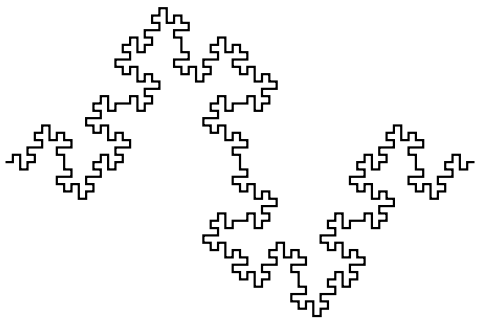
How to use BLAS and LAPACK with a C Languages Programming

Henry R. Moncada
University of Texas at El Paso



February 22, 2023

Contenido

Linear Algebra

The purpose of this presentation is to show you how to use **BLAS /LAPACK** functions in your own C/C++ or Fortran code. For this, Let us explain how to solves a system of linear equations.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= -4 \\ 2x_1 + 3x_2 + 4x_3 &= -1 \\ 3x_1 + 4x_2 + x_3 &= -2 \end{aligned}$$

Solving this system of linear equations, we get the following solutions:

$$(x_1, x_2, x_3) = (11, -9, 1)$$

Linear Algebra

Rewriten the system of linear equations in a matrix/array form

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -4 \\ -1 \\ -2 \end{bmatrix}$$

We can represented the system using coefficients:

- A is coefficient matrix,
- b is the vector containing the right sides of equations,
- x is the vector containing the solutions of equations.

$$A\,x = b$$

Linear Algebra

$$A\ x = b$$

The square matrix A is invertible, if there exists a matrix A^{-1} such that

$$A^{-1}\ A = A\ A^{-1} = I$$

where I is the identity matrix.

$$\text{If } \exists\ A^{-1} \quad \Leftrightarrow \quad det(A) \neq 0$$

The solution of the system of linear equations can be solved as following

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}\ b \\ x &= A^{-1}\ b \Rightarrow x = A\backslash b \end{aligned}$$

MATLAB AND INTEL MATH KERNEL LIBRARY (MKL)

- Intel MKL is a library of optimized math routines for science, engineering, and financial applications.
- Core math functions include BLAS, LAPACK, ScaLAPACK, sparse solvers, fast Fourier transforms, and vector math.
- The routines in MKL are hand-optimized specifically for Intel processors.
- Utilizes standard C and Fortran APIs for compatibility with BLAS, LAPACK and FFTW functions from other math libraries.
- It is available with both free community-supported and paid support licenses.
- MATLAB uses the Intel Math Kernel Library (MKL) behind the scenes to perform many linear algebra operations.

What version of the MKL was being used by MATLAB. Let's ask MATLAB

```
 >> version -lapack
ans =
Intel(R) Math Kernel Library Version 10.3.5 Product Build 20110720 for
Intel(R) 64 architecture applications

>> version -blas
ans =
Intel(R) Math Kernel Library Version 10.3.5 Product Build 20110720 for
Intel(R) 64 architecture applications
```

BLAS, AND LAPACK

- **BLAS (Basic Linear Algebra Subprograms)** is a collection of low-level matrix and vector arithmetic operations, for performing common linear algebra operations such as vector addition, scalar multiplication, dot products, linear combinations, and matrix multiplication.
- **LAPACK (Linear Algebra Package)** is a collection of higher-level linear algebra operations. It provides routines for solving systems of linear equations such as matrix factorizations (LU, LLt, QR, SVD, Schur, etc), linear least squares, eigenvalue problems, and singular value decomposition. LAPACK is built on top of the BLAS; many users of LAPACK only use the LAPACK interfaces and never need to be aware of the BLAS at all. LAPACK is generally compiled separately from the BLAS, and can use whatever highly-optimized BLAS implementation you have available.

BLAS

- Specified set of low-level subroutines that perform common linear algebra operations such as copying, vector scaling, vector dot products, linear combinations, and matrix multiplication
  - Level 1: Contains vector operations on strided arrays, dot products, vector norms, a generalized vector addition of the form

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \boldsymbol{y}$$

  - Level 2: Contains matrix-vector operations including a generalized matrix-vector multiplication (GEMV):

$$\boldsymbol{y} \leftarrow \alpha A \boldsymbol{x} + \beta \boldsymbol{y}$$

  - Level 3: Contains matrix-matrix operations, including a general matrix multiplication (GEMM) of the form

$$\boldsymbol{C} \leftarrow \alpha AB + \beta C$$

    where A and B can optionally be transposed inside the routine and all three matrices may be strided

LAPACK

- CLAPACK routines have names of the form *XYYZZZ*.
  - The first letter *X* indicates the data type the routine expects. *S* Real, *D* Double precision,*C* Complex, *Z* Double complex, the same as double in C
  - The letters *YY* tell the type of matrices the routine deals with. *GE* and *TR*, meaning general and triangular, are the ones we will use.
  - *ZZZ* is the name of the actual computation done by a particular subroutine, e.g., *SV* denotes factor the matrix and solve a system of equations, *LS* denotes solve over or undetermined linear system using orthogonal factorization.
  - For example,
    - **SGEBRD** is a single precision (S) routine that performs a bidiagonal reduction (BRD) of a real general matrix (GE).
    - **DGEEVX** is a double precision (D) routine that computes the eigenvalues and, optionally, the left and/or right eigenvectors (EVX) for real general matrix (GE), X stand for expert

How to implement BLAS/LAPACK

- LAPACK and BLAS subroutines are written in Fortran.
- Difference between **C** and **Fortran**
  - The way matrices are **stored in memory**

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

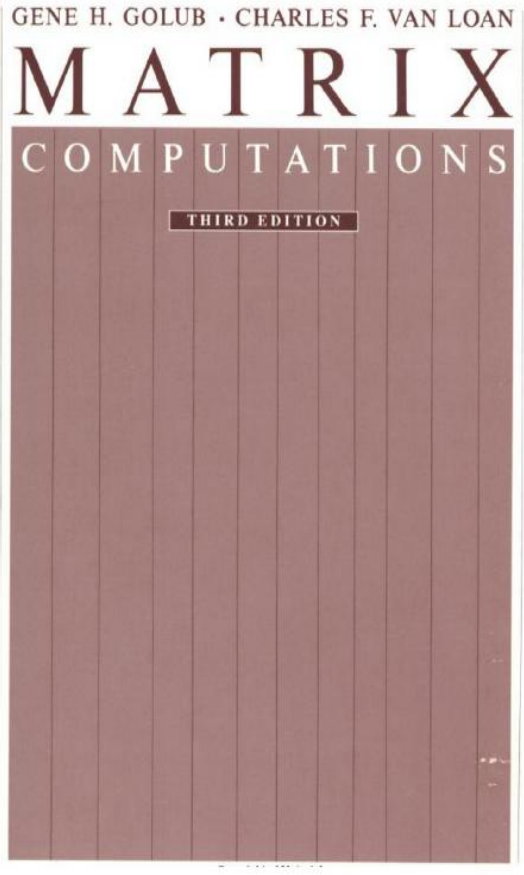
  - In **C** matrices are stored in **row major order**. Would be stored as, `{a b c d e f g h i}`
  - In **Fortran** matrices are stored in **column major order**. Would be stored as, `{a d g b e h c f i}`

How arguments are passed to a subroutine

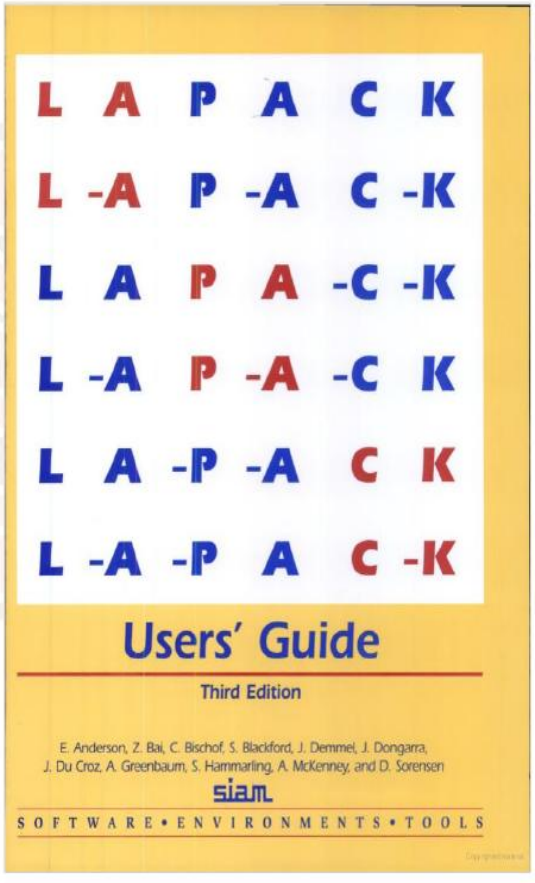
The way arguments are passed to a subroutine.

- Pass-by-Value:
  - In C pass-by-value, notice NO & and $*$
  - A duplicate copy of original parameter is passed in
  - No effect on the original parameter in main(). So if the data passed (that is stored in the function variable) is modified inside the function program.
- Pass-by-Reference :
  - Also called pass by address, a copy of the address (pointer) of the actual parameter is stored
  - Original parameter gets affected if value of parameter changed inside function.
  - Because a pointer is copied, if the value at that pointers address is changed in the function program, the value is also changed in main()

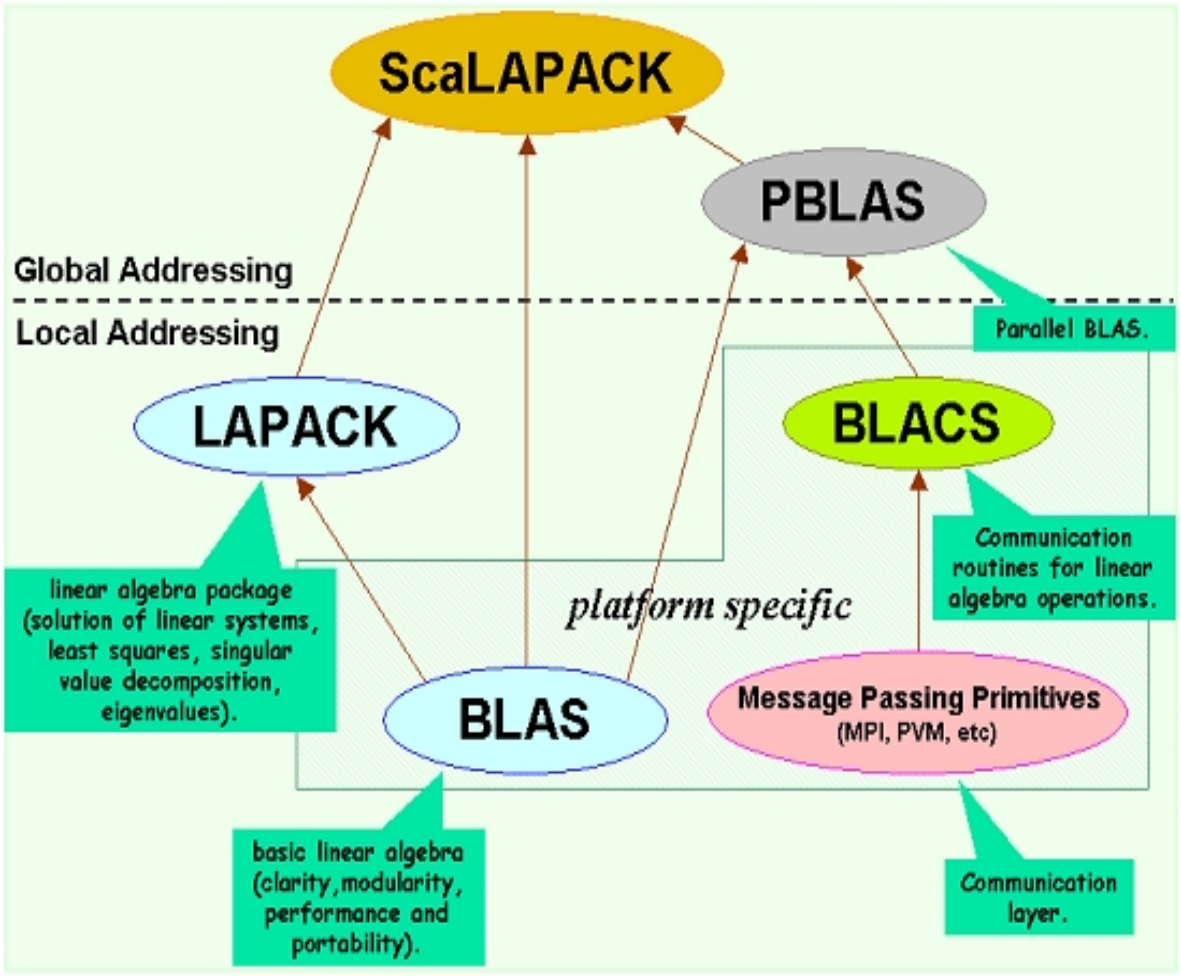
Math Background



Implementation



Hierarchy Libraries



Example : Exponential Matrix

$$y = \mathrm{e}^{\mathrm{A}}$$

Example 1: A is Symmetric, then all of its eigenvalue are real

Example 2: A is Real, then eigenvalue are complex

Exponential Matrix

Consider the Taylor series of exponential

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^2}{3!} + \cdots + \frac{x^p}{p!} + \cdots$$

given a square matrix A we can define the matrix exponential as follows

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!} = I + A + \frac{A^2}{2!} + \frac{A^2}{3!} + \cdots + \frac{A^p}{p!} + \cdots$$

Symmetric Matrix

Let be $A \in \mathbb{R}^{n\times n}$ symmetric $A = A^T$, then the matrix has a complete set of linear independent eigenvectors $v_1, v_2, \cdots, v_n$

$$A\ v_k = \lambda_k v_k, \qquad\qquad k = 1, 2, \cdots, n$$

Thus, defining the matrix $T = [v_1, v_2, \cdots, v_n]$ whose columns are the eigenvectors we have

$$AT = [A\ v_1, A\ v_2, \cdots, A\ v_n] = [\lambda_1 v_1, \lambda_2 v_2, \cdots, \lambda_n v_n] = T\Lambda$$

and thus $A = T\Lambda T^{-1}$ where

$$\Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$$

Using $A = T\Lambda T^{-1}$ we can write

$$\mathrm{e}^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k = T \left(\sum_{k=0}^{\infty} \frac{1}{k!} \Lambda^k \right) T^{-1} = T\mathrm{e}^{\Lambda} T^{-1}$$

$$\mathrm{e}^A = T \begin{pmatrix} \mathrm{e}_1^{\lambda} & & & \\ & \mathrm{e}_2^{\lambda} & & \\ & & \ddots & \\ & & & \mathrm{e}_n^{\lambda} \end{pmatrix} T^{-1}$$

Symmetric Matrix - Solve with Matlab

Symmetric Matrix - Solve with Matlab

```
>> A = [5 4; 4 5]
A =

   4   5
   5   4

>> expm(A)
ans =

   4052.9   4050.2
   4050.2   4052.9

>> [v lambda]=eig(A)
v =

  -0.70711   0.70711
   0.70711   0.70711

lambda =

   1   0
   0   9

>> v*expm(lambda)*inv(v)
ans =

   4052.9   4050.2
   4050.2   4052.9
```

SYMMETRIC MATRIX - SOLVE WITH MATLAB

```
>> A = [5 4; 4 5]
A =

   4   5
   5   4

>> expm(A)
ans =

   4052.9   4050.2
   4050.2   4052.9

>> [v lambda]=eig(A)
v =

  -0.70711   0.70711
   0.70711   0.70711

lambda =

   1   0
   0   9

>> v*expm(lambda)*inv(v)
ans =

   4052.9   4050.2
   4050.2   4052.9
```

```
>>  A = [ 1.96   -6.49  -0.47   -7.20   -0.65;
          -6.49    3.80  -6.39    1.50   -6.34;
          -0.47   -6.39   4.17   -1.51    2.67;
          -7.20    1.50  -1.51    5.70    1.80;
          -0.65   -6.34   2.67    1.80   -7.10; ]
A =

    1.96000  -6.49000  -0.47000  -7.20000  -0.65000
   -6.49000   3.80000  -6.39000   1.50000  -6.34000
   -0.47000  -6.39000   4.17000  -1.51000   2.67000
   -7.20000   1.50000  -1.51000   5.70000   1.80000
   -0.65000  -6.34000   2.67000   1.80000  -7.10000

>> expm(A)

ans =

    2.3433e+06  -2.8945e+06   1.9081e+06  -2.1849e+06   7.7534e+05
   -2.8945e+06   3.5800e+06  -2.3616e+06   2.6972e+06  -9.6022e+05
    1.9081e+06  -2.3616e+06   1.5583e+06  -1.7775e+06   6.3382e+05
   -2.1849e+06   2.6972e+06  -1.7775e+06   2.0376e+06  -7.2207e+05
    7.7534e+05  -9.6022e+05   6.3382e+05  -7.2207e+05   2.5787e+05
```

SYMMETRIC MATRIX - SOLVE WITH C

- Allocate memory space for the input/output parameters and workspace arrays, A must be real symmetric matrix (* require memory allocation - malloc).

$$*A, *B, *W, *Z, *WORK, *ISUPPZ, *IWORK$$

- Compute the eigenvalues and eigenvectors of A.

$$(W, Z)$$

Setup Lapack library **dsyevr**

```
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 matrix $B = Z * D$.
- Compute

$$D = exp(W_j)$$

- Compute $B = Z * D$
- Compute the product, using Cblas library **cblas_dgemm**

$$A = B * Z^T$$

```
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasTrans, N, N, N, 1, B, N, Z, N, 0, A, N)
```

- Print results $C = expm(A) = Z * D * Z^T$

SYMMETRIC MATRIX - C INTERFACE

LAPACK C wrapper, link your program with the `lapack_wrapper.a` library in the lib directory.

```
static 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;
}

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

LAPACK - DSYEVR

```
dsyevr('V', 'A', 'L', N, A, N, 0, 0 0, 0, dlamch('S'), &M, W, Z, N, ISUPPZ, WORK, 26*N,
       IWORK, 10*N);
```

```
subroutine dsyevr(
    character                              JOBZ,
    character                              RANGE,
    character                              UPLO,
    integer                                N,
    double precision, dimension(lda,*)     A,
    integer                                LDA,
    double precision                       VL,
    double precision                       VU,
    integer                                IL,
    integer                                IU,
    double precision                       ABSTOL,
    integer                                M,
    double precision, dimension(*)         W,
    double precision, dimension(ldz,*)     Z,
    integer                                LDZ,
    integer, dimension(*)                  ISUPPZ,
    double precision, dimension(*)         WORK,
    integer                                LWORK,
    integer, dimension(*)                  IWORK,
    integer                                LIWORK,
    integer                                INFO
)
```

LAPACK- DSYEVR

BLAS - DGEMM

$$C := \alpha * A * B + \beta * C$$

```
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasTrans, N, N, N, 1, B, N, Z, N, 0, A, N)
```

```
subroutine dgemm (
  character                                TRANSA,
  character                                TRANSB,
  integer                                  M,
  integer                                  N,
  integer                                  K,
  double precision                         ALPHA,
  double precision, dimension(lda,*)       A,
  integer                                  LDA,
  double precision, dimension(ldb,*)       B,
  integer                                  LDB,
  double precision                         BETA,
  double precision, dimension(ldc,*)       C,
  integer                                  LDC
)
```

BLAS - DGEMM

Real Matrix - Solve with Matlab

```
>> A = [-1.01   0.86  -4.60   3.31  -4.81;
         3.98   0.53  -7.04   5.29   3.55;
         3.30   8.26  -3.89   8.20  -1.51;
         4.43   4.96  -7.66  -7.33   6.18;
         7.31  -6.43  -6.16   2.47   5.58]

>> [V l] = eig(A)
V =

   0.10806 + 0.16865i   0.10806 - 0.16865i   0.73223 + 0.00000i   0.73223 - 0.00000i   0.4606
   0.40631 - 0.25901i   0.40631 + 0.25901i  -0.02646 - 0.01695i  -0.02646 + 0.01695i   0.3377
   0.10236 - 0.50880i   0.10236 + 0.50880i   0.19165 - 0.29257i   0.19165 + 0.29257i   0.3087
   0.39863 - 0.09133i   0.39863 + 0.09133i  -0.07901 - 0.07808i  -0.07901 + 0.07808i  -0.7438
   0.53954 + 0.00000i   0.53954 - 0.00000i  -0.29160 - 0.49310i  -0.29160 + 0.49310i   0.1585

l =
    2.85813 + 10.76275i                      0                      0                      0
                      0    2.85813 - 10.76275i                      0                      0
                      0                      0   -0.68667 +  4.70426i                      0
                      0                      0                      0   -0.68667 -  4.70426i
                      0                      0                      0                      0
```

Real Matrix - Solve with Matlab

```
>> B=V*expm(l)
B =

   2.42549 - 2.51083i   2.42549 + 2.51083i  -0.00299 - 0.36848i  -0.00299 + 0.36848i   0.0000
  -6.02638 - 5.84897i  -6.02638 + 5.84897i  -0.00842 + 0.01339i  -0.00842 - 0.01339i   0.0000
  -9.04024 + 0.31021i  -9.04024 - 0.31021i  -0.14801 - 0.09525i  -0.14801 + 0.09525i   0.0000
  -3.15192 - 6.39298i  -3.15192 + 6.39298i  -0.03897 + 0.04008i  -0.03897 - 0.04008i  -0.0000
  -2.16965 - 9.14981i  -2.16965 + 9.14981i  -0.24695 + 0.14876i  -0.24695 - 0.14876i   0.0000

>> C =B*inv(V)
C =

   -0.50217 +  0.00000i    4.56041 +  0.00000i    2.01829 +  0.00000i    2.38722 +  0.00000i
   -6.36228 +  0.00000i   -4.83109 +  0.00000i   12.28998 +  0.00000i   -2.47319 +  0.00000i
   -3.85152 -  0.00000i  -13.76345 -  0.00000i    6.27992 -  0.00000i   -6.46124 +  0.00000i
   -5.50963 +  0.00000i   -0.11275 +  0.00000i   10.85609 +  0.00000i   -0.33509 +  0.00000i
   -7.10558 +  0.00000i    3.60664 +  0.00000i   13.60548 +  0.00000i    1.03809 +  0.00000i

>> expm(A)
ans =

   -0.50217    4.56041    2.01829    2.38722   -0.98493
   -6.36228   -4.83109   12.28998   -2.47319   -6.76137
   -3.85152  -13.76345    6.27992   -6.46124   -2.03676
   -5.50963   -0.11275   10.85609   -0.33509   -6.46545
   -7.10558    3.60664   13.60548    1.03809   -8.66237
```

Real Matrix

Let be $A \in \mathbb{R}^{n\times n}$, then the matrix exponential can be computed starting from Jordan normal form (or Jordan canonical form).
Theorem (Jordan normal form) Any square matrix $A \in \mathbb{R}^{n\times n}$, is similar to a block diagonal matrix J, i.e. $T^{-1}AT = J$ where

$$J = \begin{pmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_m \end{pmatrix} \quad \text{and} \quad J_k = \begin{pmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{pmatrix} = \lambda_k I + N$$

The column of $T = [t_{1,1}, t_{1,2}, \ldots, t_{m,n_m}, t_{m,n_{m-1}}]$ are generalized eigenvectors, i.e.

$$At_{k,j} = \begin{cases} \lambda_k t_{k,j} & \text{if} \quad j = 1 \\ \lambda_k t_{k,j} + t_{k,j-1} & \text{if} \quad j > 1 \end{cases}$$

$$\mathrm{e}^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k = \sum_{k=0}^{\infty} \frac{1}{k!} \left(T\ \Lambda^k\ T^{-1} \right)^k = T \begin{pmatrix} \mathrm{e}^{J_1} & & & \\ & \mathrm{e}^{J_2} & & \\ & & \ddots & \\ & & & \mathrm{e}^{J_m} \end{pmatrix} T^{-1}$$

CODE DESCRIPTION IMPLEMENTATION

- Allocate memory space for the input/output parameters and workspace arrays, A must be real (* require memory allocation - malloc).

$$*A, *WR, *WI, *VL, *VR, *WORK$$

- Compute the eigenvalues (WR, WI) and left and right eigenvectors (VL, VR)

$$(WR, WI, VL, VR)$$

Setup Lapack library **dgeev**

```
dgeev('V', 'V', N, A, N, WR, WI, VL, N, VR, N, WORK, 4*N);
```

- Allocate memory for $(\exp(WR), \exp(WI))$ and compute

$$D = exp(WR + I * WI)$$

- Allocate memory for $B = VR * D$ and compute

$$B = VR * D$$

- Find the inverse matrix of VR^{-1}
- Compute the product $B * VR^{-1}$, using Cblas library **cblas_zgemm**

$$C = \alpha * A * B + \beta * C = 1 * B * invVR + 0 * C*$$

```
cblas_zgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, m, n, k, &alpha, B, LDA,
            invVR, LDB, &beta, C, LDC);
```

- Print results $C = expm(A) = B * VR^{-1} = VR * D * VR^{-1}$

CODE DESCRIPTION IMPLEMENTATION

Find the inverse matrix of VR^{-1}

- Allocate memory space for

  $$*IPIV, *WORK$$

- LU factorization

```
zgetrf_(&N, &N, A, &N, IPIV, &INFO);
```

- Inverse

```
zgetri_(&N, A, &N, IPIV, WORK, &LWORK, &INFO);
```

- Print results VR^{-1}

Real Matrix - C interface

LAPACK C wrapper, link your program with the `lapack_wrapper.a` library in the lib directory.

```
static int dgeev( char JOBVL, char JOBVR, int N, double *A, int LDA, double *WR, double *WI,
                  double *VL, int LDVL, double *VR, int LDVR, double *WORK, int LWORK){

   extern void dgeev_( char *JOBVLp, char *JOBVRp, int *Np, double *A, int *LDAp,
                       double *WR, double *WI, double *VL, int *LDVLp, double *VR,
                       int *LDVRp, double *WORK,int *LWORKp, int *INFOp );

   int INFO;

   dgeev_(&JOBVL, &JOBVR, &N, A, &LDA, WR, WI, VL, &LDVL, VR, &LDVR, WORK, &LWORK, &INFO);

   return INFO;
}
```

Generate LU decomposition of a general matrix

```
extern void zgetrf_(int* M, int *N, double complex* A, int* lda, int* IPIV, int* INFO);
```

Generate the inverse matrix given a LU decomposition of the matrix

```
extern void zgetri_(int* N, double complex* A, int* lda, int* IPIV, double complex* WORK,
                    int* lwork, int* INFO);
```

Use BLAS library $C = \alpha * A * B + \beta * C = 1 * B * invVR + 0 * C$

```
cblas_zgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, m, n, k, &alpha, BB, LDA, invVR,
            LDB, &beta, C, LDC);
```

For further references see :

- Using Lapack Routines in Fortran Programs
- Using Lapack Routines in C Programs
- LAPACK EXAMPLES - Example Programs for LAPACK
- LAPACK: Linear Algebra PACKage
- Linear Algebra PACKage
- The LAPACKE C Interface to LAPACK
- Calling LAPACK, BLAS, and CBLAS Routines from C/C++ Language Environments
- example_DGESV_colmajor.c File Reference
- example_DGESV_rowmajor.c
- Using LAPACK from C
- Intel® Math Kernel Library – Documentation
- Math Kernel Library
- Calculating the Square Root of a Matrix
- Function pass by value vs. pass by reference
- C Tutorial – Call by Value or Call by Reference
- Argument passing conventions
- LAPACK- DSYEVR
- DSYEVR Example
- C wrapper for LAPACK
- BLAS - DGEMM
- Using BLAS from C with row major data
- cblas_?gemm

END

QUESTIONS ???

Temporary page!

LaTeX was unable to guess the total number of pages correctly
As there was some unprocessed data that should have been
added to the final page this extra page has been added to
receive it.
If you rerun the document (without altering it) this surplus
page will go away, because LaTeX now knows how many pag
to expect for this document.