

15. Einführung in LAPACK und BLAS

Im Bereich des wissenschaftlichen Rechnens tauchen einige Aufgaben immer wieder als Teilprobleme auf, insbesondere das Lösen von linearen Gleichungssystemen, lineare Ausgleichsprobleme (kleinste Fehlerquadrate, least squares), Eigenwertberechnung, Singulärwertzerlegung. Es wäre unsinnig jedesmal neue Software zu schreiben die diese Aufgaben erledigt.

Bsp.: Lösen eines linearen Gleichungssystems mittels Gauß-Verfahren/LU-Zerlegung. Während der Grundalgorithmus sehr einfach ist, ist ein numerisch stabile Variante mit Pivotisierung bereits anspruchsvoller. Eine betont effiziente Implementierung erfordert einiges an Arbeit. Sicherzustellen dass das Programm tatsächlich wie gewünscht arbeitet ist ein Problem an sich...

Daher werden für solche Aufgaben Software-Bibliotheken verwendet in denen Algorithmen enthalten sind die von Spezialisten entwickelt wurden, und von vielen Anwendern getestet. Ein Beispiel solcher Bibliotheken sind die meist in Kombination benutzten Bibliotheken BLAS (Basic Linear Algebra Subprograms) und LAPACK (Linear Algebra PACKage), welche bei <http://www.netlib.org/> frei zur Verfügung stehen (oft in Linux/Unix Installationen enthalten). Diese Programme wurden ursprünglich in FORTRAN77 geschrieben, können jedoch auch in den meisten anderen Programmiersprachen benutzt werden, insbesondere C/C++.

Aufbau der Bibliotheken

BLAS und LAPACK sind hierarchisch aufgebaut. D.h. komplexere Aufgaben werden nach Möglichkeit in einfachere zerlegt um Software für die einfachen Aufgaben möglichst oft wiederzuverwenden.

Bezeichnung	allgemeiner Typ	Aufwand	Beispiel
LAPACK (driver)	einfaches Interface		einfache Nutzung von computational
LAPACK (computational)	höhere Aufgaben	ab $O(N)$	LGS lösen, Eigenwerte
LAPACK (auxiliary)	Interna		
BLAS Level 3	Matrix-Matrix	$O(N^3)$ auf $O(N^2)$ Daten	$AB + C$
BLAS Level 2	Matrix-Vektor	$O(N^2)$ auf $O(N^2)$ Daten	$Ax + b, A + cB$
BLAS Level 1	Vektor-Vektor	$O(N)$ auf $O(N)$ Daten	$a^T b, a + b$

Die Meisten Hardware Hersteller stellen für ihre Systeme Optimierte BLAS Varianten (Vendor-BLAS) zur Verfügung die insbesondere bei Level 2 und Level 3 enorme Geschwindigkeitsvorteile gegenüber den generischen Varianten haben. Es gibt aber bereits auch Varianten die automatisch für eine gegebene Rechnerklasse optimiert werden, z.B. ATLAS (Automatically Tuned Linear Algebra Software) <http://math-atlas.sourceforge.net/>. Da LAPACK diese Routinen für alle geeigneten Teilaufgaben nutzt, kann automatisch nahezu optimale Geschwindigkeit erreicht werden. Aus dem selben Grund ist es i.A. eine gute Idee BLAS auch für einfache Aufgaben in eigenen Programmen zu nutzen. (Selbst wenn man nicht von Anfang an optimiertes BLAS benutzt, kann dies später lohnenswert werden.)

Namensgebung der Routinen

Die Bezeichnungen der Routinen wirken auf den ersten Blick kryptisch (z.B. DGEMM, SDOT, DNRM2, CGESV), und sind historisch bedingt auf acht Zeichen begrenzt. Daher folgen sie einem relativ einfachen Schema um Zweck und Eigenschaften zu kodieren. I.A. gibt der erste Buchstabe den Datentyp an:

- S Single precision real, 32 Bit Floating point,
- D Double precision real, 64 Bit Floating point,
- C Single precision complex, 2*32 Bit Floating point,
- Z double precision complex, 2*64 Bit Floating point,
- I (Integer).

Damit werden SDOT (dot product/Skalarprodukt) und DNRM2 (2-Norm) bereits recht eindeutig erkennbar. Bei Routinen für Matrixoperationen folgen Eigenschaften der Matrix, welche teilweise das Speicherformat beeinflussen:

- GE GGeneral, vollbesetzt,
- GB General Bandet, Bandmatrizen,

PP symmetric/Hermitian Positive definite (Packed storage), nur oberes Dreieck gespeichert,

... und einige mehr, siehe z.B. Übersicht LGS Löser, <http://www.netlib.org/lapack/lug/node26.html>

Der verbleibende Name gibt die eigentliche Aufgabe an, z.B. DGEMM Multipliziert zwei Matrizen

$$\alpha * A * B + \beta C \mapsto C$$

(α und β sind Skalare). CGESV löst lineare Gleichungssysteme

$$A * X = B$$

und überschreibt B mit dem Ergebnis X . (B kann mehrere Spalten haben, entspricht mehreren rechten Seiten.)

Die Dokumentation der Routinen steht im wesentlichen im Fortran Quellcode, es existieren aber auch Übersichten z.B. <http://www.netlib.org/lapack/individualroutines.html>.

Dokumentation der Routinen am Beispiel

```

SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
* -- LAPACK driver routine (version 3.1) --
*   Univ. of Tennessee, Univ. of California Berkeley and NAG Ltd..
*   November 2006
*
*   .. Scalar Arguments ..
*   INTEGER          INFO, LDA, LDB, N, NRHS
*   ..
*   .. Array Arguments ..
*   INTEGER          IPIV( * )
*   DOUBLE PRECISION A( LDA, * ), B( LDB, * )
*   ..
*
* Purpose
* =====
*
* DGESV computes the solution to a real system of linear equations
*   A * X = B,
* where A is an N-by-N matrix and X and B are N-by-NRHS matrices.
*
* The LU decomposition with partial pivoting and row interchanges is
* used to factor A as
*   A = P * L * U,
* where P is a permutation matrix, L is unit lower triangular, and U is
* upper triangular. The factored form of A is then used to solve the
* system of equations A * X = B.
*
* Arguments
* =====
*
* N          (input) INTEGER
*            The number of linear equations, i.e., the order of the
*            matrix A.  N >= 0.
*
* NRHS      (input) INTEGER
*            The number of right hand sides, i.e., the number of columns
*            of the matrix B.  NRHS >= 0.
*
* A          (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*            On entry, the N-by-N coefficient matrix A.

```

```

*          On exit, the factors L and U from the factorization
*          A = P*L*U; the unit diagonal elements of L are not stored.
*
* LDA      (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,N).
*
* IPIV     (output) INTEGER array, dimension (N)
*          The pivot indices that define the permutation matrix P;
*          row i of the matrix was interchanged with row IPIV(i).
*
* B        (input/output) DOUBLE PRECISION array, dimension (LDB, NRHS)
*          On entry, the N-by-NRHS matrix of right hand side matrix B.
*          On exit, if INFO = 0, the N-by-NRHS solution matrix X.
*
* LDB      (input) INTEGER
*          The leading dimension of the array B.  LDB >= max(1,N).
*
* INFO     (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*          > 0:  if INFO = i, U(i,i) is exactly zero.  The factorization
*                has been completed, but the factor U is exactly
*                singular, so the solution could not be computed.
*
* =====

```

Nutzung aus Fortran

Bsp: lapack_LGS_solve.f

```

PROGRAM HA7_LAPACK_LGS_solve
implicit none

```

```

C      Loest das LGS
C      Ax=b

```

```

C      Konstanten
integer N,M
parameter (N=3,M=3)

```

```

C      Variablen fuer das Hauptprogramm
double precision A(N,M)
double precision b(N)
integer i,j

```

```

C      Variablen die von DGESV intern benötigt werden
integer ipiv(N)
integer info

```

```

data A / 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 6.0D0,
+       7.0D0, 8.0D0, 10.0D0 /
data b / 1.0D0, 2.0D0, 3.0D0 /

```

```

write (*,*) 'A= '
write (*,99) ((A(i,j), j=1,M), i=1,N)

```

```
        write (*,*)

        write (*,*) 'b= '
        do 15, i = 1,N
            write (*, 98) b(i)
15      continue

        write (*, '(//)')

C      der Funktionsaufruf zum loesen des LGS
        call DGESV(N, 1, A, N, ipiv, b, N, info)

        if (info .NE. 0) then
            write (*,*) 'Fehler in DGESV, INFO=', info
        else
            write (*,*) 'Loesung x ist in b gespeichert '
            write (*,*) 'x='
            write (*, 98) (b(i), i=1,N)

            write (*, '(//)')
            write (*,*) 'A enthaelt jetzt die LU Faktoren der permutierten '
            write (*,*) ' Matrix '
            write (*,*) 'LU= '
            write (*,99) ((A(i, j), j=1,M), i=1,N)
        endif

98      format (' ',E8.2E2)
99      format (3(' ',E8.2E2, ' '))

        END
```

Compilieren+Linken erfolgt mit

```
g77 -o lapack_LGS_solve_f -l lapack -l blas lapack_LGS_solve.f
```

bzw.

```
gfortran -o lapack_LGS_solve_f -l lapack -l blas lapack_LGS_solve.f
```

(neuere Versionen von gcc compiler)

Nutzung aus C

Bsp: lapack_LGS_solve.c

```
/* prototypes of standard functions */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/* BLAS */
void dgemm_ ( char *transa, char *transb, int *m, int *n, int *k,
             double *alpha, double *a, int *lda,
             double *b, int *ldb,
             double *beta, double *c, int *ldc );
```

```
/* LAPACK */

void dpbtrf_ ( char *UPLO, int *N, int *KD, double *AB, int *LDAB,
             int *INFO );

void dpbtrs_ ( char *UPLO, int *N, int *KD, int *NRHS,
             double *AB, int *LDAB, double *B, int *LDB, int *INFO );

void dgbtrf_ ( int *M, int *N, int *KL, int *KU, double *AB, int *LDAB,
             int *IPIV, int *INFO );

void dgbtrs_ ( char *TRANS, int *N, int *KL, int *KU, int *NRHS,
             double *AB, int *LDAB, int *IPIV, double *B, int *LDB,
             int *INFO );

/* DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO ) */
void dgesv_( int* N, int* NRHS, double* A, int* LDA, int* IPIV,
            double* B, int* LDB, int* INFO );

int main()
{
    double A[3*3]={ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 10.0 };
    double b[3]  ={ 1.0, 2.0, 3.0 };
    int ldA=3, n=ldA, m=3;
    int i, j;

    /* Variablen die von DGESV intern benoetigt werden */
    int ipiv[3];
    int info;

    /* wir koennen nicht einfach eine 1 uebergeben, muessen eine
       Variable dafür definieren und diese uebergeben */
    int one=1;

    printf("A=\n");
    for (i=0; i<n; i++)
    {
        for (j=0; j<m; j++)
        {
            printf("_ _%8.2e", A[j*ldA+i]);
        }
        printf("\n");
    }

    printf("\n");

    printf("b=\n");
    for (i=0; i<n; i++)
    {
        printf("_ _%8.2e_\n", b[i]);
    }

    printf("\n");
}
```

/ in Fortran77 werden alle Variablen als Pointer uebergeben
(Pointer auf Speicherstelle wo die Variable gespeichert),
in C passiert das fuer Arrays auch automatisch, bei einfachen
Variablen muss man dies an den Fortran Code per Referenz
uebergeben ("&" Operator)*

*Funktionsname klein schreiben, einen Unterschtrich "_" anhaengen
(kommen "_" im Funktionsnamen selbst vor, muessen zwei "_"
angehaengt werden)*

**/*

/ der Funktionsaufruf zum loesen des LGS */*
dgesv_(&n, &one, A, &n, ipiv, b, &n, &info);

```

if (info != 0)
{
    printf("Fehler in DGESV, INFO=%d\n", info);
}
else
{
    printf("Loesung exist in b gespeichert\n");
    printf("x=\n");
    for (i=0; i<n; i++)
    {
        printf("%8.2e\n", b[i]);
    }

    printf("\n\nA enthaelt jetzt die LU Faktoren der permutierten\n"
           "Matrix.\n");
    printf("LU=\n");
    for (i=0; i<n; i++)
    {
        for (j=0; j<m; j++)
        {
            printf("%8.2e", A[j*ldA+i]);
        }
        printf("\n");
    }
} /* end else */
}

```

Compilieren+Linken erfolgt mit

```
gcc -o lapack_LGS_solve_c -l lapack -l blas -lg2c lapack_LGS_solve.c
```

bzw.

```
gcc -o lapack_LGS_solve_c -llapack -lblas -lgfortran lapack_LGS_solve.c
```

(neuere Versionen von gcc compiler)

Aufgaben

1. Lösen sie das folgende Gleichungssystem

$$Ax = b,$$

$$A = [i^{j-1}]_{i,j=1,\dots,5},$$

$$b = [2 * i + 1]_{i=1,\dots,5}$$

mit DGESV (Fortran oder C Code modifizieren).

2. Lösen sie das folgende Gleichungssystem

$$By = z,$$

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

$$z = [0 \ 0 \ 0 \ 0 \ 6]^T$$

unter Ausnutzung des Eigenschaft das B symmetrisch positiv definit ist. (Entsprechende LAPACK Routine suchen, Fortran oder C Code modifizieren).

3. Berechnen Sie mit den entsprechenden BLAS Routinen

$$a' * b, ||b||, ||a||,$$

$$a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]^T, b = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]^T.$$

4. Bestimmen Sie mit LAPACK die Eigenwerte der Matrizen A und B (siehe oben).
5. Informieren Sie sich zu den weiteren Möglichkeiten der Bibliotheken.