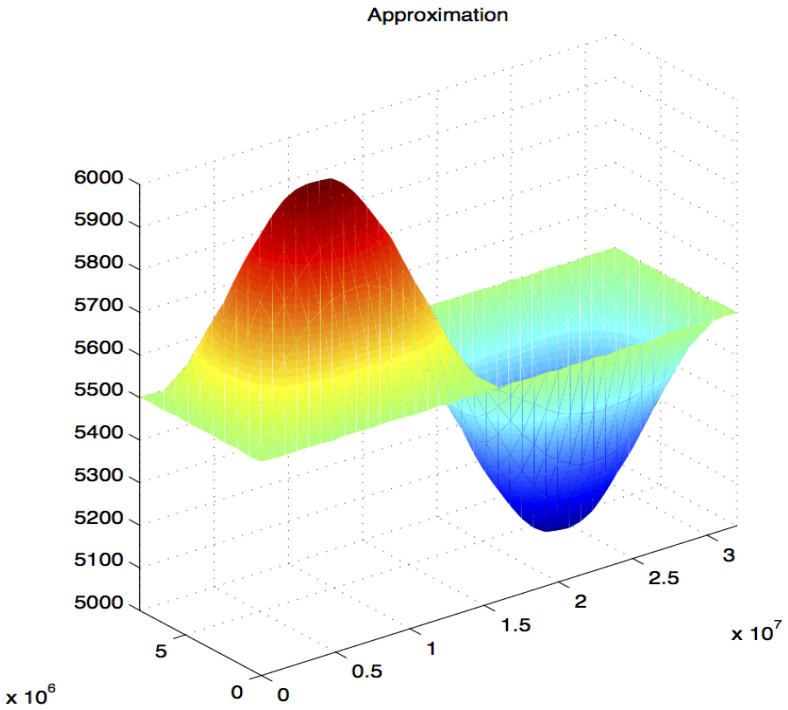
Projet d'Informatique et Mathématiques Appliquées
en
Algèbre Linéaire Numérique

Model reduction approaches for PDE problems (Phase 2)



Contents

1 Introduction

The second phase of this project is mostly concerned with the implementation and evaluation of efficient algorithms for computing a limited number of eigenpairs as described in the following two sections.

2 Improved subspace iteration algorithms

Part of the Phase 2 of the project consists in developing a Fortran implementation of the algorithm described in the Phase 1 subject. The base for these development is the provided Fortran code which contains the following components:

1. a driver program in the file `main.f90` which generates a matrix (according to different parameters) and then uses subspace iteration variants of the algorithm 1 to compute its eigenvalues and eigenvectors, or its singular values and left singular vectors;
2. the subspace iteration method (version "ev") that compute eigenvalues and eigenvectors of a square matrix in the `m_subspace_iter.F90` file;
3. a template of the version "sv" that computes singular values and left singular vectors of a (possibly rectangular) matrix in the `m_subspace_iter.F90` file. **The provided template includes a copy of the version for square matrices (comment lines): it is left to the students to modify this template in order to compute singular values and left singular vectors. Attention should be given to the dimensions of matrices and vectors. Furthermore, the interface of the subroutine should not be modified. Please also note that, in the interface, the name of some variables has changed with respect to "ev" routine**.

The students are supposed to modify only the `m_subspace_iter.F90` file; the `main.f90` file as well as all the other provided files should not be modified. The code can be compiled by typing the `make` command. This will generate an executable file named `main` which executes the driver program. Upon execution, the `main` program 1) asks a number of parameters, 2) generates a matrix, 3) executes one subspace iteration method on the generated matrix and 4) optionally prints the computed eigenvalues on the screen. The asked parameters are, in order:

1. the method variant (0 for version "ev", 1 for version "sv", 2 for LAPACK direct solver);
2. the value of p (number of products per iteration);
3. the number of rows m of the A matrix whose singular values and left singular vectors have to be computed.
4. the number of columns n of the A matrix.
5. the type of matrix to be generated: the driver program can generate 4 types of matrices. Each type has different spectral properties and will therefore yield a different convergence behavior for the subspace iteration variants;
6. the dimension l of the invariant subspace;
7. the amount of information requested, to be expressed as a value between 0 and 1; for instance a value of 0.05 means that the method will stop after finding singular value σ_k such that $\frac{\sigma_k}{\sigma_1} < 1-0.05$ which basically amounts to retaining a number of singular values accounting for 95% of the information.
8. a parameter which defines whether the computed singular values should be printed or not.

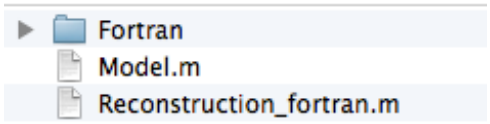
The provided code is guaranteed to compile and execute correctly on the computers in the TP rooms.

Appendix 7 explains the interfaces of the routines used for performing matrix-matrix product and the eigenvalue decomposition operations. These routines are used in both subspace iteration variants. The usage of the Fortran builtin function `matmul` for computing the matrix product **is not admitted**.

3 Model reduction strategies in PDE problems and an application

Once the Fortran code of the "sv" version is developed as required above, it can be directly used in the Matlab code used for the experiments in Phase 1. In order to do so, follow these steps:

1. Make sure that the directory containing the Fortran files described in the previous section is located next to the Matlab files.



2. Copy the provided `mexopts.sh` file in the Matlab configuration directory of your account by executing the following command:

   ```
   $ cp /mnt/n7fs/ens/tp_abuttari/mexopts.sh ~/.matlab/R2014a/
   ```

   If the directory does not exist, create it.

3. Make the `PATH` environment variable point to the location where the Matlab `mex` command is located by issuing the following command:

   ```
   $ export PATH=/applications/matlab/bin:$PATH
   ```

4. Compile the mexfile needed to interface the Fortran and the Matlab codes by executing the following command in the directory containing the Fortran codes:

   ```
   $ make mexfile
   ```

5. Before launching Matlab, set the `OMP_NUM_THREADS` to 4. By doing this, the Fortran code will use all the four cores available on the TP machines for performing matrix products. This is done to make the comparison with the `svd` function fair since Matlab, by default, uses all the cores available on the machine:

   ```
   $ export OMP_NUM_THREADS=4
   $ matlab
   ```

4 Deliverables phase : 2 developments and numerical experiments

The version of the eigensolver (described in the Phase 1 subject) should be written in Fortran as explained in Section 2.

The code should be well documented and structured (as suggested in sofware lectures). **The readability of the code will be seriously taken into account in the evaluation**.

Deliverables for the second part of the project include:

- A final report

  1. Summary of the work done in this project (1 page max)
  2. Detailed description of the final singular values and left singular vectors solver algorithm. All numerical and algorithmic modifications with respect to Algorithm 1 and with respect to the algorithms proposed in the report at Phase 1 should be explained/justified (3 pages max).

3. A summary of the experiments (maximum of 5 pages, results should be analyzed and commented). The version should be compared with the other ones on synthetic data in terms of both efficiency and accuracy. One could, for example, analyse the influence of increasing values on the block size on the performance. Do not hesitate to work with high dimensional matrices ($m > 1000$).
4. A summary of the evaluation of the algorithm within the Matlab code used for the reconstruction scenario. Describe whether and how the use of the algorithm helps improving both the speed and the accuracy of the data reconstruction application with respect to the `svd` function used in Phase 1. Present experimental results that illustrate your conclusions (2 pages max).
5. Concluding remarks (1 page max).

- A file (**pdf format, any other format (doc, ppt, odt etc) will not be accepted**) of presentation will be used during the oral examination (maximum of 5 slides). This presentation (10 min) should summarize the work done and will be used to illustrate the algorithmic work and the results (precision, performance). The quality of the slides and of the presentation are seriously taken into account in the evaluation.
- The modified `m_subspace_iter.F90` (make sure you do not include the other unmodified source files) containing the implementation of the algorithm should be provided.

5 Grading

Your final grade will result from the evaluation of the oral exam (50% of the total) and the deliverables of Phase 1 (25% of the total) and Phase 2 (25% of the total).

6 Important dates

- Deliverables for the second phase (codes, technical report and **pdf file for the oral presentation**) should be provided by **May 29th 2015, 8PM** by email to ehouarn.simon@enseeiht.fr and alfredo.buttari@enseeiht.fr.
- Oral examination will start by June 1st 2015.

7 Bibliography

Appendix 1: LAPACK/BLAS routines Usage

The Fortran implementation of the algorithms described above requires the usage of two routines (whose interface is described below) of the LAPACK and BLAS libraries:

DGEMM : this routine is used to perform a matrix-matrix multiplication of the form $C = \alpha \cdot op(A) \cdot op(B) + \beta C$ where $op(A)$ is either A or A^T, in double-precision, real arithmetic.

DSYEV : this routine computes all the eigenvalues and, optionally, all the eigenvectors of a symmetric, double-precision, real matrix using the QR method.

7.1 A quick note on the leading dimension

The leading dimension is introduced to separate the notion of "matrix" from the notion of "array" in a programming language. In the following we will assume that 2D arrays are stored in "column-major" format, i.e., coefficients along the same column of an array are stored in contiguous memory location; for example the array

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

is stored in memory as such

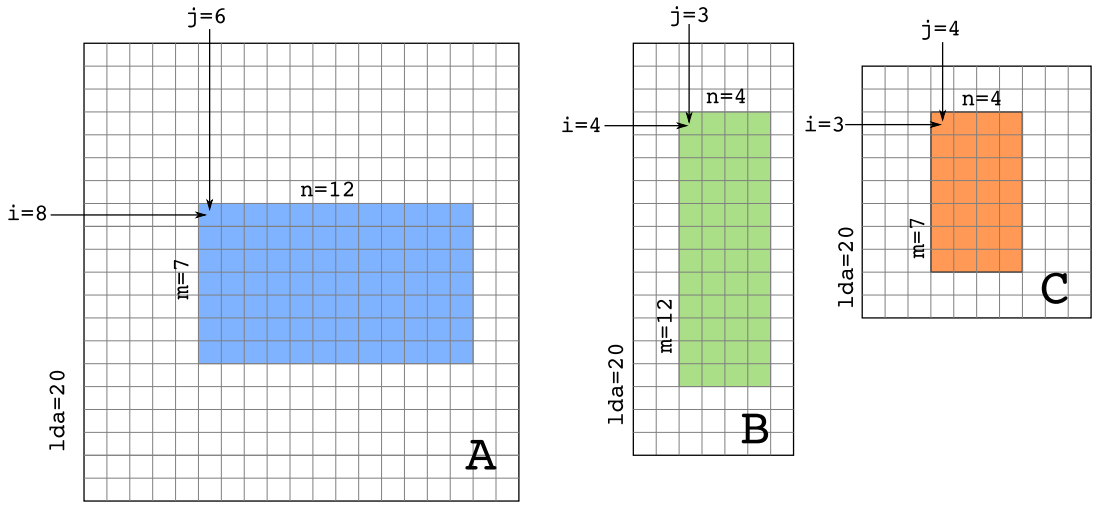
$$a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{32}, a_{13}, a_{23}, a_{33}$$

This is the convention used in the Fortran language and in the LAPACK and BLAS libraries (that were originally written in Fortran).

A matrix can be described as any portion of a 2D array through four arguments (please note below the difference between matrix and array):

- `A(i,j)`: the reference to the upper-left-most coefficient of the **matrix**;
- `m`: the number of rows in the **matrix**;
- `n`: the number of columns in the **matrix**;
- `lda`: the leading dimension of the **array** that corresponds to the number of rows in the **array** that contains the **matrix** (or, equivalently, the distance, in memory, between the coefficients $a_{i,j}$ and $a_{i,j+1}$ for any i and j).

Example:



Assuming the three arrays `A`, `B` and `C` in the figure above have been declared as

```
double precision :: A(20,19), B(18,7), C(11,10)
```

The leftmost matrix (the shaded area within the array `A`) in the figure above can be defined by:

- `A(8,6)` is the reference to the upper-left-most coefficient;
- `m=7` is the number of rows in the matrix;
- `n=12` is the number of columns in the matrix;
- `lda=20` is the leading dimension of the array containing the matrix.

The product of the first (leftmost) two matrices in the figure above can be computed and stored in the last (rightmost) matrix with this call to the BLAS `DGEMM` routine:

```
CALL DGEMM('N', 'N', 7, 4, 12, 1.D0, A(8,6), 20, B(4,3), 18, 0.D0, C(3,4), 11)
```

7.2 DGEMM: interface

```
      SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
*     .. Scalar Arguments ..
      DOUBLE PRECISION ALPHA,BETA
      INTEGER K,LDA,LDB,LDC,M,N
      CHARACTER TRANSA,TRANSB
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
*     ..
*
*  Purpose
*  =======
*
*  DGEMM  performs one of the matrix-matrix operations
*
*     C := alpha*op( A )*op( B ) + beta*C,
*
*  where  op( X ) is one of
*
*     op( X ) = X   or   op( X ) = X',
*
*  alpha and beta are scalars, and A, B and C are matrices, with op( A )
*  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.
*
*  Arguments
*  ==========
*
*  TRANSA - CHARACTER*1.
*           On entry, TRANSA specifies the form of op( A ) to be used in the matrix multiplication
*           as follows:
*
*              TRANSA = 'N' or 'n',  op( A ) = A.
*
*              TRANSA = 'T' or 't',  op( A ) = A'.
*
*              TRANSA = 'C' or 'c',  op( A ) = A'.
*
*           Unchanged on exit.
*
*  TRANSB - CHARACTER*1.
*           On entry, TRANSB specifies the form of op( B ) to be used in the matrix multiplication
*           as follows:
*
*              TRANSB = 'N' or 'n',  op( B ) = B.
*
*              TRANSB = 'T' or 't',  op( B ) = B'.
*
*              TRANSB = 'C' or 'c',  op( B ) = B'.
*
*           Unchanged on exit.
*
*  M      - INTEGER.
```

```
*           On entry,  M  specifies  the number  of rows  of the  matrix op( A )  and of the
*           matrix C. M  must  be at least  zero. Unchanged on exit.
*
*  N      - INTEGER.
*           On entry,  N  specifies the number  of columns of the matrix op( B ) and the number of
*           columns of the matrix C. N must be at least zero. Unchanged on exit.
*
*  K      - INTEGER.
*           On entry,  K  specifies  the number of columns of the matrix op( A ) and the number of
*           rows of the matrix op( B ). K must be at least  zero. Unchanged on exit.
*
*  ALPHA  - DOUBLE PRECISION.
*           On entry, ALPHA specifies the scalar alpha. Unchanged on exit.
*
*  A      - DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is k  when  TRANSA = 'N' or
*           'n',  and is  m  otherwise. Before entry with  TRANSA = 'N' or 'n',  the leading  m by
*           k part of the array  A  must contain the matrix  A,  otherwise the leading  k by m
*           part of the array  A  must contain  the matrix A. Unchanged on exit.
*
*  LDA    - INTEGER.
*           On entry, LDA specifies the first dimension of A as declared in the calling (sub)
*           program. When  TRANSA = 'N' or 'n' then LDA must be at least  max( 1, m ), otherwise
*           LDA must be at least  max( 1, k ). Unchanged on exit.
*
*  B      - DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is n  when  TRANSB = 'N' or
*           'n',  and is  k  otherwise. Before entry with  TRANSB = 'N' or 'n',  the leading  k
*           by n part of the array  B  must contain the matrix  B,  otherwise the leading  n by k
*           part of the array  B  must contain  the matrix B. Unchanged on exit.
*
*  LDB    - INTEGER.
*           On entry, LDB specifies the first dimension of B as declared in the calling (sub)
*           program. When  TRANSB = 'N' or 'n' then LDB must be at least  max( 1, k ), otherwise
*           LDB must be at least  max( 1, n ). Unchanged on exit.
*
*  BETA   - DOUBLE PRECISION.
*           On entry,  BETA  specifies the scalar  beta.  When  BETA  is supplied as zero then C
*           need not be set on input. Unchanged on exit.
*
*  C      - DOUBLE PRECISION array of DIMENSION ( LDC, n ).
*           Before entry, the leading  m by n  part of the array  C must contain the matrix  C,
*           except when  beta  is zero, in which case C need not be set on entry. On exit, the
*           array  C  is overwritten by the  m by n  matrix ( alpha*op( A )*op( B ) + beta*C ).
*
*  LDC    - INTEGER.
*           On entry, LDC specifies the first dimension of C as declared in  the  calling
*           subprogram. LDC  must  be  at  least max( 1, m ). Unchanged on exit.
```

7.3 DSYEV: interface

```
      SUBROUTINE DSYEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO )

*     .. Scalar Arguments ..
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDA, LWORK, N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), W( * ), WORK( * )
*     ..
*
*  Purpose
*  =======
*
*  DSYEV computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.
*
*  Arguments
*  =========
*
*  JOBZ    (input) CHARACTER*1
*          = 'N':  Compute eigenvalues only;
*          = 'V':  Compute eigenvalues and eigenvectors.
```

```
*
*  UPLO    (input) CHARACTER*1
*          = 'U':  Upper triangle of A is stored;
*          = 'L':  Lower triangle of A is stored.
*
*  N       (input) INTEGER
*          The order of the matrix A.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA, N)
*          On entry, the symmetric matrix A.  If UPLO = 'U', the leading N-by-N upper triangular
*          part of A contains the upper triangular part of the matrix A.  If UPLO = 'L', the
*          leading N-by-N lower triangular part of A contains the lower triangular part of the
*          matrix A. On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal
*          eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle
*          (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is
*          destroyed.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,N).
*
*  W       (output) DOUBLE PRECISION array, dimension (N)
*          If INFO = 0, the eigenvalues in ascending order.
*
*  WORK    (workspace/output) DOUBLE PRECISION array, dimension (MAX(1,LWORK))
*          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
*
*  LWORK   (input) INTEGER
*          The length of the array WORK.  LWORK >= max(1,3*N-1). For optimal efficiency,
*          LWORK >= (NB+2)*N, where NB is the blocksize for DSYTRD returned by ILAENV.
*
*          If LWORK = -1, then a workspace query is assumed; the routine only calculates the
*          optimal size of the WORK array, returns this value as the first entry of the WORK
*          array.
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*          > 0:  if INFO = i, the algorithm failed to converge; i
*                off-diagonal elements of an intermediate tridiagonal
*                form did not converge to zero.
*
```