Kutta Method

    - dynamic arrays
    - array operators
    - functions with array result

3. Gauss Algorithm

    - dynamic arrays
    - subarrays

4. Trace of a Product Matrix

    - dynamic arrays
    - subarrays
    - dot product expressions
    - SUM-notation

5. Boothroyd/Dekker Matrix

    - dynamic arrays
    - operator concept

Well-known algorithms were deliberately chosen so that a brief explanation of the mathematical background will suffice. Since the programs are largely self-explanatory, comments are kept to a minimum. Note that FORTRAN-SC allows lower and upper case letters and identifiers with up to 31 characters.

Example 1

Interval Newton Method

An inclusion of a zero of the real-valued function f (x) is computed. It is assumed that f ' (x) is a continuous function on the interval [a,b], where $0 \notin \{f'(x) : x \in [a,b]\}$ and $f(a) \cdot f(b) < 0$. If an inclusion X_n for the zero of such a function f(x) is already known, a better inclusion X_{n+1} can usually be computed by the iteration formula:

$$X_{n+1} := \left(m(X_n) - \frac{f(m(X_n))}{f'(X_n)} \right) \cap X_n$$

where m(X) is some point in the interval X (for example the midpoint).

For this example, the function $f(x) = \sqrt{x} + (x+1) \cdot \cos x$ is used.
In FORTRAN-SC, interval expressions are written in mathematical notation. Generic function names are used for the interval square root and interval sine and cosine functions. For the mathematical theory see [1].

```
      P R O G R A M   I N E W T
      INTERVAL X, Y, F, DERIV, M
      LOGICAL CRITER
      EXTERNAL F, DERIV, M, CRITER

 1    WRITE(*,*) 'Please enter starting interval'
C     The interval notation in FORTRAN-SC is (< inf , sup >)
      READ (*,*,END = 999) Y
      IF ( CRITER(Y) ) THEN
         REPEAT
            X= Y
            WRITE(*,*) X

C           The iteration formula (.IS. is the InterSection operator):
            Y= ( M(X) - F(M(X))/DERIV(X) ) .IS. X

         UNTIL (X .EQ. Y)
      ELSE
         WRITE(*,*) ' Criterion not satisfied'
      END IF
      GOTO 1
 999  STOP
      END
```

Example 1

```
      F U N C T I O N   M (X)
C     A point close to the midpoint of the interval X is computed.
C     The corresponding point interval is returned.
      INTERVAL M, X
      M = IVAL( INF(X) + 0.5*(SUP(X)-INF(X)) )
      END


      F U N C T I O N   F (X)
      INTERVAL F, X
      F = SQRT(X) + (X + 1) * COS(X)
      END


      F U N C T I O N   D E R I V (X)
C                             F ' (X)
      INTERVAL DERIV, X
      DERIV = 1 / (2 * SQRT(X)) + COS(X) - (X + 1) * SIN(X)
      END


      F U N C T I O N   C R I T E R (X)
      LOGICAL CRITER
      INTERVAL X, F, DERIV
      EXTERNAL F, DERIV
      CRITER = ( 0 .IN. F(X) ) .AND. .NOT. ( 0 .IN. DERIV(X) )
C                   .IN. is the relational operator "element of"
      END
```

With the starting interval (<2, 3>) the computed inclusions are

```
(<   0.2000000E+01 ,   0.3000000E+01 >)
(<   0.2000000E+01 ,   0.2218138E+01 >)
(<   0.2051400E+01 ,   0.2064727E+01 >)
(<   0.2059037E+01 ,   0.2059055E+01 >)
(<   0.2059044E+01 ,   0.2059046E+01 >)
```

Example 1

The same algorithm written in FORTRAN 77 using the ACRITH subroutine library consists of a long list of subroutine calls. Such a program is hard to read, write and understand. Coding and debugging is very time consuming.

```
      P R O G R A M   I N E W T
      LOGICAL CRITER
      REAL YLB,YUB,YN,XLB,XUB,DLB,DUB,MIDLB,MIDUB
      REAL INTLB,INTUB,MIDPNT
      INTEGER IER
      CHARACTER*80 STRING
      CHARACTER*51 STR51
      INTEGER MAXLEN,ICODE,LENGTH

111   WRITE(*,*) ' Please enter starting interval'
      READ(*,*,END=999) STRING
C     convert the interval with correct rounding:
      CALL CONV(STRING,80,YLB,YUB,YN,ICODE,LENGTH)

      IF ( CRITER(YLB,YUB) ) THEN
10        XLB=YLB
          XUB=YUB
C         Output with interval rounding:
          CALL IOUT(YLB,YUB,15,STR51)
          WRITE(*,*) STR51
C         The above iteration formula:
          CALL DERIV(XLB,XUB,DLB,DUB)
          MIDLB=MIDPNT(XLB,XUB)
          MIDUB=MIDLB
          CALL F (MIDLB,MIDUB,INTLB,INTUB)
          CALL IDIV (INTLB,INTUB,DLB,DUB,INTLB,INTUB,IER)
          CALL ISUB (MIDLB,MIDUB,INTLB,INTUB,INTLB,INTUB,IER)
C         Intersection of two intervals:
          YUB=MIN (INTUB,XUB)
          YLB=MAX (INTLB,XLB)
          IF (.NOT. (YLB.EQ.XLB.AND.YUB.EQ.XUB) )  GOTO 10
      ELSE
          WRITE(*,*) ' Criterion not satisfied'
          GOTO 111
      END IF

999   STOP
      END

C***********************************************************************
C M I D P N T
C   RESULT IS THE MIDPOINT OF THE INTERVAL ARGUMENT
C***********************************************************************
      FUNCTION  M I D P N T (XLB,XUB)
      REAL XLB,XUB,MIDPNT

      MIDPNT = XLB+0.5*(XUB-XLB)

      RETURN
      END
```

Example 1

```
C*******************************************************************
C  F
C    INTERVAL FUNCTION F(X) = SQRT (X) + (X + 1) * COS (X)
C*******************************************************************
      SUBROUTINE  F (XLB,XUB, RESLB,RESUB)
      REAL XLB,XUB, RESLB,RESUB, RLB,RUB, CLB,CUB
      INTEGER IER

      CALL ISQRT (XLB,XUB, RLB,RUB)
      CALL ICOS (XLB,XUB, CLB,CUB)
      CALL IADD (XLB,XUB, 1.0,1.0, RESLB,RESUB, IER)
      CALL IMUL (RESLB,RESUB, CLB,CUB, RESLB,RESUB, IER)
      CALL IADD (RLB,RUB, RESLB,RESUB, RESLB,RESUB, IER)
      RETURN
      END

C*******************************************************************
C  D E R I V
C    INTERVAL FUNCTION F'(X) = 1/(SQRT(X)*2) + COS (X) -
C                              (X + 1) * SIN (X)
C    F'(X) IS THE FIRST DERIVATIVE OF F(X)
C*******************************************************************
      SUBROUTINE  D E R I V (XLB,XUB, RESLB,RESUB)
      REAL XLB,XUB, RESLB,RESUB, CLB,CUB, SLB,SUB, RLB,RUB
      INTEGER IER

      CALL ICOS (XLB,XUB, CLB,CUB)
      CALL ISIN (XLB,XUB, SLB,SUB)
      CALL ISQRT (XLB,XUB, RLB,RUB)
      CALL IMUL (RLB,RUB, 2.0,2.0, RLB,RUB, IER)
      CALL IDIV (1.0,1.0, RLB,RUB, RLB,RUB, IER)
      CALL IADD (XLB,XUB, 1.0,1.0, RESLB,RESUB, IER)
      CALL IMUL (RESLB,RESUB, SLB,SUB, RESLB,RESUB, IER)
      CALL ISUB (CLB,CUB, RESLB,RESUB, RESLB,RESUB, IER)
      CALL IADD (RLB,RUB, RESLB,RESUB, RESLB,RESUB, IER)
      RETURN
      END


C*******************************************************************
C  C R I T E R
C    GUARANTEES EXISTENCE AND UNIQUENESS OF A ZERO IN THE INTERVAL X
C*******************************************************************
      LOGICAL FUNCTION  C R I T E R (XLB,XUB)
      REAL XLB,XUB,INTLB,INTUB,YLB,YUB

      CALL F (XLB,XUB,INTLB,INTUB)
      CALL DERIV (XLB,XUB,YLB,YUB)
      CRITER = INTLB.LE.0.0 .AND. INTUB.GE.0.0 .AND.
     &           (YLB.GT.0.0 .OR. YUB.LT.0.0)
      RETURN
      END
```

Example 2

Runge-Kutta Method

The initial-value problem for a system of differential equations is to be solved.

The Runge-Kutta method to solve one differential equation may be written in FORTRAN 77 in an almost mathematical notation. In FORTRAN-SC it is possible to use the same notation for a system of differential equations. The concept of dynamic arrays is used to make the main program independent of the size of the system. Only as much storage as needed is occupied during runtime.

The following system of first-order differential equations

$$Y' = F(x, Y)$$

with initial condition $Y(x_0) = Y_0$ is considered. If the solution Y is known at a point x, then $Y(x+h)$ may be computed by:

$$\begin{aligned}
K1 &= h \cdot F(x, Y) \\
K2 &= h \cdot F(x + h/2, Y + K1/2) \\
K3 &= h \cdot F(x + h/2, Y + K2/2) \\
K4 &= h \cdot F(x + h, Y + K3) \\
Y(x+h) &= Y + (K1 + 2 \cdot K2 + 2 \cdot K3 + K4) / 6
\end{aligned}$$

Starting at x_0, an approximate solution can be computed at the points $x_i = x + i \cdot h$.

Example 2

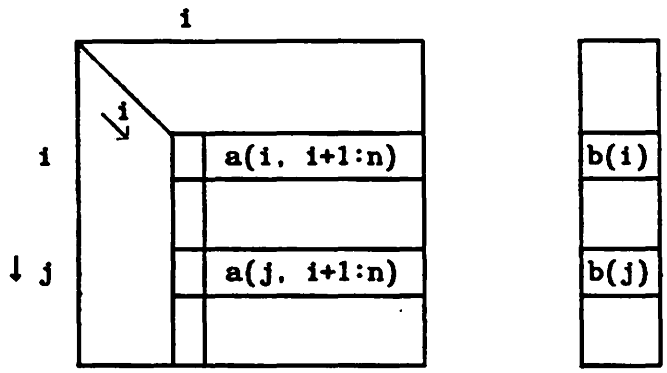
```
      P R O G R A M   R U N G E
C     The actual size of the problem is determined in routine INIT
      DYNAMIC / REAL(:) / F, Y, K1, K2, K3, K4
C                         F, Y, K1, K2, K3, K4 are real vectors
      REAL x, h
      EXTERNAL F

      CALL INIT(x, Y, h)
*********************************************************************
****  Classical Runge-Kutta method    (10 steps)                ****
****  for a system of first-order differential equations        ****
****       Y' = F(x, Y)                                         ****
*********************************************************************
      DO 10 i = 1, 10
         K1 = h * F(x, Y)
         K2 = h * F(x + h / 2, Y + K1 / 2)
         K3 = h * F(x + h / 2, Y + K2 / 2)
         K4 = h * F(x + h, Y + K3)
         Y  = Y + (K1 + 2 * K2 + 2 * K3 + K4) / 6
         x  = x + h
         WRITE (*,*) 'x=', x, '     Y=(', Y, ')'
10    CONTINUE
      END


      S U B R O U T I N E   I N I T (x, Y, h)
******************************************************************
*                              ( y1 )   ( 0 )                    *
*  Initial values  Y(0.0) = ( y2 ) = ( 1 )                       *
*                              ( y3 )   ( 1 )                    *
******************************************************************
      DYNAMIC / REAL(:) / Y
      REAL x, h

      ALLOCATE Y(3)
      x = 0
      h = 0.1
      Y(1) = 0
      Y(2) = 1
      Y(3) = 1
      END


      F U N C T I O N   F (x, Y)
******************************************************************
*                          (            y2*y3 )                  *
*  Problem: Y'= F(x,Y) = (           -y1*y3 )                    *
*                          (   -0.522*y1*y2 )                    *
******************************************************************
      DYNAMIC / REAL(:) / F, Y
      REAL x

      ALLOCATE F(=Y)
      F(1) =         Y(2)*Y(3)
      F(2) =        -Y(1)*Y(3)
      F(3) = -0.522*Y(1)*Y(2)
      END
```

Example 3

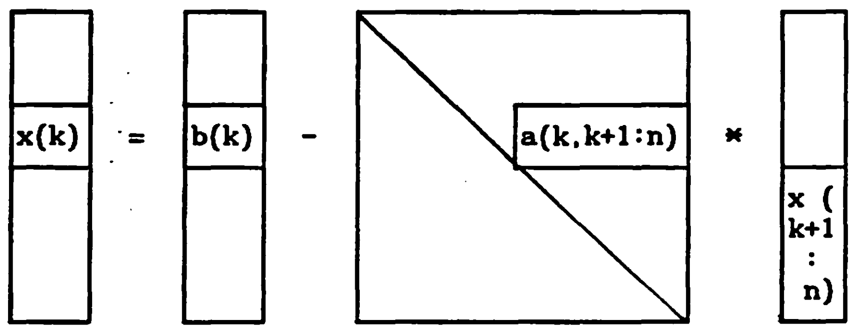
Gauss Elimination

The system of linear equations $A \cdot x = b$ is to be solved.

This example demonstrates the usage of subarrays. In the i-th step of the Gaussian algorithm, multiples of the i-th row are subtracted from the rows $i+1, \ldots, n$. In order to build up a LU-decomposition of the system matrix A, the rows must only be changed in columns $i+1, \ldots, n$ in the i-th step.



In the k-th step of backsolving the triangular system, the trailing part of the k-th row of A is multiplied by the computed part of the solution vector x as shown below:



Example 3

```
      P R O G R A M   G A U S S

      DYNAMIC /REAL(:,:)/ A, /REAL(:)/ x, b
C                 matrix  A ,  vectors x and b
      INTEGER i, j, k, n

      WRITE (*,*) 'Enter system dimension n'
      READ (*,*) n
      ALLOCATE A(n,n), x, b(n)

      WRITE (*,*) 'Enter the coefficient matrix A row-wise',
     &            ' and then the right-hand side vector b'

      READ (*,*) (A(i,:), i=1,n), b
C                 A(i,:) is the i-th row of A

C     GAUSS elimination with LU-decomposition

      DO 10 i = 1, n
        DO 10 j = i + 1, n
          A(j,i) = A(j,i) / A(i,i)
          A(j,i+1:n) = A(j, i+1:n) - A(j,i) * A(i, i+1:n)
          b(j)       = b(j)        - A(j,i) * b(i)
 10   CONTINUE

C     Backsolving

      x(n) = b(n) / A (n,n)
      DO 20 k = n-1, 1, -1
         x(k) = ( b(k) - A(k, k+1:n) * x(k+1:n) ) / A(k,k)
 20   CONTINUE

      WRITE(*,*) 'Approximate solution :', x

      END
```

Example 4

Trace of a Product Matrix

Dot product expressions are sums of real or complex constants, variables, vectors, matrices and single products of these. A dot product expression which is parenthesized and prefixed by the symbol # is evaluated without rounding error. If the exact result is to be stored in a floating-point variable, the #-sign must be followed by one of the rounding symbols < or > for the directed roundings or * for the rounding to the nearest floating-point number. A special notation for finite sums is provided. They are introduced by the keyword SUM. The notation is similar to implied-DO.

The following FORTRAN-SC program demonstrates the use of this tool. The trace of a product matrix A•B is computed without evaluating the product matrix itself. The result will be of maximum accuracy, i.e. it is the best possible floating-point approximation of the exact solution.

The trace of the product matrix is given by:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{ji}$$

```
      PROGRAM   T R A C E

      INTEGER i, j, n
      DYNAMIC /DOUBLE REAL(:,:)/ A, B

      READ (*,*) n
      ALLOCATE A, B (n, n)
      READ (*,*) A, B

      WRITE(*,100) #*( SUM( A(i,:) * B(:,i), i = 1, n ) )

100   FORMAT(' The trace of the product matrix is : ',G20.10)
      END
```

Example 5

Boothroyd/Dekker Matrix

This program generates a Boothroyd/Dekker matrix [13]. The elements of the n×n Boothroyd/Dekker matrix are given by

$$d_{ij} = \binom{n+i-1}{i-1} \cdot \binom{n-1}{n-j} \cdot \frac{n}{i+j-1}$$

These matrices are often used to test inversion algorithms because their inverse is explicitly known. The i , j-th element of the inverse is

$$(-1)^{i+j} \cdot d_{ij}$$

In the following FORTRAN-SC program, the user-defined operator .over. is used to compute the binomial coefficients of two integer values n and k. The name of the implementing function is "N OVER K".

```
      PROGRAM   D E K K E R
      DYNAMIC /INTEGER(:,:)/ D
      INTEGER i, j, n
      OPERATOR .over. = N OVER K (INTEGER, INTEGER) INTEGER

      WRITE(*,*) ' Please enter the dimension of the matrix'
      READ (*,*) n
      ALLOCATE D(n, n)

      DO 10 i = 1, n
         DO 10 j = 1, n
            D(i,j) = ((n+i-1) .over. (i-1)) *
     &                  ((n-1) .over. (n-j)) * n / (i+j-1)
10    CONTINUE

      DO 20 j = 1, n
         WRITE(*,*) D(j,:)
20    CONTINUE
      END


      INTEGER FUNCTION  N OVER K (n, k)
      INTEGER n, k, i

      N OVER K = 1
      DO 10 i = 1, MIN(k, n-k)
         N OVER K = N OVER K * (n-i+1) / i
10    CONTINUE
      END
```

Literature

[1] Alefeld, G., Herzberger, J.: Introduction to Interval Analysis. New York: Academic Press (1983).

[2] American National Standards Institute: American National Standard Programming Language FORTRAN. ANSI X3.9-1978 (1978).

[3] American National Standards Institute: American National Standard Programming Language FORTRAN. Draft S8, Version 104, ANSI X3.9-198x (1987).

[4] Bleher, J. H., Rump, S. M., Kulisch, U., Metzger, M., Ullrich, Ch., Walter, W.: FORTRAN-SC: A Study of a FORTRAN Extension for Engineering/Scientific Computation with Access to ACRITH. Computing 39, pp. 93-110, Springer (1987).

[5] Bohlender, G., Kaucher, E., Klatte, R., Kulisch, U., Miranker, W. L., Ullrich, Ch., Wolff v. Gudenberg, J.: FORTRAN for Contemporary Numerical Computation. IBM Research Report RC 8348 (1980). Computing 26, pp. 277-314 (1981).

[6] IBM High-Accuracy Arithmetic Subroutine Library (ACRITH). General Information Manual, GC 33-6163-02, 3rd Edition (April 1986).

[7] IBM High-Accuracy Arithmetic Subroutine Library (ACRITH). Program Description and User's Guide, SC 33-6164-02, 3rd Edition (April 1986).

[8] IBM System/370 RPQ, High-Accuracy Arithmetic. SA 22-7093-0 (1984).

[9] Kulisch, U. (ed.): PASCAL-SC: A PASCAL Extension for Scientific Computation. Information Manual and Floppy Disks, Version IBM PC. Stuttgart: B. G. Teubner; Chichester: John Wiley & Sons (1987).

[10] Kulisch, U., Miranker, W. L.: Computer Arithmetic in Theory and Practice. New York: Academic Press (1981).

[11] Kulisch, U., Miranker, W. L.(eds.): A New Approach to Scientific Computation. New York: Academic Press (1983).

[12] Moore, R. E.: Interval Analysis. Englewood Cliffs, N.J.: Prentice Hall (1966).

[13] Zurmühl, R., Falk, S.: Matrizen und ihre Anwendungen. Teil 2: Numerische Methoden. Springer (1984).