DATA ANALYSIS - SCIENTIFIC COMPUTING

Practical Tutorial 2 - Subspace Iteration Methods

Freshman Year, Computer Science Department

Issam Habibi
Younes Saoudi
Hamza Mouddene

2019-2020

Contents

Limitations of the Power Method

Question 01

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.
$v \in \mathbb{R}^n$ given.
$\beta = v^T \cdot A \cdot v$
repeat
$y = A \cdot v$
$v = y/\|y\|$
$\beta_{old} = \beta$
$\beta = v^T \cdot A \cdot v$
until $|\beta - \beta_{old}|/|\beta_{old}| < \epsilon$
$\lambda_1 = \beta$ and $v_1 = v$

```
 10 First eigenvalues
 ========================
Eigenvalue    1:     1.000   accuracy : 0.990E-15   number of iterations :       9890
Eigenvalue    2:     0.997   accuracy : 0.976E-15   number of iterations :        306
Eigenvalue    3:     0.903   accuracy : 0.977E-15   number of iterations :       2156
Eigenvalue    4:     0.891   accuracy : 0.982E-15   number of iterations :       1844
Eigenvalue    5:     0.878   accuracy : 0.912E-15   number of iterations :        275
Eigenvalue    6:     0.781   accuracy : 0.955E-15   number of iterations :        285
Eigenvalue    7:     0.702   accuracy : 0.931E-15   number of iterations :        493
Eigenvalue    8:     0.653   accuracy : 0.967E-15   number of iterations :       2304
Eigenvalue    9:     0.644   accuracy : 0.906E-15   number of iterations :        196
Eigenvalue   10:     0.553   accuracy : 0.974E-15   number of iterations :        538


 ================================
 Time =    3.0499999523162842
 ================================
```

(a) Deflated Power Method

```
Calling LAPACK DSYEV on A
End of LAPACK DSYEV


  10 First eigenvalues
 ========================
Eigenvalue    1:     1.000
Eigenvalue    2:     0.997
Eigenvalue    3:     0.903
Eigenvalue    4:     0.891
Eigenvalue    5:     0.878
Eigenvalue    6:     0.781
Eigenvalue    7:     0.702
Eigenvalue    8:     0.653
Eigenvalue    9:     0.644
Eigenvalue   10:     0.553


 ================================
 Time =   0.10800000280141830
 ================================
```

(b) LAPACK subroutine `dsyev`

Figure 1.1: Comparison between the running times of the subroutine `deflated_power_method` and the Lapack subroutine `dysev` when computing 10 eigenpairs.

One can see from the figures that, although the extracted eigenvalues are obviously the same, the Deflated Power Method is 30 times slower than the `dysev` LAPACK subroutine when trying to compute 10 eigenpairs.

Question 02

The main drawback of the Deflated Power Method is the fact that this method computes the whole spectral decomposition of the matrix A, which is a square n×n matrix; a process which proves to be exceptionally time consuming.For the LAPACK subroutine *dsyev*, the matrix A is first reduced to tridiagonal form using

orthogonal similarity transformations, then the QR algorithm is applied to the tridiagonal matrix to compute the eigenvalues which is significantly more optimized.

Extending the Power Method to Compute Dominant Eigenspace Vectors

2.1 A Basic Method to Compute a Dominant Eigenspace: `subspace_iter_v0`

Question 03

If one were to extend Algorithm 1 to iterate on the matrix V, then the latter will **not** converge towards a matrix with columns containing the m most dominant eigenvectors as one can see in the following example:
Let there be a matrix $A = \begin{pmatrix} 7 & 9 & 9 \\ 9 & 7 & 9 \\ 9 & 9 & 7 \end{pmatrix}$. The two most dominant eigenpairs of the matrix A are : $\lambda_1 = 25$; $v_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ and $\lambda_2 = -2$; $v_2 = \begin{pmatrix} 0.4359 \\ -1.3831 \\ 0.9472 \end{pmatrix}$
Trying to extract the two most dominant eigenvectors with the **`edited_power_method.m`** script will return the dominant eigenvalue $\lambda_1 = 25$, which normally corresponds to the eigenvector $v_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$, and the vectors $V = \begin{pmatrix} -0.5738 & -0.0636 \\ -0.5738 & -0.0636 \\ -0.5738 & -0.0636 \end{pmatrix}$.
One can thus notice that the matrix V converges towards a matrix with columns just representing the eigenvector v_1 *under a different form* (which will be revealed shortly) instead of a matrix with columns containing the $m = 2$ most dominant eigenvectors.
This "different form" is actually none other than `beta/norm(beta)` with `beta = transpose(v) * A * v`. If one computes its value after the while loop, they will find it to be equal to V.
To avoid this problem, `v = y/norm(y)` should be replaced with `v = mgs(y)` in the **`edited_power_method.m`** script (line 16). The matrix V will contain the eigenvectors associated to the m most dominant eigenvalues as a result. This, however, does not solve the drawback of the Power Method seen in **Question 01**.

(***See Matlab Script*** `edited_power_method.m`)

Question 04

Although a computation of the whole spectral decomposition of the matrix H is performed in Algorithm 2, H is a square m×m matrix (with m being the number of eigenvalues extracted). Thus, this computation is far less significant, especially when compared to the one of the matrix A which is a square n×n matrix (n> > m).The reason behind that is the fact that we only need a few number of eigenpairs in order to provide enough information about the data (as seen in TP1) . Therefore, a data matrix containing thousands of columns could be analysed using only a few dozens of eigenpairs which leads us to choose a relatively small number m .

Question 05

Algorithm 2 Subspace Iteration Version 0

Input: Symmetric Matrix $A \in \mathbb{R}^{n \times n}$, the number of required eigenpairs m, the tolerance ϵ and the maximum number of iterations $MaxIter$.
Output: m dominant eigenvectors V_{out} and the corresponding eigenvalues Λ_{out}

Generate a set of m orthonormal vectors V $\in \mathbb{R}^{n \times m}$; k = 0
repeat
k = k + 1
$Y = A \cdot V$
$H = V^T \cdot A \cdot V$
Compute acc = $\|A \cdot V - V \cdot H\| / \|A\|$
$V \leftarrow$ orthonormalisation of the columns of Y
until ($k > MaxIter$ **or** $acc \leq \epsilon$)
Compute the spectral decomposition $X \cdot \Lambda_{out} \cdot X^T = H$, where the eigenvalues of H (diag(Λ_{out})) are arranged in descending order of magnitude.
Compute the corresponding eigenspace $V_{out} = V \cdot X$

(***See Fortran Script*** `iter_v0.f90`)

2.2 Improved Version Using Raleigh-Ritz Projection: `subspace_iter_v1`

Question 06

Algorithm 3 Rayleigh-Ritz Projection

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

Algorithm 4 Subspace Iteration Version 1 with Rayleigh-Ritz Projection

Input: Symmetric Matrix $A \in \mathbb{R}^{n \times n}$, the number of required eigenpairs m, the tolerance ϵ, the maximum number of iterations $MaxIter$ and the target percentage of the trace of A, $TracePercentage$
Output: n_{ev} dominant eigenvectors V_{out} and the corresponding eigenvalues Λ_{out}

Generate a set of m orthonormal vectors V $\in \mathbb{R}^{n \times m}$; k = 0; $PercentageReached = 0$
repeat
k = k + 1
Compute Y such that $Y = A \cdot V$
$V \leftarrow$ orthonormalisation of the columns of Y
Apply *Rayleigh-Ritz projection* on matrix A and orthonormal vectors V
Convergence Analysis: save the converged eigenpairs and update $PercentageReached$
until ($PercentageReached > TracePercentage$ **or** $n_{ev} > m$ **or** $k > MaxIter$)

All of the steps of Algorithm 4 are already identified and highlighted in French in the file

`subspace_iter_v1.m`

(***See Matlab Script*** `subspace_iter_v1.m`)

Towards an Efficient Solver: `subspace_iter_v2 & subspace_iter_v3`

Two ways of improving the efficiency of the solver are proposed. Our aim is to build an algorithm that combines both the block approach and the deflation method in order to speed-up the convergence of the solver.

3.1 Block approach (`subspace_iter_v2`)

Orthonormality 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 $V = A.V$ (first step of the iteration) by $V = A^p.V$). Note that this very simple acceleration method is applicable to all versions of the algorithm.

Question 07

The definition of matrix multiplication is that if $C = AB$ for an $n \times m$ matrix A and an $m \times p$ matrix B, then C is an $n \times p$ matrix with entries

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \tag{3.1}$$

From this, a simple algorithm can be constructed which loops over the indices i from 1 through n and j from 1 through p, computing the above using a nested loop:

Algorithm 5 Iterative algorithm of matrix multiplication

```
1: Input: matrices A and B
2: Let C be a new matrix of the appropriate size
3: for i from 1 to n do
4:     for j from 1 to p do
5:         sum = 0
6:         for k from 1 to m do
7:             sum ← sum + A_ik * B_kj
8:         end for
9:         C_ij ← sum
10:    end for
11: end for
12: return C
```

This algorithm takes time $\Theta(n \times m \times p)$. A common simplification for the purpose of algorithms analysis is to assume that the inputs are all square matrices of size $n \times n$, in which case the running time is $\Theta(n^3)$, i.e., cubic. So if we assume that A is equal to B we will find that the calculation A^2 has a complexity of $\Theta(n^3)$, then by recurrence we can show that the calculation A^p has a complexity of $\Theta((p-1)n^3)$.
Now we suppose that we have a matrix $V \in M_{n \times m}$, the calculation of $A^p.V$ has a complexity of $\Theta(m \times n^2 + (p-1)n^3)$.
One good way to reduce the cost is implementing a loop that calculates $V = AV$ p times rather than calculating A^p, this will lead us to the same result $V = A^pV$ in a less costy way due to the size of V , the complexity here should be $\Theta(p \times m \times n^2)$.
Another way out is Singular Value Decomposition SVD. Given an $n \times n$ real matrix A of full rank , SVD splits it apart as $A = U\Sigma U^T$ where Σ is a diagonal matrix, in time $theta(n^3)$. By the properties of SVD, $A^m = U\Sigma^m U^T$, so only the powers of the diagonal matrix need to be computed, and this can be done in $\Theta(n \times log(m))$ time.
Performing the final multiplication $U \times \Sigma^m \times U^T$ takes $\Theta(n^{2.3727})$, so we have altogether $\Theta(n^3 + nlog(m))$ operations.

Question 08

`iter_v2.f90`
(***See Fortran file*** `iter_v2.f90`)

3.2 Deflation Method (`subspace_iter_v3`)

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 ($V = A\Delta V$), the orthonormalisation and Rayleigh-Ritz Projection steps.
Specifically, suppose the first nbc^2 columns of V have converged, and partition $V = [V_c, V_{nc}]$ where V_c has nbc columns and V_{nc} has $m - nbc$ columns 3. Then, we can form the matrix $[V_c, A\Delta V_{nc}]$, which is the same as if we multiply V by A. However, we still need to orthogonalise V_{nc} with respect to the frozen vectors V_c by first orthogonalising V_{nc} against V_c and then against itself.
Finally, the Rayleigh-Ritz Projection step can also be limited to the columns V_{nc} of V.

Question 09

The execution of the main program with the option $disp = 2$, displays the precision of a vector when it converges at the end of the computation to check if everything is OK. With the `subspace_iter_method_v1`, this precision is different for some of the vectors because of the Raleigh-Ritz projection which gives as output a vector containing the approximations of the eigenvalues which is used to calculate the norm of the residue, to deter-
mine if a vector converges or not, and thus to know if the eigenvalue makes it possible to reach the percentage.

Question 10

The subspace iter v2 method will be much faster than the subspace iter v1, because using $V = A^p$ instead of $V = A \times V$ allows us to accelerate the method globally, unlike the first one that uses an othonormalization and computation of proper couples all the iterations.

Question 11

`iter_v3.f90`
(***See Fortran file*** `iter_v3.f90`)

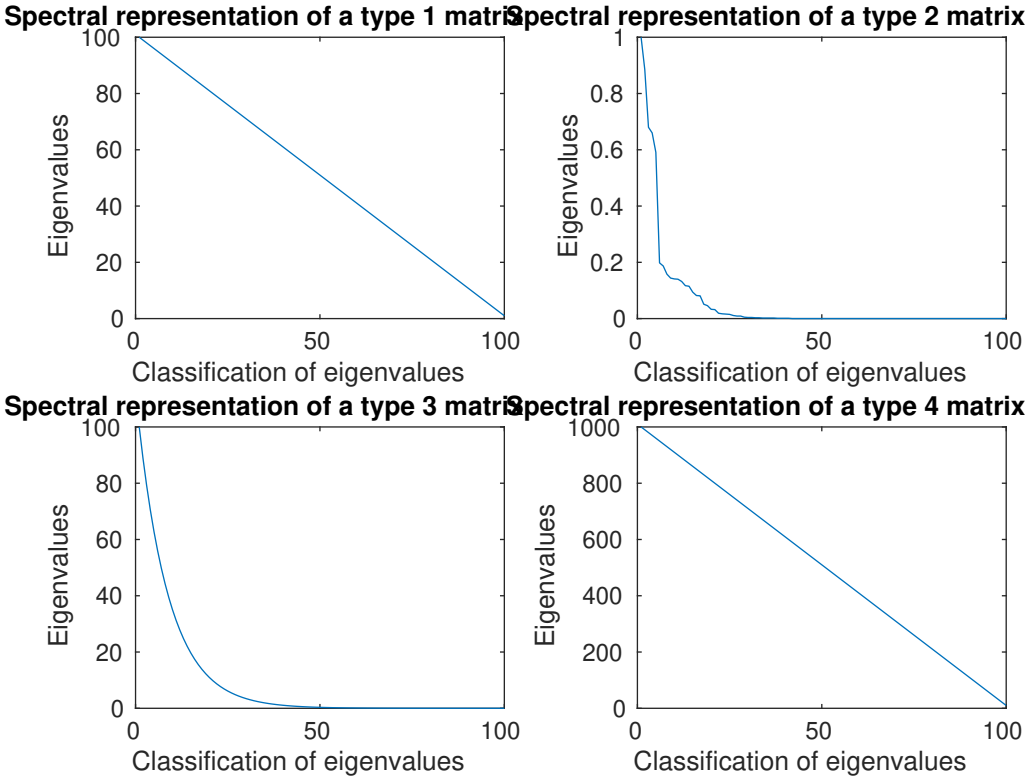
Numerical experiments

Question 12

We can notice that when p increases, time decreases as a logical consequence of the decrease in the number of iterations in our algorithm. As such, the reduction in orthonormalization and eigenvector search allows us to save considerable time.

Question 13

The following images show the eigenvalue distribution of these different types:



Question 14

For the first and the fourth types , the $DSYEV$ is faster with a matrix that has big dimensions than the rest of the methods followed by $subspace_iter_v3$ method . For matrix with small dimensions, all of the methods have almost the same performance with a small advantage again for the $DSYEV$ method and the $power_v11$ method. For the second and the third types , we can witness the same results except that $subspace_iter_v2$ performs better in the second type.

Fortran/Matlab Source Code Listings

5.1 Subspace Iteration Version 0

```
! This file is provided as part of the "projet long" for the "Calcul Scientifique et Analyse de Donn es" course
! at INP-ENSEEIHT
! Date: 02/2019
! Authors: P. Amestoy, P. Berger, A. Buttari, Y. Diouane, S. Gratton, R. Guivarch, F.H. Rouet, E. Simon
!
! This file contains the implementation of
! - one routine for computing the eigenpairs of a symmetric matrix with the basic subspace iteration method
!   (TO BE DEVELOPPED DURING "TP2 projet")
!
!
!    ------------------------------------------------------------------------------------------------------
!
!! subspace_iter_v0
! -----------------
!
! routine that computes a certain number of eigenvalues and eigenvectors of a matrix A
! using the subspace iteration method (v0)
!
! ..Input arguments
!    n : integer, size of the matrix A
!    m : integer, maximum number of searched eigenvalues
!    a : double precision matrix
!
!    maxit : integer, maximum number of iterations of the power method
!    eps : the tolerance for the stopping criterion of the power method
!
! ..Input/Output arguments
!    v : double precision matrix, (input) the starting subspace
!                                 (output) the eigenvectors corresponding
!                                          to the dominant eigenvalues
!
! ..Output arguments
!    w : double precision vector, the eigenvalues
!    acc : double precision vector, accuracy for the invariant subspace
!    it : integer vector, number of iterations of the method
!
!    ierr : integer, indicates how the routine ends
!           0 : OK
!          -1 : inconsistancy of the values of the arguments (ERROR)
!          -2 : allocation problem (ERROR)
!          -3 : maxit reached (ERROR)
!          -4 : problem in dsyev (ERROR)
!
```

```
!-----------------------------------------------------------------------------------------------

subroutine subspace_iter_v0(n, m, a, maxit, eps, v, w, acc, it, ierr)
  implicit none
  !! the subspace dimensions
  integer,          intent(in)                    :: n, m
  !! the target matrix
  double precision, dimension(n, n), intent(in)   :: a
  !! maximum # of iteration
  integer,          intent(in)                    :: maxit
  !! the tolerance for the stopping criterion
  double precision, intent(in)                    :: eps
  !! the m dominant eigenvalues
  double precision, dimension(m), intent(out)     :: w
  !! the starting subspace. The computed eigenvectors will be
  !! returned in this array
  double precision, dimension(n, m), intent(inout) :: v
  !! the returned accuracy
  double precision, intent(out)                   :: acc
  !! the number of iteration to converge
  integer,          intent(out)                   :: it
  !! a flag for signaling errors
  integer,          intent(out)                   :: ierr

  ! external functions
  double precision, external                      :: dlange

  ! constants
  integer, parameter                              :: ione = 1
  double precision, parameter                     :: done = 1.d0, dzero = 0.d0, dmoins =
  -1.d0

  !! local variables
  integer                                         :: i, j, k
  double precision, allocatable, dimension(:,:)   :: aux, y
  !! h, rayleigh-Ritz quotient ; x, eigenvectors of h
  double precision, allocatable, dimension(:,:)   :: h, x
  !! w_aux, eigenvalues computed by spectral decomposition subroutine DSYEV
  double precision, allocatable, dimension(:)     :: w_aux
  double precision, allocatable, dimension(:)     :: work
  integer                                         :: lwork
  double precision                                :: normF_A


  ierr = 0

  if(m.gt.n)then
     ierr = -1
     return
  end if

  lwork = m*m + 5*m + n*n

  allocate(aux(n, m), y(n, m), h(m, m), x(m, m), w_aux(m), work(lwork), stat = ierr)
  if(ierr .ne. 0) then
    ierr = -2
    return
  end if

  normF_A = dlange('f', n, n, a, n, work)

  k = 0
  acc = 100*eps

  !! v is a set of orthonormal vectors
```

```
105     call gram_schmidt(v, n, m, y)
106     v = y
107
108     do while((acc .ge. eps) .and. (k .lt. maxit))
109
110       k = k + 1
111
112       !! Compute  y = a*v
113       call dgemm('n', 'n', n, m, n, done, a, n, v, n, dzero, y, n)
114
115       !! Compute  h = v'*y
116       call dgemm('t', 'n', m, m, n, done, v, n, y, n, dzero, h, m)
117
118       !! Compute the accuracy ||a*v - v*h||/||A|| == ||y - v*h||/||A||
119       !! Compute  aux = y-v*h
120       aux = y
121       call dgemm('n', 'n', n, m, m, dmoins, v, n, h, m, done, aux, n)
122
123       !! Compute acc = ||aux|| / ||A||
124       acc = dlange('f', n, m, aux, n, work) / normF_A
125
126       write(*,'(" IT:",i5," -- Accuracy is: ",es10.2,a)',advance='no') k, acc, char(13)
127       !! V <- orthonormalization of y
128       call gram_schmidt(y, n, m, v)
129
130     end do
131
132     if(acc .lt. eps) then
133       ! compute the spectral decomposition of the Rayleigh quotient h
134       call dsyev('v', 'u', m, h, m, w_aux, work, lwork, ierr)
135       if( ierr .ne.0 )then
136         write(*,'("Error in dsyev")')
137         ierr = -4
138         goto 999
139       end if
140
141       !! Sort in the decreasing order (dsyev returns eigenvalues in ascending order)
142       !! (we suppose that all the eigen values are positive)
143       do i=1, m
144         x(:, i) = h(:, m-i +1)
145         w(i) = w_aux(m-i +1)
146       end do
147
148       !! v = v*x
149       y = v
150       call dgemm('n', 'n', n, m, m, done, y, n, x, m, dzero, v, n)
151
152     else
153       ierr = -3
154     end if
155
156 999 continue
157     deallocate(aux, y, h, x, w_aux, work)
158     write(*,*)
159
160     it = k
161
162     return
163   end subroutine subspace_iter_v0
```

5.2 Edited Power Method (Matlab)

```
function [ w, v0, it, flag ] = edited_power_method(A, m, eps, kmax)

    n = size(A,1); %Number of row of Matrix A
    %Question 3 Modification :
    %OLD: v = ones(n,1); v = v / norm(v); % Random normalized Vector
    %NEW:
    v = randn(n, m); v = mgs(v); %Randomized orthonormal n*m Matrix


    beta = norm(transpose(v)*A*v); %Eventual Eigenvalues
    cv = false; %Did it converge?
    k = 0;  %Number of iterations

    while(~cv)
        y = A*v;
        v = y/norm(y);
        beta_old = beta;
        %Question 3 Modification :
        %OLD : beta = transpose(v) * y
        %NEW:
        beta = norm(transpose(v)*y);
        err1 = (abs(beta - beta_old) / abs(beta_old));
        k = k + 1;
        cv = ( (err1 <= eps) | (k >= kmax));
    end

    it = k;
    w = beta;
    v0 = v;
    if (cv)
        flag = 0; %raise flag = 0 : converged!
    else
        flag = -3; %raise flag = -3 : could not converge!
    end
```

5.3 Subspace Iteration Version 1 (Matlab)

```
% version am lior e de la m thode de l'espace invariant (v1)
% avec utilisation de la projection de Raleigh-Ritz

% Donn es
% A          : matrice dont on cherche des couples propres
% m          : taille maximale de l'espace invariant que l'on va utiliser
% percentage : pourcentage de la trace recherch
% eps        : seuil pour d terminer si un vecteur de l'espace invariant a converg
% maxit      : nombre maximum d'it ration de la m thode

% R sultats
% W : vecteur contenant les valeurs propres (ordre d croissant)
% V : matrice des vecteurs propres correspondant
% n_ev : nombre de valeurs propres calcul es
% it : nombre d'it rations de la m thode
% flag : indicateur sur la terminaison de l'algorithme
    %  flag = 0  : on converge en ayant atteint le pourcentage de la trace recherch
    %  flag = 1  : on converge en ayant atteint la taille maximale de l'espace
    %  flag = -3 : on n'a pas converg  en maxit it rations

function [ W, V, n_ev, it, flag ] = subspace_iter_v1( A, m, percentage, eps, maxit )

    % calcul de la norme de A (pour le crit re de convergence d'un vecteur (gamma))
    normA = norm(A, 'fro');

    % trace de A
    traceA = trace(A);

    % valeur correspondnat au pourcentage de la trace    atteindre
    vtrace = percentage*traceA;

    n = size(A,1);
    W = zeros(n,1);

    % num ro de l'it ration courante
    k = 0;
    % somme courante des valeurs propres
    eigsum = 0.0;
    % nombre de vecteurs ayant converg s
    nb_c = 0;

    % indicateur de la convergence
    conv = 0;

    % on g n re un ensemble initial de m vecteurs orthogonaux
    Vr = randn(n, m);
    Vr = mgs(Vr);

    % rappel : conv = (eigsum >= trace) | (nb_c == m)
    while (~conv & k < maxit),

        k = k+1;
        %% Y <- A*V
        Y = A*Vr;
        %% orthogonalisation
        Vr = mgs(Y);

        %% Projection de Rayleigh-Ritz
        [Wr, Vr] = rayleigh_ritz_projection(A, Vr);

        %% Quels vecteurs ont converg     cette it ration
        analyse_cvg_finie = 0;
```

```
63          % nombre de vecteurs ayant converg     cette it ration
64          nbc_k = 0;
65          % nb_c est le dernier vecteur    avoir converg     l'it ration pr c dente
66          i = nb_c + 1;
67
68          while(~analyse_cvg_finie),
69              % tous les vecteurs de notre sous-espace ont converg
70              % on a fini (sans avoir obtenu le pourcentage)
71              if(i > m)
72                  analyse_cvg_finie = 1;
73              else
74                  % est-ce que le vecteur i a converg
75
76                  % calcul de la norme du r sidu
77                  aux = A*Vr(:,i) - Wr(i)*Vr(:,i);
78                  res = sqrt(aux'*aux);
79
80                  if(res >= eps*normA)
81                      % le vecteur i n'a pas converg ,
82                      % on sait que les vecteurs suivants n'auront pas converg  non plus
83                      % => it ration finie
84                      analyse_cvg_finie = 1;
85                  else
86                      % le_vecteur i a converg
87                      % un de plus
88                      nbc_k = nbc_k + 1;
89                      % on le stocke ainsi que sa valeur propre
90                      W(i) = Wr(i);
91
92                      % on met    jour la somme des valeurs propres
93                      eigsum = eigsum + W(i);
94
95                      % si cette valeur propre permet d'atteindre le pourcentage
96                      % on a fini
97                      if(eigsum >= vtrace)
98                          analyse_cvg_finie = 1;
99                      else
100                         % on passe au vecteur suivant
101                         i = i + 1;
102                     end
103                 end
104             end
105         end
106
107         % on met    jour le nombre de vecteurs ayant converg s
108         nb_c = nb_c + nbc_k;
109
110         % on a converg  dans l'un de ces deux cas
111         conv = (nb_c == m) | (eigsum >= vtrace);
112
113     end
114
115     if(conv)
116         % mise    jour des r sultats
117         n_ev = nb_c;
118         V = Vr(:, 1:n_ev);
119         W = W(1:n_ev);
120         it = k;
121     else
122         % on n'a pas  converg
123         W = zeros(1,1);
124         V = zeros(1,1);
125         n_ev = 0;
126         it = k;
127     end
```

```
128
129     % on indique comment on a fini
130     if(eigsum >= vtrace)
131         flag = 0;
132     else if (n_ev == m)
133             flag = 1;
134         else
135             flag = -3;
136         end
137     end
138 end
```

5.4 Subspace Iteration Version 2

```
! This file is provided as part of the "projet long" for the "Calcul Scientifique et Analyse de Donn es" course
! at INP-ENSEEIHT
! Date: 02/2019
! Authors: P. Amestoy, P. Berger, A. Buttari, Y. Diouane, S. Gratton, R. Guivarch, F.H. Rouet, E. Simon
!
! This file contains the implementation of
! - one routine for computing the eigenpairs of a symmetric matrix with the subspace iteration method
!   with Rayleigh-Ritz projection + block approach (v2) + deflation (v3) (TO BE DEVELOPPED)
!
!
!-------------------------------------------------------------------------------------------------------------
!
!! subspace_iter_v2
! -----------------
!
! routine that computes a certain amount of eigenvalues and eigenvectors of a matrix A
! using the subspace iteration method with Rayleigh-Ritz projection + block approach (v2) + deflation (v3)
!
! ..Input arguments
!    n : integer, size of the matrix A
!    m : integer, size of the searched subspace
!    a : double precision matrix
!    p : number of products Y = A^p . V
!
!    percentage : double precision, fraction of the trace we would like to obtain
!
!    maxit : integer, maximum number of iterations of the power method
!    eps : the tolerance for the stopping criterion of the power method
!
! ..Input/Output arguments
!    v : double precision matrix, (input) the starting subspace
!                                 (output) the eigenvectors corresponding
!                                          to the dominant eigenvalues
!
! ..Output arguments
!    w : double precision vector, the eigenvalues
!    n_ev : integer, the number of computed eigenvalues
!    acc_ev : double precision vector, accuracy for each eigenvalues
!    it_ev : integer vector, number of iterations for each eigenvalues
!
!    ierr : integer, indicates how the routine ends
!           0 : OK
!           1 : the maximum number of eigenvalues is reached before obtaining the percentage (WARNING)
!          -1 : inconsistancy of the values of the arguments (ERROR)
!          -2 : allocation problem (ERROR)
!          -3 : maxit reached (ERROR)
!          -4 : problem in dsyev (ERROR)
!
!-------------------------------------------------------------------------------------------------------------

  subroutine subspace_iter_v2(n, m, a, p, percentage, maxit, eps, v, w, n_ev, acc_ev, it_ev, ierr)
    implicit none
    !! the subspace dimensions
    integer,          intent(in)                    :: n, m
    !! the target matrix
```

```
53      double precision, dimension(n, n), intent(in)     :: a
54      !! number of products Y = A^p . V
55      integer,          intent(in)                      :: p
56      !! the percentage of the trace we want to obtain
57      double precision, intent(in)                      :: percentage
58      !! maximum # of iteration
59      integer,          intent(in)                      :: maxit
60      !! the tolerance for the stopping criterion
61      double precision, intent(in)                      :: eps
62      !! the starting subspace. The computed eigenvectors will be
63      !! returned in this array
64      double precision, dimension(n, m), intent(inout) :: v
65      !! the m dominant eigenvalues
66      double precision, dimension(m), intent(out)       :: w
67      !! number of computed eigenvalues
68      integer,          intent(out)                     :: n_ev
69      !! accuracy for each eigenvalues
70      double precision, dimension(m),   intent(out)     :: acc_ev
71      !!  number of iterations for each eigenvalues
72      integer,          dimension(m),   intent(out)     :: it_ev
73      !! a flag for signaling errors
74      integer,          intent(out)                     :: ierr
75
76      ! external functions
77      double precision, external                        :: dlange, ddot
78
79      ! constants
80      integer, parameter                                :: ione = 1
81      double precision, parameter                       :: done = 1.d0, dzero = 0.d0, dmoins =
        -1.d0
82
83      !! local variables
84      integer                                           :: i, j
85      double precision, allocatable, dimension(:,:)     :: y
86      double precision, allocatable, dimension(:,:)     :: h, x
87      double precision, allocatable, dimension(:)       :: aux_acc, w_aux, t
88      double precision                                  :: trace, p_trace, eig_sum, normF_A
89      integer                                           :: conv
90      integer                                           :: k
91      double precision                                  :: acc
92      logical                                           :: ok
93      double precision                                  :: beta
94      !! the length of the workspace for dsyev
95      integer                                           :: lwork
96      !! the workspace
97      double precision, allocatable, dimension(:)       :: work
98      !! the matrix where we will store A^p
99      double precision, dimension(n,n)                  :: c
100
101     ierr = 0
102
103     !inconsistancy of the values of the arguments (ERROR)
104     if(m.gt.n)then
105        ierr = -1
106        return
107     end if
108
109     !allocation problem (ERROR)
110     lwork = m*m + 5*m + n*n
111     allocate(y(n, m), h(m, m), x(m, m), w_aux(m), aux_acc(n), t(m), work(lwork), stat=ierr)
112     if(ierr .ne. 0) then
113       ierr = -2
114       return
115     end if
116
```

```
117     !! Calculate A^p
118     c = a
119     do i = 1, p
120         call dgemm('n', 'n', n, n, n, done, a, n, c, n, dzero, c, n)
121     end do
122
123     !Calculate the trace of matrix A
124     trace = 0.d0
125     do i = 1, n
126       trace = trace + a(i,i)
127     end do
128     p_trace = percentage * trace
129     eig_sum = 0.d0
130
131     normF_A = dlange('f', n, n, a, n, work)
132
133     it_ev = 0
134     acc_ev = 0.D0
135     n_ev = 0
136     k = 0
137
138     !! Initial set of orthonormal vectors
139     call gram_schmidt(v, n, m, y)
140     v = y
141
142     do while((eig_sum .lt. p_trace) .and. (n_ev .lt. m) .and. (k .lt. maxit))
143
144       k = k + 1
145
146       !! A. Compute  y = a*v
147       call dgemm('n', 'n', n, m, n, done, c, n, v, n, dzero, y, n)
148
149       !! B. Orthonormalisation
150       call gram_schmidt(y, n, m, v)
151
152       !! C. Rayleigh-Ritz projection
153       !!    1. H = V^T A V
154       !!       Y = A V
155       call dgemm('n','n', n, m, n, done, c, n, v, n, dzero, y, n)
156       !!       H = V'*Y
157       call dgemm('t', 'n', m, m, n, done, v, n, y, n, dzero, h, m)
158       !!    2. Spectral decomposition
159       call dsyev('v', 'u', m, h, m, w_aux, work, lwork, ierr)
160       !problem in dsyev (ERROR)
161       if( ierr .ne.0 )then
162         write(*,'("Error in dsyev")')
163         ierr = -4
164         goto 999
165       end if
166
167       !! Sort in the decreasing order
168       !! (we suppose that all the eigen values are positve)
169       do i = 1, m
170         t(i) = w_aux(m-i+1)
171         x(:, i)  = h(:, m-i+1)
172       end do
173
174       !!    3. V = VX
175       y = v
176       call dgemm('n', 'n', n, m, m, done, y, n, x, m, dzero, v, n)
177
178       !! D. Convergence analysis step
179       conv = 0
180       i = n_ev + 1
181       !! the larger eigenvalue will converge more swiftly than
```

```
182       !! those corresponding to the smaller eigenvalue.
183       !! for this reason, we test the convergence in the order
184       !! i=1,2,.. and stop with the first one to fail the test
185       ok = .false.
186       do while(.not. ok)
187         if( i .gt. m) then
188           ok = .true.
189         else
190           !!compute acc=norm(a*v(:,i) - v(:,i)*t(i),2)/lambda;
191           !!--compute aux_acc=a*v(:,i) - v(:,i)*t(i)
192           !!--compute acc=||aux_acc||/||a||
193           aux_acc = v(:,i)
194           beta = -t(i)
195           call dgemv('n', n, n, done, a, n, v(1,i), ione, beta, aux_acc, ione)
196           acc = sqrt(ddot(n, aux_acc, ione, aux_acc, ione))/normF_A
197           ! write(*,*) i, acc
198
199           if(acc.gt.eps) then
200             ok = .true.
201           else
202             ! write(*,*) 'vector', i, 'converges', acc
203             conv = conv + 1
204             w(i) = t(i)
205             acc_ev(i) = acc
206             it_ev(i) = k
207             eig_sum = eig_sum + w(i)
208             i = i + 1
209             if( eig_sum .ge. p_trace) ok = .true.
210           end if
211         end if
212       end do
213
214       n_ev = n_ev + conv
215       !write(*,*) n_ev
216
217     end do
218
219     !the maximum number of eigenvalues is reached before obtaining the percentage (WARNING)
220     if(n_ev .eq. m) then
221       ierr = 1
222     end if
223
224     !maxit reached (ERROR)
225     if(k .eq. maxit) then
226       ierr = -3
227     end if
228
229 999 continue
230     deallocate(y, h, x, w_aux, aux_acc, t)
231     write(*,*)
232
233     return
234   end subroutine subspace_iter_v2
```