LIEN DU RAPPORT FINAL: https://1drv.ms/w/s!ArCPQVP6DRoYow1g81OkoXTXlriB?e=k7Ed4g

SEANCE 1 : SUBSPACE ITERATION METHODS

Partie I (Elisa)

Question 1:

Taille de la matrice	Type de la matrice	Temps eig (s)	Temps puissance itérée (s)
200*200	1	5.000e-02	1.360e+00
200*200	2	1.000e-02	2.000e-02
200*200	3	1.000e-02	5.000e-02
200*200	4	1.000e-02	1.350e+00
400*400	1	1.000e-01	1.100e+01
400*400	2	2.000e-02	4.000e-02
400*400	3	3.000e-02	3.200e-01
400*400	4	5.000e-02	1.141e+01
600*600	1	8.000e-02	4.173e+01
600*600	2	7.000e-02	1.300e-01
600*600	3	1.000e-01	1.050e+00
600*600	4	7.000e-02	4.273e+01

Question 2:

Nouvelle version de l'algorithme :

```
function [ V, D, n_ev, itv, flag ] = power_v12( A, m, percentage, eps, maxit )

        n = size(A,1);
        % initialisation des résultats
        W = [];
        V = [];
        itv = [];
        n_ev = 0;
        % trace de A
        tA = trace(A);
        % somme des valeurs propres
        eig_sum = 0.0;
        % indicateur de la convergence (pourcentage atteint)
        convg = 0;
```

```
% numéro du couple propre courant
k = 0;
while (~convg && k < m)

        k = k + 1;
        % méthode de la puissance itérée
        v = randn(n,1);
        z = A*v;
        beta = v'*z;
        % conv = || beta * v - A*v||/|beta| < eps
        % voir section 2.1.2 du sujet
        norme = norm(beta*v - z, 2)/norm(beta,2);
        nb_it = 1;

        while(norme > eps && nb_it < maxit)
                v = z / norm(z,2);
                z = A*v;
                beta = v'*z;
                norme = norm(beta*v - z, 2)/norm(beta,2);
                nb_it = nb_it + 1;
        end

        % la calcul de ce couple propre a échoué => échec global
        if(nb_it == maxit)
                flag = -3;
                D = 0;
                % on sort de la fonction en plein milieu
                % ce n'est pas très bien structuré
                % pardon aux enseignants de PIM
                return;
        end

        % on sauvegarde le couple propre
        W(k) = beta;
        V(:,k) = v;
        itv(k) = nb_it;
        eig_sum = eig_sum + beta;
        % déflation
        A = A - beta * (v*v');
        % est-ce qu'on a atteint le pourcentage
        convg = eig_sum/tA > percentage;

end
% on a atteint le pourcentage
if (convg)
        n_ev = k;
        flag = 0;
        W = W';
        D = diag(W);
else
        % ce n'est pas le cas
        flag = 1;
        D = 0;
end
```

```
end
```

Table des valeurs:

Taille de la matrice	Type de la matrice	Temps eig (s)	Temps puissance itérée (s)
200*200	1	3.000e-02	5.700e-01
200*200	2	1.000e-02	4.000e-02
200*200	3	1.000e-02	5.000e-02
200*200	4	1.000e-02	8.500e-01
400*400	1	4.000e-02	3.440e+00
400*400	2	2.000e-02	4.000e-02
400*400	3	5.000e-02	9.000e-02
400*400	4	5.000e-02	3.540e+00
600*600	1	9.000e-02	1.204e+01
600*600	2	7.000e-02	5.000e-02
600*600	3	1.000e-01	2.700e-01
600*600	4	7.000e-02	1.242e+01

Il faut refaire toutes les mesures sur le même PC pour pouvoir confirmer que le nouvel algorithme est bien 2 fois plus rapide que l'ancien.

Question 3:

Le principal défaut de l'algorithme de la méthode à la puissance itérée est qu'il comporte deux boucles while imbriquées, ce qui augmente considérablement la complexité de l'algorithme.

Partie II (Nicolas)

Question 4:

Appliquer la l'algorithme de la puissance à une famille de m vecteurs plutôt qu'à un vecteur risque de nous faire converger vers une famille liée de m fois le vecteur propre associé à la valeur propre dominante.

Question 5:

A est une matrice symétrique réelle, donc par le théorème spectrale est diagonalisable en base orthonormée et par définition de H elle est également diagonalisable par le même argument.

L'algorithme subspace_iter_v0 va permettre à la matrice H d'approcher puis de contenir les m plus grandes valeurs propres de A en module, or nous considérons m<<n ainsi déterminer le spectre de H est un problème bien plus simple que celui du spectre de A, tout en récupérant les informations des valeurs propres dominantes de la matrice A.

Question 6: done

Question 7: ALGORITHME SUBSPACE_ITER_v1

Algorithme 4 (indications des lignes sur subspace_iter_v1):
22 : Initialisations des entrées
22 : Initialisations des sorties
Paramètres initiaux:
38 : Initialisation de k à 0.
40 : Initialisation de eigsum à 0 (correspondant à *PercentReached* dans l'algo 4)
48 : Génération de m vecteurs aléatoires (/!\ la famille de vecteurs ne doit pas être liée)
49 : Orthonormalisation de la famille par le procédé de Schmidt
Boucle while :
54 : Incrémentation de k
56 : Calcul de Y = A*V
58 : Orthonormalisation de Y par le procédé de Schmidt et attribution à la variable V
61 : Projection de Rayleigh-Ritz appliquée à A et V
70 - 109 : Analyse de la convergence (Boucle while imbriquée):
92 : Sauvegarde des valeurs propres dans W (la sauvegarde des valeurs propres est dans V)
97 : Mise à jour de eigsum (correspondant à *PercentReached*)

```
function [ V, D, n_ev, it, itv, flag ] = subspace_iter_v1( A, m, percentage, eps, maxit ) % Définition des entrées et sorties
  % calcul de la norme de A (pour le critère de convergence d'un vecteur (gamma))
  normA = norm(A, 'fro');
  % trace de A
  traceA = trace(A);
  % valeur correspondant au pourcentage de la trace à atteindre
  vtrace = percentage*traceA;
  n = size(A,1);
  W = zeros(m,1);
  itv = zeros(m,1);
  % numéro de l'itération courante
  k = 0;   % Initialisation de k à 0
  % somme courante des valeurs propres
  eigsum = 0.0;   % Initialisation de eigsum à 0 (correspondant à PercentReached)
  % nombre de vecteurs ayant convergés
  nb_c = 0;
  % indicateur de la convergence
  conv = 0;
  % on génère un ensemble initial de m vecteurs orthogonaux
   Vr = randn(n, m); % Génération de m vecteurs aléatoires (la famille de vecteurs ne doit pas être liée)
  Vr = mgs(Vr); % Orthonormalisation de la famille par le procédé de Schmidt
  % rappel : conv = (eigsum >= trace) | (nb_c == m)
  while (~conv && k < maxit)
     k = k+1;   % Incrémentation de k
     %% Y <- A*V
```

```
Y = A*Vr; % Calcul de Y = A*V
%% orthogonalisation
    Vr = mgs(Y); % Orthonormalisation de Y par le procédé de Schmidt et attribution à la variable V

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

%% Quels vecteurs ont convergé à cette itération
analyse_cvg_finie = 0;
% nombre de vecteurs ayant convergé à cette itération
nbc_k = 0;
% nb_c est le dernier vecteur à avoir convergé à l'itération précédente
i = nb_c + 1;

while(~analyse_cvg_finie) % Analyse de la convergence
   % tous les vecteurs de notre sous-espace ont convergé
   % on a fini (sans avoir obtenu le pourcentage)
   if(i > m)
      analyse_cvg_finie = 1;
   else
      % est-ce que le vecteur i a convergé

      % calcul de la norme du résidu
      aux = A*Vr(:,i) - Wr(i)*Vr(:,i);
      res = sqrt(aux'*aux);

      if(res >= eps*normA)
         % le vecteur i n'a pas convergé,
         % on sait que les vecteurs suivants n'auront pas convergé non plus
         % => itération finie
         analyse_cvg_finie = 1;
      else
         % le_vecteur i a convergé
         % un de plus
         nbc_k = nbc_k + 1;
         % on le stocke ainsi que sa valeur propre
         W(i) = Wr(i);  % Sauvegarde des valeurs propres dans W

         itv(i) = k;

         % on met à jour la somme des valeurs propres
         eigsum = eigsum + W(i);  % Mise à jour de eigsum (correspondant à PercentReached)

         % si cette valeur propre permet d'atteindre le pourcentage
         % on a fini
         if(eigsum >= vtrace)
            analyse_cvg_finie = 1;
         else
            % on passe au vecteur suivant
            i = i + 1;
         end
      end
   end
```

```
        end

        % on met à jour le nombre de vecteurs ayant convergés
        nb_c = nb_c + nbc_k;

        % on a convergé dans l'un de ces deux cas
        conv = (nb_c == m) | (eigsum >= vtrace);

    end

    if(conv)
        % mise à jour des résultats
        n_ev = nb_c;
        V = Vr(:, 1:n_ev);
        W = W(1:n_ev);
        D = diag(W);
        it = k;
    else
        % on n'a pas convergé
        D = zeros(1,1);
        V = zeros(1,1);
        n_ev = 0;
        it = k;
    end
    % on indique comment on a fini
    if(eigsum >= vtrace)
        flag = 0;
    else if (n_ev == m)
            flag = 1;
        else
            flag = -3;
        end
    end
end
```

Partie III (Elisa et Nicolas)

Question 8: Nicolas

Pour une composante du produit de deux matrices de taille n, on réalise un produit de deux scalaires puis une somme de ces n éléments qui est de l'ordre de $O(n)$.
Puisque la matrice produit contient n^2 éléments, un produit matriciel simple est de l'ordre de $O(n^3)$.
Par récurrence, le calcul de $A\text{^}p$ est de l'ordre de $O((p-1)n^3)$.

Question 9: fait par Nicolas, testé par Elisa (fichier test_v0v1v2.m)

Question 10: Elisa

On observe que le nombre d'itérations et le temps d'exécution du script diminuent lorsque la valeur de p augmente sans que la qualité des couples propres ne change d'ordre de grandeur. Cela s'explique par le fait que p est un facteur d'accélération : à chaque itération on multiplie la matrice A p-fois par elle-même au lieu d'une fois, ainsi l'algorithme converge plus vite. De plus, orthonormaliser est très coûteux or la version v1 requiert d'orthonormaliser après chaque itération, ainsi en réduisant le nombre d'itérations on réduit le nombre d'opérations d'orthonormalisation. L'algorithme est ainsi moins coûteux et s'exécute plus rapidement.

Question 11: Nicolas

Dans l'algorithme de subspace_iter_v1 (ou v2), certains vecteurs sont susceptible de converger plus rapidement que d'autres vers les couples propres recherchés, ainsi continuer l'itération sur certains vecteurs va créer un écart dans l'ordre de grandeur de la norme résiduelle des couples propres.

Question 12: Nicolas

Avec l'algorithme subspace_iter_v3, lorsqu'un vecteur atteint une norme résiduelle satisfaisante, on stocke ces vecteurs et on itére l'algorithme sur les vecteurs restants, économisant des opérations.

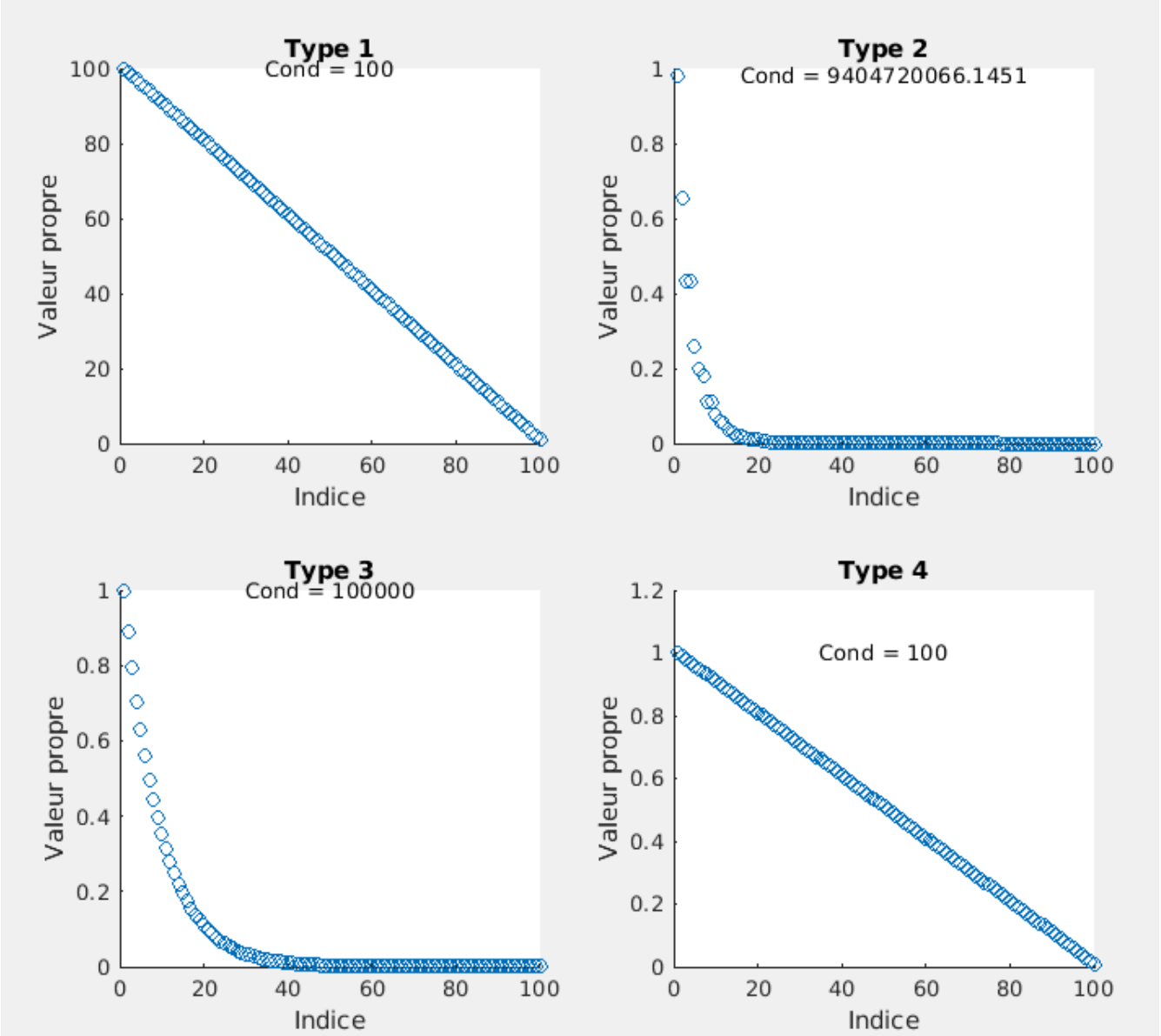
Question 13: Nicolas

Partie IV (Nicolas et Elisa)

Question 14: Nicolas

La différence entre les 4 types de matrices de taille n repose sur leurs conditionnements :
1 - Les valeurs propres sont générée par une séquence arithmétique décroissante de raison -1 allant de n à 1, son conditionnement est de 100
2 - Les valeurs propres sont générée selon une distribution exponentielle de paramètre alpha=10, son conditionnement est de l'ordre de 10^{10}
3 - Les valeurs propres sont générées par une séquence géométrique décroissant, son conditionnement est de 10^{5}.
4 - Les valeurs propres sont générées par une séquence linéaire comptant n valeurs propres allant de 1 à l'inverse de son conditionnement imposé à 100.

Généré à l'aide du script eigenvalues_distributions.m



Le conditionnement de la matrice traduit la stabilité de la solution du système linéaire Ax=b pour une perturbation de b donnée,
Dans le contexte de la recherche de valeurs propres, une matrices mal conditionnée peut engendrer des erreurs sur les valeurs propres estimées par les algorithmes itératifs.

Question 15: Elisa

SEANCE 2 : APPLICATION A LA COMPRESSION D'IMAGES

Question 1:

- Dimension de sigma_k : k*k
- Dimension de U_k : q*k
- Dimension de V_k : k*p

Question 2: