Calculating the Square Root of a Matrix

By Jochen Voss, last updated 2012-02-18

This tutorial explains how to use the LAPACK and BLAS libraries in a C program to calculate the square root (or another function) of a positive semi-definite, symmetric matrix. This serves as a non-trivial example for the use of LAPACK functions. We will see how to call the LAPACK Fortran code from a C program and how to use the BLAS C bindings to multiply matrices. This text assumes that you are already familiar with my text about **numerical linear algebra packages on Linux**. Feedback is very welcome; please direct any comments and suggestions for improvement to **me**.

Contents

Overview

We want to write a program that, for a given symmetric, positive semi-definite matrix A, finds another square matrix B, such that $A=B\cdot B$. In analogy to the square root of a number, this matrix B is called the square root of the matrix A.

To find the matrix B we perform several steps. First we diagonalise A, i.e. we find an orthogonal matrix Z, such that

$$A = Z\cdot D\cdot Z^T,$$

where D is a diagonal matrix. The columns of Z consist of the orthonormal eigenvectors of A and the diagonal elements of D are the corresponding eigenvalues. We will use a LAPACK function for spectral factorisation of a symmetric matrix to obtain D and Z.

Next, we have to find the square root of the diagonal matrix D. Because A is positive semi-definite, all the diagonal elements of D are positive and we can define a new matrix D' by replacing every diagonal element with its square root. This new matrix satisfies

$$D = D'\cdot D',$$

i.e. we have found the square root of D.

Finally we have to calculate

$$A' = Z\cdot D'\cdot Z^T$$

to obtain the square root A' of A. This will be done using the BLAS level 3 routines for matrix-matrix multiplictions.

Strictly speaking each positive number has *two* square roots: a positive one and a negative one. Similarly a symmetric, positive definite $n\times n$-matrix has 2^n distinct square roots. These can be obtained by choosing all possible combinations of plus and minus signs in front of the n diagonal elements of the matrix D'. Here we will ignore this ambiguity and only calculate the unique positive semi-definite square root.

Diagonalising a Matrix

We can find the LAPACK functions to diagonalise matrices on the **Standard Eigenvalue and Singular Value Problems** page of the LAPACK Users' Guide. Because our matrix is real val-

ued and symmetric we choose the DSYEVR function from **table 2.5** there.

The Fortran function DSYEVR can be accessed within C programs as `dsyevr_` and we can find the argument list in the initial comment of the Fortran source file the **dsyevr.f** in the netlib archive. We use the following wrapper code to conveniently access the function from C.

```
int
dsyevr(char JOBZ, char RANGE, char UPLO, int N,
       double *A, int LDA, double VL, double VU,
       int IL, int IU, double ABSTOL, int *M,
       double *W, double *Z, int LDZ, int *ISUPPZ,
       double *WORK, int LWORK, int *IWORK, int LIWORK)
{
  extern void dsyevr_(char *JOBZp, char *RANGEp, char *UPLOp, int *Np,
                      double *A, int *LDAp, double *VLp, double *VUp,
                      int *ILp, int *IUp, double *ABSTOLp, int *Mp,
                      double *W, double *Z, int *LDZp, int *ISUPPZ,
                      double *WORK, int *LWORKp, int *IWORK, int *LIWORKp,
                      int *INFOp);
  int INFO;
  dsyevr_(&JOBZ, &RANGE, &UPLO, &N, A, &LDA, &VL, &VU,
          &IL, &IU, &ABSTOL, M, W, Z, &LDZ, ISUPPZ,
          WORK, &LWORK, IWORK, &LIWORK, &INFO);
  return INFO;
}
```

The most important arguments are shortly described in the following list. Full information can be found in the Fortran source file.

- A is the input matrix, its memory layout is specified by the parameters N and LDA.
- The list of eigenvalues is returned in the array W.
- The eigenvectors are returned in the matrix Z.
- Because the input matrix is symmetric we need to specify only half of it. UPLO specifies which half of the input matrix is used.

Fortran Array Conventions

The matrix arguments in the function `dsyevr` are of type `double *`; they are pointers to the top-left element of the matrix. The matrices are expected to be in column-major order, i.e. the elements of each column must be adjacent in memory. This is different from the usual C language convention for two-dimensional arrays. We will use the following helper functions to access the elements. As usual the index `i` is used for the row, and `j` is used for the column. The top left element of the matrix is accessed by setting `i=0` and `j=0`. The value `N` is the number of rows of the matrix A.

```
void
set_entry(double *A, int i, int j, double val)
{
  A[j*N+i] = val;
}

double
get_entry(const double *A, int i, int j)
{
  return A[j*N+i];
}
```

Note that Fortran uses array indices starting at 1. Since this does not affect the data-layout in memory we can ignore this and use the C indexing conventions (i.e. first entry has index 0) in our code.

Multiplying Matrices

We will use the CBLAS function `cblas_dgemm` to calculate matrix products. The function is declared in the header file `<cblas.h>`, the arguments are described in the CBLAS reference document (file `/usr/share/doc/atlas3-doc/cblas.ps.gz` on Debian Linux) and in the **dgemm.f** Fortran source file.

The CBLAS functions allow for matrices in either row-major order or column-major order. The use of `dsyevr` to calculate the eigenvectors forces us to choose column-major order here.

The call to calculate C=A*B for N×N matrices turns out to be

```
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
            N, N, N, 1, A, N, B, N, 0, C, N);
```

Calculating the Square Root

The following code (file **matroot.c**) puts everything together to calculate the square root of a matrix.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <cblas.h>


static const int N = 7;

<em>... add the code from above here ...</em>

static double
dlamch(char CMACH)
{
  extern double dlamch_(char *CMACHp);
  return dlamch_(&CMACH);
}

int
main ()
{
  double *A, *B, *W, *Z, *WORK;
  int *ISUPPZ, *IWORK;
  int  i, j;
  int  M;

  /* allocate and initialise the matrix */
  A = malloc(N*N*sizeof(double));
  for (i=0; i<N; ++i) {
    for (j=0; j<i-1; ++j) {
      set_entry(A, i, j, 0);
    }
  }
  for (i=0; i<N-1; ++i)  set_entry(A, i+1, i, -1);
  for (i=1; i<N-1; ++i)  set_entry(A, i, i, 2);
  set_entry(A, 0, 0, 1);
  set_entry(A, N-1, N-1, 1);

  /* allocate space for the output parameters and workspace arrays */
  W = malloc(N*sizeof(double));
  Z = malloc(N*N*sizeof(double));
  ISUPPZ = malloc(2*N*sizeof(int));
  WORK = malloc(26*N*sizeof(double));
  IWORK = malloc(10*N*sizeof(int));

  /* get the eigenvalues and eigenvectors */
  dsyevr('V', 'A', 'L', N, A, N, 0, 0, 0, 0, dlamch('S'), &M,
         W, Z, N, ISUPPZ, WORK, 26*N, IWORK, 10*N);

  /* allocate and initialise a new matrix B=Z*D */
  B = malloc(N*N*sizeof(double));
  for (j=0; j<N; ++j) {
    double  lambda=sqrt(W[j]);
    for (i=0; i<N; ++i) {
      set_entry(B, i, j, get_entry(Z,i,j)*lambda);
    }
  }

  /* calculate the square root A=B*Z^T */
  cblas_dgemm(CblasColMajor, CblasNoTrans, CblasTrans, N, N, N,
              1, B, N, Z, N, 0, A, N);

  /* emit the result */
  for (i=0; i<N; ++i) {
    for (j=0; j<N; ++j) {
      double  x = get_entry(A, i, j);
```

```
      printf("%6.2f", x);
    }
    putchar('\n');
  }
  putchar('\n');

  /* check the result by calculating A*A */
  memcpy(B, A, N*N*sizeof(double));
  cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, N, N, N,
              1, A, N, B, N, 0, Z, N);

  for (i=0; i<N; ++i) {
    for (j=0; j<N; ++j) {
      double  x = get_entry(Z, i, j);
      printf("%6.2f", x);
    }
    putchar('\n');
  }

  return 0;
}
```

To compile this program we use the following command.

```
cc matroot.c -o matroot -llapack -lblas -lm
```

The program has the following output.

```
 0.84 -0.52 -0.13 -0.07 -0.05 -0.04 -0.03
-0.52  1.23 -0.46 -0.11 -0.06 -0.04 -0.04
-0.13 -0.46  1.25 -0.45 -0.11 -0.06 -0.05
-0.07 -0.11 -0.45  1.25 -0.45 -0.11 -0.07
-0.05 -0.06 -0.11 -0.45  1.25 -0.46 -0.13
-0.04 -0.04 -0.06 -0.11 -0.46  1.23 -0.52
-0.03 -0.04 -0.05 -0.07 -0.13 -0.52  0.84

 1.00 -1.00  0.00 -0.00  0.00  0.00 -0.00
-1.00  2.00 -1.00 -0.00 -0.00  0.00  0.00
 0.00 -1.00  2.00 -1.00 -0.00 -0.00  0.00
-0.00 -0.00 -1.00  2.00 -1.00  0.00 -0.00
 0.00 -0.00 -0.00 -1.00  2.00 -1.00  0.00
 0.00  0.00 -0.00  0.00 -1.00  2.00 -1.00
-0.00  0.00  0.00 -0.00  0.00 -1.00  1.00
```

The first matrix is the result: the square root of the original matrix. To verify that everything worked correctly, we multiply this matrix with itself. The result is emitted as the second matrix, which is indeed identical to the original one. The strange signs in front of the zeros are caused by small rounding errors.

Conclusion

In this tutorial we have learnt how we can combine functions from the LAPACK and BLAS libraries to solve a non-trivial problem from linear algebra, namely to calculate the square root of a symmetric, positive semi-definite matrix.

The same method can be easily used to calculate other functions of the matrix. For example to get the exponential of the matrix we would just apply the function `exp` instead of `sqrt` to the eigenvalues.

References

- my text about **numerical linear algebra packages on Linux**
- the **LAPACK homepage**, containing the **LAPACK user's guide**
- my **numerical linear algebra lecture notes**