INP ENSEEIHT

Projet d'Informatique et Mathématiques Appliquées
en
Algèbre Linéaire Numérique

USING EOF (EMPIRICAL ORTHOGONAL FUNCTIONS) TO PREDICT THE TEMPERATURE OF THE OCEAN

Contents

1 Introduction and general context

The aim of this project is to illustrate the use of eigenvalues computations in the context of data analysis in a physical application, namely weather forecast. This study consists of two parts:

- The application: the aim is to understand the use of *Empirical Orthogonal Functions* (EOF) in data analysis and climate prediction (this is described in Section 2). These tools will be applied (using Matlab) on sets of climate data. The figure on the first page illustrates such a set: global Sea Surface Temperature (SST) on a $5° \times 5°$ grid in late 2007 (courtesy Group for High Resolution Sea Surface Temperature).
- Computation of eigenpairs: computing EOF amounts to computing the dominant eigenpairs of a matrix derived from the physical data, therefore we are interested in efficient methods for the computation of eigenvalues/eigenvectors. We will focus on some variants of the *power method* (this is described in Section 3); the aim will be to understand the numerical properties and the performance issues of these algorithms, and to implement them in Fortran. In the end, these algorithms will be embedded in the application described above.

2 Application context and introduction to EOF

The method of *Empirical Orthogonal Functions* (EOF) analysis consists in decomposing some data in terms of orthogonal basis functions that express the dominant patterns of the data. These EOF correspond to the eigenvectors of the *covariance matrix* associated to the data.

The tutorial *A Primer for EOF Analysis of Climate Data* by A. Hannachi (Department of Meteorology, University of Reading (UK)) describes in very simple terms what EOF are and the role they play in climate analysis/prediction. **Read the first 11 pages of that document** and pay attention to the following points:

- What the *data matrix* X looks like,
- What the *anomaly field* (or *anomaly matrix*) is,
- What the *covariance matrix* is,
- How EOF are constructed as solutions of a maximization problem,
- What a *Principal Component* (PC) is,
- What the *explained variance* is (and its relation to the trace of the anomaly field) and how to extract dominant EOF,
- How to compute EOF (using an eigenvalue decomposition or a singular value decomposition),
- How EOF can be used for data analysis (what do they express?) and/or prediction.

Note that a crucial point is to understand which objects (EOF, PC, rows/columns of the data matrix) correspond to spatial data or time data.

You will apply EOF analysis to some data corresponding to Sea Surface Temperature. You will be provided some data and some pieces of Matlab code that will help you to display and analyze the data. **You will have to write a Matlab code that performs an EOF analysis and illustrate its use to climate prediction; see Section 4.**

3 Numerical issues and extension of the power method

The basic *power method*, which was introduced in the ALN lectures, is recalled in Algorithm 1; it can be used to determine the eigenpair associated to the largest (in module) eigenvalue.

For the purpose of our discussion we focus on real matrices even if most results can be extended to complex matrices.

Algorithm 1 Vector power method

```
Input: Matrix $A \in \mathbb{R}^{n \times n}$
Output: $(\lambda_1, v_1)$ eigenpair associated to the largest (in module) eigenvalue.
$x_0 \in \mathbb{R}^n$ given and $p = 0$
$\beta_p = {x_p}^T \cdot A \cdot x_p$
repeat
  $y_{p+1} = A \cdot x_p$
  $x_{p+1} = y_{p+1} / \|y_{p+1}\|$
  $\beta_{p+1} = {x_{p+1}}^T \cdot A \cdot x_{p+1}$
  $p = p + 1$
until $|\beta_{p+1} - \beta_p| / |\beta_p| < \varepsilon$
$\lambda_1 = \beta_{p+1}$ and $v_1 = x_{p+1}$
```

Our objective in this section is to introduce (often recalling methods introduced either in the "Algèbre Lineaire" or the "Algèbre Lineaire Numérique" modules) all the ingredients that will be necessary to extend the basic power method to compute blocks of eigenpairs and more precisely to **compute an invariant subspace associated to the largest eigenvalues.**

3.1 The Schur decomposition

Let first recall the Schur decomposition (introduced in the "Algèbre Lineaire" module) that will be applied in our context to symmetric matrices but that we first present in a more general context.

Theorem 1 (General Schur decomposition). *Let $A \in \mathbb{R}^{n \times n}$, there exists an orthogonal matrix U such that*

$$U^T \cdot A \cdot U = T,$$

where T is upper triangular. By appropriate choice of U, the eigenvalues of A, which are the diagonal elements of T, may be made to appear in any order.

For the proof see [ ] (Chapter 1, page 13).
The decomposition $U^T \cdot A \cdot U = T$, is called a *Schur decomposition* of A. Columns of U are called *Schur vectors*. This decomposition is not unique since it depends on the order of the eigenvalues in T and because of potential multiple eigenvalues.
Furthermore, if A is symmetric. $T^T = (U^T \cdot A \cdot U)^T = U^T \cdot A^T \cdot U^{TT} = U^T \cdot A \cdot U = T$. T is thus both upper and lower triangular and hence is diagonal.

$$\Lambda = U^T \cdot A \cdot U \text{ with } \Lambda = diag(\lambda_1, \cdots, \lambda_n)$$

Therefore $A \cdot U = U \cdot \Lambda$ so that, if u_i denotes the i-th column of U then (λ_i, u_i) is an eigenpair of A. We summarize all this in the following theorem.

Theorem 2 (Spectral decomposition of a symmetric matrix). *If A symmetric, it thus has an orthogonal basis of eigenvectors and there exists a decomposition called spectral decomposition:*

$$U^T \cdot A \cdot U = \Lambda \text{ , where } U \text{ is orthogonal, and } \Lambda = diag(\lambda_1, \cdots, \lambda_n)$$

3.2 Rayleigh quotient and Rayleigh-Ritz projection method

We describe in this section the most commonly used method for extracting an approximate eigenspace from a larger subspace: *the Rayleigh-Ritz projection method*, which makes deep use of the notion of *Rayleigh quotient*.
To keep the introduction of Rayleigh-Ritz method simple we first show, as in [:] (Chapter 4.1) how to find exact eigenpairs; we extend this to computing exact eigenspaces and will then suggest (without giving the proof) a procedure to compute approximate eigenpairs.

Theorem 3 (Rayleigh quotient). *Let $\mathcal{U}$ be a subspace and let the columns of the matrix U be an orthogonal basis for $\mathcal{U}$ (U^T is the left inverse of U, $U^T\ U = I$). We define the Rayleigh quotient to be*

$$H = U^T \cdot A \cdot U\ .$$

If $\mathcal{X} \subset \mathcal{U}$ is an eigenspace of A then there is an eigenpair (λ, w) of H such that $(\lambda, U \cdot w)$ is an eigenpair of A with $\mathcal{R}(U \cdot w) = \mathcal{X}$.

Proof. Note that no assumption is made on the size of the subspace $\mathcal{U}$. Let (λ, X) be an eigenpair of A corresponding to $\mathcal{X}$ and let $X = U \cdot W$. By definition, (λ, X) being an eigenpair of A, $A \cdot X = \lambda X$, and $A \cdot U \cdot w = \lambda U \cdot w$. Thus,

$$H \cdot w = U^T \cdot A \cdot U \cdot w = U^T \cdot U \cdot \lambda w = \lambda w\ ,$$

so that (λ, w) is an eigenpair of H. □

With Theorem 3, we have shown that the exact eigenspace contained in $\mathcal{U}$ can be obtained by looking at eigenpairs of the Rayleigh quotient H which is a much smaller matrix of order the dimension of the subspace $\mathcal{U}$.
Furthermore (and we assume it by continuity), we can expect that when $\mathcal{U}$ contains an approximate eigenspace $\tilde{\mathcal{X}}$ of A there would be an eigenpair $(\tilde{L}, \tilde{W})$ of H such that $(\tilde{L}, U \cdot \tilde{W})$ is an approximate eigenpair of A with $\mathcal{R}(U \cdot \tilde{W}) = \tilde{\mathcal{X}}$.
This suggests that to approximate an eigenpair of A we can perform the following steps that define the main steps :

1. Find U an orthogonal basis of a subspace $\mathcal{U}$.
2. Form the Rayleigh quotient $H = U^T \cdot A \cdot U$.
3. Compute (M, W) an eigenpair of H.
4. Then $(M, U \cdot W)$ is an approximate eigenpair of A.

One can also extend the previous procedure to compute an approximation of an eigenspace of dimension m that is illustrated here in the case of a symmetric matrix A (this is often referred to as the *Rayleigh-Ritz projection method*: given U an orthogonal basis of a subspace $\mathcal{U}$,

1. Form the Rayleigh quotient $H = U^T \cdot A \cdot U$.
2. H is symmetric : compute the spectral decomposition of H : $X^T \cdot H \cdot X = \Lambda$
3. X is thus an approximate eigenspace of H corresponding to eigenvalues in $diag(\Lambda)$ so that $U \cdot X$ is an approximate eigenspace of A corresponding to eigenvalues in $diag(\Lambda)$.

Note that in the spectral decomposition of H, the diagonal matrix Λ can be organized in any order (for example in descending order of magnitude) which could then give the idea of studying per column the quality of each approximate eigenpair.

3.3 Extending the power method to computing a dominant eigenspace

3.3.1 Colinearity

It should be first noticed (and can be easily experimented in Matlab) that if one extends Algorithm 1 to iterate simultaneously on m initial vectors (matrix X_p), instead of just one (vector x_p) then all vectors will tend to be colinear to the eigenvector associated to the largest (in module) eigenvalue. This property should be observed and explained and a solution should be provided in the modified algorithm to address this issue.

3.3.2 Stopping criterion

Another difficulty to adapt Algorithm 1 in order to compute blocks of eigenpairs at once is the stopping criterion, because a set of eigenpairs and not only one vector must be tested for convergence.

The current stopping criterion in Algorithm 1 relies on the stability of the computed eigenvalue (it tests the fact that the computed eigenvalue no longer changes "much"). This does not take into account the invariance of the eigenvector which is numerically more meaningful.

One should first notice that, in Algorithm 1, at convergence $x_p = v_1$ holds and therefore $A \cdot x_p = \lambda_1 \cdot x_p$. By analogy with the backward error introduced during the lectures for the solution of linear systems ($A \cdot x = b$ leading to backward error $\frac{\|A \cdot x - b\|}{\|A\|\|x\| + \|b\|}$) we suggest to replace the stopping criterion in Algorithm 1 by a backward error on the invariance of the eigenvectors that can be estimated as $\|A \cdot x_p - \beta_{p+1} \cdot x_p\| / \|A\|$ where the Frobenious norm could used to compute the norm of a matrix.

Furthermore, when a complete set of eigenpairs is expected to converge then the previous criterion should be generalized to a matrix X_p instead of a vector x_p.

3.3.3 Efficiency and controlled accuracy

In our application one might be interested in computing only an estimation of the dominant eigenspace of a symmetric matrix.

The Rayleigh quotient H computed on an estimate of the eigenspace U ($H = U^T \cdot A \cdot U$) is a small square symmetric matrix which holds the eigenspace information and can thus be used for computing the invariance of the eigenspace and thus a global stopping criterion.

Furthermore since H is small computing the spectral decomposition of H is cheap, and the Rayleigh-Ritz projection method can be applied to improve the stopping criterion, to accelerate the method and to obtain the dominant eigenpairs of A. This should be carefully described and explained.

4 Deliverables phase 1: prototype and algorithmic study

A technical report that includes the following items should be provided. An algorithm describing the Matlab prototype should provided, the Matlab code should be well documented and included in the report. The final Matlab file will be provided in the deliverables of the second phase of the project.

1. Analysis of climate data and Empirical Orthogonal Functions (EOF)
   1. **Summarize document** *A Primer for EOF Analysis of Climate Data* by A. Hannachi (Department of Meteorology, University of Reading (UK)) [ ].
   2. Given the EOF of a set of data describing the variations of temperature of the oceans, propose a scenario to illustrate how EOF can be used for prediction.
   3. **Describe the algorithm (referred to as EOF based Prediction)** that, for a given set of data, computes the EOF (indicate which mathematical ingredients are needed); explain how to use them in the context of the previous scenario and how to validate the accuracy of the prediction.
   4. A prototype of the EOF based Prediction algorithm will be developed in Matlab. Efficiency is not our main concern in this context and standard Matlab tools can be used (for example to compute a Schur decomposition). You are provided a few Matlab files: `EOF.m` is the main code and is able to read either some real SST data (for a start, we provide some Kaplan data [ ]; see `http://www.esrl.noaa.gov/psd/data/gridded/data.kaplan_sst.html` for a short description), either some synthetic data generated with `gendata.m` (look at the file or run '`help gendata`' to see what parameters do, and use different values to test your scenario).
2. **Computing an approximation of the eigenspace of a symmetric matrix A of order n**
   1. Describe the difficulties of extending the power method introduced in the lectures to computing a block of m eigenpairs ($m \ll n$) of A associated to the largest (in module) eigenvalues of A.
   2. Propose a first basic algorithm that generalizes the computation of an eigenpair to the computation of a set of eigenpairs at once.
   3. Propose an improved algorithm that in the context of our application can be more efficient at computing an estimation of the dominant eigenvectors of a symmetric matrix A.

5 Deliverables phase 2 and developments

At the end of the first phase, we will provide an additional document to define a common scenario for use of EOF (data analysis and prediction) that will have to be implemented. Your Matlab prototype should be adapted accordingly. Furthermore, a high level algorithmic description of the algorithms to compute an approximation of the dominant eigenvectors will also be included in this document to guide the FORTRAN developments.

Deliverables for the second part of the project will be described in more details later but they should include:

1. A short report to describe the work done:
   - related to the application (maximum of 2 pages)
   - related to the numerical tools (maximum of 2 pages)
   - a summary of the experiments (maximum of 2 pages) (results should be analysed and commented).
2. All files (well commented Matlab and Fortran files).
3. A file (**pdf format**) that summarizes (4 page max) the work done and will be used to illustrate the algorithmic work and the results (precision, performance).

6 Important dates

- The first phase of this work (algorithmic and prototyping; see 4) should be dropped in the mail box of the IMA department by **Friday April 6th 2012**.
- A more detailed description of the work to be done for the second phase will then be given before April 7th (preliminary description in 5
- During the **week of April 23-27**, each group will have an appointment with the teachers (between 1pm and 2pm) in order to receive comments on the first phase of this work.
- Deliverables for the second phase (codes, technical report and **pdf file for the oral presentation**) should be provided by **May 18th 2012** by email to François Henry Rouet (frouet@enseeiht.fr)
- Oral examination will start by May 21st.

7 Bibliography

[1] A. Hannachi. *A primer for EOF analysis of climate data.* `www.met.rdg.ac.uk/~han/Monitor/eofprimer.pdf`. Department of of Meteorology, University of Reading (UK), 2004.

[2] A. Kaplan, M. A. Cane, Y. Kushnir, A. C. Clement, M. B. Blumenthal, and B. Rajagopalan. Analyses of global sea surface temperature 1856-1991. *Journal of Geophysical Research*, 103(18):567–18, 1998.

[3] G. W. Stewart. *Matrix Algorithms: Volume 2, Eigensystems.* Society for Industrial and Applied Mathematics (SIAM), 2001.

INP ENSEEIHT

Projet d'Informatique et Mathématiques Appliquées
en
Algèbre Linéaire Numérique

USING EOF (EMPIRICAL ORTHOGONAL FUNCTIONS) TO PREDICT THE TEMPERATURE OF THE OCEAN (PHASE 2)

Contents

1 Introduction

Complementary data are provided in this short note to define more precisely the scenario for the prediction (in Section 2) and the algorithms used to compute the eigenspace associated to the dominant eigenvalues (Section 3).

2 EOF analysis of Climate data

The method of *Empirical Orthogonal Functions* (EOF) analysis consists in decomposing some data in terms of orthogonal basis functions that express the dominant patterns of the data. These EOF correspond to the eigenvectors of the *covariance matrix* associated to the data. The aim is to apply EOF analysis on data corresponding to Sea Surface Temperature. You were asked to provide **a Matlab code that performs an EOF analysis and illustrate its use to climate prediction**.

For this next phase we impose the following scenarii:

Data analysis

0. We assume that the data is a matrix $F \in \mathbb{R}^{n_t \times n_s}$ where n_t is the temporal dimension (number of time steps) and n_s is the physical dimension (size of the domain). We assume that we have a parameter called *PercentTrace* that corresponds to the amount of *explained variance* that we want to reach. *PercentTrace* will influence the number of EOF (eigenpairs) that have to be kept.
1. Plot the data (this is an animated plot where each frame corresponds to a time step, i.e. a row of matrix F); use the pieces of Matlab code that were provided at the beginning of the project.
2. Compute the anomaly matrix Z from F (see command `detrend` in Matlab).
3. Compute the covariance matrix $S = Z^T \cdot Z$.
4. Compute the dominant eigenpairs of S: either use the built-in Matlab `eig` function and filter the dominant eigenpairs a posteriori (using *PercentTrace* and `trace`), either call the dominant eigenspace method that you developed (cf Algorithm 3). We call $V \in \mathbb{R}^{n_s \times n_d}$ the EOF (eigenvectors) that have been kept (n_d is the number of EOF that have been kept). We call D the set of eigenvalues ordered by decending order of magnitude; $\sum_{i=1}^{n_d} D_i \geqslant \frac{PercentTrace}{100} tr(A)$ must hold.
5. Plot the n_d EOF (use the `subplot` functionality) and their corresponding principal components ($PC_i = Z \cdot V_i$). If you use some SST (Sea Surface Temperature) data, check that each EOF corresponds to some plausible physical data (e.g. geographical patterns appear as in the initial data).
6. Check the quality of the basis (EOF that have been kept): $\frac{\|Z \cdot V \cdot V' - Z\|}{\|Z\|}$ should be small.
7. Check that another basis of n_d vectors of size n_s (e.g. a random basis) would give a worse result: generate a basis W and use the same critetion as above. NB: using `rand` is not enough, you must have n_d independent vectors.

Prediction

0. We assume that matrix F contain n_y years of data; e.g. if F corresponds to monthly measures, $n_t = 12n_y$. We assume that the data on the first $n_y - 1$ years is completely known, and we assume that the data on the last (n_y-th year) is only partially known, i.e. known at only some parts of the domain. For example, in the case of SST measures, the domain is a 2D grid $\mathcal{D}$ of size $n_x \times n_y = n_f$. $\mathcal{D}$ is partitioned into $\mathcal{D} = \mathcal{D}_i \cup \mathcal{D}_\omega$; $\mathcal{D}_i$ is the part of the domain where the data is known, and $\mathcal{D}_\omega$ is the part where the data is unknown and has to be predicted.
1. We consider the first $n_y - 1$ years (i.e. first rows of F): compute the EOF on the data corresponding to these first $n_y - 1$ years applying the same techniques as before. We call V the EOF basis; we define V_i as the restriction of V to $\mathcal{D}_i$ (V_i will be used in the next step).
2. We now consider the last year (e.g. the twelve last rows of F if F is based on monthly measures); we call F_y the corresponding data.
    1. Fit the known data (corresponding to known entries $\mathcal{D}_i$) on the EOF V_i: the aim is to find the components of each row of F_y, restricted to $\mathcal{D}_i$, in the V_i basis; each row j in F_y, restricted to $\mathcal{D}_i$ can be written as $F_y(j, \mathcal{D}_i) = \sum_{k=1}^{n_d} \alpha_k V(\mathcal{D}_i, k)^T$. Therefore, we want to solve the overdetermined system $V(\mathcal{D}_i, :)\alpha = F_y(:, \mathcal{D}_i))^T$ which is a least-squares problem; you can for example use a QR factorization or the normal equations. α is of size $n_d \times 12$ in case of monthly measures.
    2. Once α is determined, we have expressed the known data in the EOF basis. The prediction consists in considering that the same α components are valid on the whole domain, i.e. the predicted data on the whole domain F_p is computed as $F_p = (V \cdot \alpha)^T$.
    3. Since you actually have the missing data in matrix F ($F_y(:, \mathcal{D}_\omega)$), you can assess the quality of your prediction by computing $\frac{\|F_p - F_y\|}{\|F_y\|}$.
    4. Re-run steps 1-3 with a random basis instead of the random basis and check the quality of this "prediction".

3 Extending the power method to compute dominant eigenspace vectors

It has been proposed in the first document to extend the Power Method to iterate simultaneously on m initial vectors V, instead of just one. A direct extension of the power method will in general tend to force all m vectors to be colinear to the eigenvector associated to the largest (in module) eigenvalue. It is thus necessary to enforce orthogonality of the vectors as the iteration proceeds.

3.1 Subspace_V0: basic version

The *first basic version of the method* to compute an invariant subspace associated to the largest eigenvalues is described in Algorithm 1 and makes great use of *Rayleigh quotient and spectral decomposition* as introduced in our first document.

Given set of m linearly independent vectors Y, the Algorithm 1 computes the eigenvectors associated with the m largest (in module) eigenvalues in the $\mathcal{R}(Y)$.

Algorithm 1 Dominant eigenspace method: basic version

```
Input: Symmetric matrix $A \in \mathbb{R}^{n\times n}$, tolerance $\varepsilon$ and $MaxIter$ (max nb of iterations)
Output: $m$ dominant eigenvectors $V_{out}$ and the corresponding eigenvalues $\Lambda_{out}$.

Generate a set of $m$ linearly independent vectors $Y \in \mathbb{R}^{n\times m}$; $niter = 0$
repeat
    $V \longleftarrow$ orthonormalization of the columns of $Y$
    Compute $Y$ such that $Y = A \cdot V$
    Form the Rayleigh quotient $H = V^T \cdot A \cdot V$
    $niter = niter + 1$
until ( Invariance of the subspace $V$ or $niter > MaxIter$ )
Compute the spectral decomposition of the Rayleigh quotient $H$ from which the $m$ dominant eigenvalues
$\Lambda_{out}$ and corresponding eigenspace $V_{out}$ can be deduced.
```

To define the invariance of the subspace let us extend the notion of backward error introduced in our lectures for the solution of linear systems to this eigenspace method. To simplify our discussion, assume that we have converged so that $AV = V\Lambda_{out}$ with $\Lambda_{out} = \text{diag}(\lambda_1, \dots, \lambda_m)$. At convergence we thus have $VH = VV^TAV = VV^TV\Lambda_{out} = V\Lambda_{out}$. Thus, in our context, a possible measure of the backward error could be : $\|AV - VH\|/\|A\|$.

Note that the spectral decomposition has been introduced in the first document.

3.2 Subspace_V1: improved version making use of Raleigh-Ritz projection

Several modifications are needed to make the simple subspace iteration an efficient and practically applicable code. First we may chose to operate on a subspace whose dimension m is larger than the number of the dominant eigenvalues (n_{ev}) needed. As described in [1], the matrix is symmetric positive definite (thus with positive eigenvalues) and the user might ask to compute the smallest eigenspace such that the sum of the associated dominant eigenvalues is larger than a given percentage of the trace of the matrix A. The Rayleigh-Ritz projection procedure can then be used to get the approximate eigenspace and stop when the expected percentage is reached.

The Rayleigh-Ritz projection procedure is combined with subspace iteration method to improve the convergence, it consists in rearranging the orthogonal matrix V so that its columns reflect the fast convergence of the dominant subspaces.

The algorithmic description, for a symmetric matrix A, of this procedure is given below. We assume that the matrix is symmetric positive definite, define the Raleigh-Ritz projection algorithm and then introduce the improved version of our algorithm.

Algorithm 2 Raleigh-Ritz projection

```
Input: Matrix $A \in \mathbb{R}^{n\times n}$ and an orthonormal set of vectors $V$.
Output: The approximate eigenvectors $V$ and the corresponding eigenvalues $\Lambda$.
Compute the Rayleigh quotient $H = V^TAV$.
Compute the spectral decomposition $H = X\Lambda X^T$, where the eigenvalues of $H$ ($diag(\Lambda)$) are arranged
in descending order of magnitude.
Compute $V = V\,S$.
```

Algorithm 3 Dominant eigenspace method with Raleigh-Ritz projection

```
Input: Symmetric matrix $A \in \mathbb{R}^{n\times n}$, tolerance $\varepsilon$, $MaxIter$ (max nb of iterations) and $PercentTrace$
the target percentage of the trace $A$
Output: $m$ dominant eigenvectors $V_{out}$ and the corresponding eigenvalues $\Lambda_{out}$.

Generate an initial set of $m$ orthonormal vectors $V \in \mathbb{R}^{n\times m}$; $niter = 0$; $PercentReached = 0$
repeat
    Compute $Y$ such that $V = A \cdot V$ and orthonormalize $V$
    Rayleigh-Ritz projection applied on orthonormal vectors $V$ and matrix $A$
    Convergence (Section 3.2.1): save eigenpairs that have converged and update $PercentReached$
    $niter = niter + 1$
until ( $PercentReached > PercentTrace$ or $niter > MaxIter$ )
```

3.2.1 Convergence

Convergence is tested immediately after a Rayleigh-Ritz Projection step. We want to test which approximate eigenvector has converged; we will test in order the columns associated to the largest entries in Λ and will stop as soon as one eigenvector has not converged.

Let $\gamma = \|A\|$ and note that if the j_{th} column of V has converged then $\|r_j\| = \|A \cdot V_j - T(j,j) \cdot V_j\| \leq \gamma\,\varepsilon$, where ε is a convergence criterion and γ is a scaling factor. A natural choice of γ is an estimate of some norm of A.

Convergence theory says the eigenvectors corresponding to the largest eigenvalues will converge more swiftly than those corresponding to smaller eigenvalues. For this reason, we should test convergence of the eigenvectors in the order $j = 1, 2, \dots$ and stop with the first one to fail the test.

3.3 Subspace_V2: toward an efficient solver

Two ways of improving the efficiency of the solver are proposed.

1. **Block approach**

   Orthonormalisation is performed at each iteration and is quite costly. One simple way to accelerate the approach is to perform p products at each iteration (replace $V = A \cdot V$ by $V = A^p \cdot V$). Note that this very simple acceleration method is applicable to all versions of the algorithm. One may then want to experiment the influence of large values of p.

2. **Deflation method**

   Because the columns of V converge in order, we can freeze the converged columns of V. This freezing results in significant savings in the matrix-vector ($V = A \cdot V$), the orthogonalization and Rayleigh-Ritz Projection step.

   Specifically, suppose the first l columns of V have converged, and partition $V = [V_1, V_2]$ where V_1 has l columns. Then, we can form the matrix $[V_1, A \cdot V_2]$, which is the same as if we multiply V_1 by A. However, we still need to orthogonalize V_2 with respect to the frozen vectors V_1 by first orthogonalizing V_2 against V_1 and then against itself.

   Finally, the Rayleigh-Ritz Projection step can also be limited to the columns of V that have not converged.

Commencer et Écrire Algo. Section 3.3

4 Deliverables phase 2 and developments

We have provided (see Section 2) an additional document to define a common scenario for the prediction that has to be implemented. Your Matlab prototype should be adapted accordingly.

PLEASE read carefully the README file in Src_Phase2.tgz where all files and testing procedures are described.

The three versions of the eigensolver (described in Section 3) should be written in Fortran. *For Subspace_ V2 version at least one of the two proposed acceleration methods (block approach or deflation method) should be implemented.* A driver is provided to validate and experiment the three versions of the codes. A Mex file is also provided to enable calling the Fortran code directly in Matlab. This Matlab code will thus also enable you do experiment with the three versions of the eigensolver on a real application.

Al codes should be well documented and structured (as suggested in sofware lectures). This part will also be avaluated.

Deliverables for the second part of the project include:

1. A short report to describe the work done:
   - introduce the work done during this second phase
   - summarize the experiments (maximum of 2 pages) (results should be analysed and commented).
   - A global conclusion for the project.
2. All files (well commented Matlab and Fortran files).
   1. **EOF.m**: Matlab file implementing the proposed scenario and calling your Fortran eigensolver.
   2. The module **m_subspace_iter.f90** that includes the three subroutines implementing the three version of the algorithms described in Section 3.
3. 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 4 slides). This presentation (5 min) should summarize the work done and will be used to illustrate the algorithmic work and the results (precision, performance).

5 Important dates

- During the **week of April 23-27**, each group will have an appointment with the teachers (between 1pm and 2pm) in order to receive comments on the first phase of this work.
- Deliverables for the second phase (codes, technical report and **pdf file for the oral presentation**) should be provided by **May 18th 2012** by email to François Henry Rouet (frouet@enseeiht.fr)
- Oral examination will start by May 21st.

6 Bibliography

[1] A. Hannachi. *A primer for EOF analysis of climate data.* `www.met.rdg.ac.uk/~han/Monitor/eofprimer.pdf`. Department of of Meteorology, University of Reading (UK), 2004.

[2] G. W. Stewart. *Matrix Algorithms: Volume 2, Eigensystems.* Society for Industrial and Applied Mathematics (SIAM), 2001.

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.

6.1 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

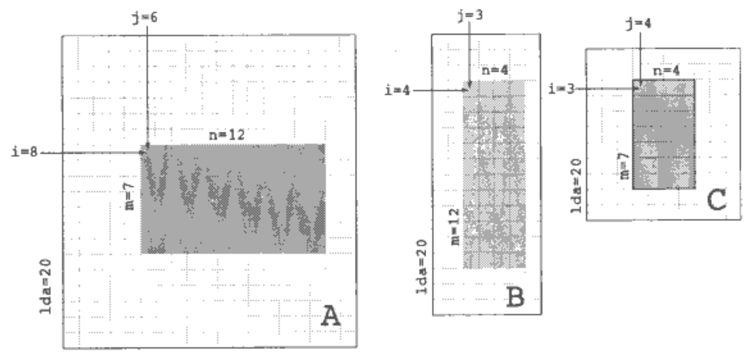
$$a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{32}, a_{13}, a_{23}, a_{33}$$

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)
```

6.2 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.
```

6.3 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.
```