



RAPPORT

Séance-Projet 2 : Subspace Iteration Methods

DEGAIL Dylan
FERRATO Mathys
LAVAUUR Jonas

Département Sciences du Numérique - Première année
2021-2022

Table des matières

1	Limitations of the power method	3
1.1	Question 1	3
1.2	Question 2	3
1.3	Question 3	4
2	Extending the power method to compute dominant eigenspace vectors	5
2.1	Question 4	5
2.2	Question 5	6
2.3	Question 6	6
2.4	Question 7	6
3	subspace_iter_v2 and subspace_iter_v3 : toward an efficient solver	7
3.1	Block approach (subspace_iter_v2)	7
3.1.1	Question 8	7
3.1.2	Question 9	7
3.1.3	Question 10	7
3.2	Deflation method (subspace_iter_v3)	9
3.2.1	Question 11	9
3.2.2	Question 12	9
3.2.3	Question 13	9
4	Numerical experiments	10
4.1	Question 14	10
4.2	Question 15	11

Table des figures

1	Comparaison de power_v11 et eig	3
2	Algorithme power_v12	3
3	Comparaison de power_v11 et power_v12	4
4	Modifications de puissance_iterree.m	5
5	Résultats avec puissance_iterree.m modifié	5
6	Algo 4 avec lignes correspondantes à chaque étape	6
7	Résultats de subspace_iter_v2 avec $p = 1$	7
8	Résultats de subspace_iter_v2 avec $p = 10$	8
9	Résultats de subspace_iter_v2 avec $p = 100$	8
10	Précisions de chaque paire avec subspace_iter_v1	9
11	Précisions de chaque paire avec subspace iter v3 pour $m = 20$ et $m = 100$	9
12	Spectre en fonction du type des matrices pour $n = 20$	10
13	Spectre en fonction du type des matrices pour $n = 200$	10
14	$t = f(n)$ pour $\text{imat} = 1$	11
15	$t = f(n)$ pour $\text{imat} = 2$	12
16	$t = f(n)$ pour $\text{imat} = 3$	12
17	$t = f(n)$ pour $\text{imat} = 4$	13

1 Limitations of the power method

1.1 Question 1

```
n = 200 : imat = 1 : Temps eig = 2.000e-02
              Temps puissance itérée = 5.430e+00

              imat = 2 : Temps eig = 1.000e-02
              Temps puissance itérée = 9.000e-02

              imat = 3 : Temps eig = 1.000e-02
              Temps puissance itérée = 2.000e-01

              imat = 4 : Temps eig = 2.000e-02
              Temps puissance itérée = 5.380e+00

n = 400 : imat = 1 : Temps eig = 4.000e-02
              Temps puissance itérée = 5.177e+01

              imat = 2 : Temps eig = 4.000e-02
              Temps puissance itérée = 1.450e+00

              imat = 3 : Temps eig = 6.000e-02
              Temps puissance itérée = 1.410e+00

              imat = 4 : Temps eig = 4.000e-02
              Temps puissance itérée = 5.058e+01

n = 20 : imat = 1 : Temps eig = 0.000e+00
              Temps puissance itérée = 5.000e-02

              imat = 2 : Temps eig = 0.000e+00
              Temps puissance itérée = 3.000e-02

              imat = 3 : Temps eig = 0.000e+00
              Temps puissance itérée = 2.000e-02

              imat = 4 : Temps eig = 0.000e+00
              Temps puissance itérée = 3.000e-02
```

FIGURE 1 – Comparaison de power_v11 et eig

On remarque que pour n'importe quelles valeurs de n et $imat$, on a toujours **eig** plus rapide que l'algo de la puissance itérée **power_v11**.

1.2 Question 2

Pour n'avoir qu'un seul produit matrice×vecteur dans la boucle, on a réarrangé l'algorithme ainsi (voir **power_v11.m**) :

```
% 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
```

FIGURE 2 – Algorithme power_v12

De plus, on obtient bien qu'il est 2 fois plus rapide que le **power_v11** :

```
***** calcul avec la méthode de la puissance itérée *****

Temps puissance itérée = 1.200e-01
Nombre de valeurs propres pour attendre le pourcentage = 9
Nombre d'itérations pour chaque couple propre
couple 1 : 292
couple 2 : 264
couple 3 : 247
couple 4 : 275
couple 5 : 266
couple 6 : 281
couple 7 : 282
couple 8 : 264
couple 9 : 283
Qualité des valeurs propres (par rapport au spectre de la matrice) = [0.000e+00 , 1.554e-15]
Qualité des couples propres = [9.733e-09 , 1.487e-08]

Matrice 200 x 200 - type 3

***** calcul avec la méthode de la puissance itérée améliorée *****

Temps puissance itérée = 6.000e-02
Nombre de valeurs propres pour attendre le pourcentage = 9
Nombre d'itérations pour chaque couple propre
couple 1 : 292
couple 2 : 264
couple 3 : 247
couple 4 : 275
couple 5 : 266
couple 6 : 281
couple 7 : 282
couple 8 : 264
couple 9 : 283
Qualité des valeurs propres (par rapport au spectre de la matrice) = [0.000e+00 , 1.554e-15]
Qualité des couples propres = [9.733e-09 , 1.487e-08]
```

FIGURE 3 – Comparaison de power_v11 et power_v12

1.3 Question 3

On perd du temps en ne calculant qu'un seul couple propre par étape. La vitesse de convergence dépend du rapport entre la plus grande et la seconde plus grande valeur propre. Si les deux plus grandes valeurs propres sont proches, l'algorithme ne va pas converger rapidement.

2 Extending the power method to compute dominant eigenspace vectors

2.1 Question 4

Si on applique l'algorithme de la puissance itérée sur un ensemble de m vecteurs, V converge vers une matrice dont la première colonne sera le 1er vecteur propre dominant, et les autres colonnes seront aussi ce même vecteur propre. Ainsi, V converge vers une matrice composée de m fois le 1er vecteur propre dominant.

Pour le code de `puissance_iterree.m`, on a effectué les modifications suivantes :

```
AAt = A*A';
M = AAt;
|
V = randn(n,m);
V = mgs(V);

cv = false;
k = 0;
acc = 0;
H = V'*M*V;

while(~cv)
    H_old = H;
    V = M*V;
    for j = 1:m
        V(:,j) = V(:,j)/norm(V(:,j));
    end
    H = V'*M*V;
    k = k+1;
    acc = norm(M*V-V*H, 'fro')/norm(M, 'fro');
    cv = (acc < eps) || (k > imax);
end
```

FIGURE 4 – Modifications de `puissance_iterree.m`

Et, on obtient les résultats suivants (qui sont en accord avec notre conjecture) :

```
V =

-0.0599    0.0599    0.0599   -0.0599   -0.0599
 0.0346   -0.0346   -0.0346    0.0346    0.0346
-0.0214    0.0214    0.0214   -0.0214   -0.0214
 0.3219   -0.3219   -0.3219    0.3219    0.3219
 0.1993   -0.1993   -0.1993    0.1993    0.1993
 0.2938   -0.2938   -0.2938    0.2938    0.2938
-0.4116    0.4116    0.4116   -0.4116   -0.4116
 0.2014   -0.2014   -0.2014    0.2014    0.2014
 0.2189   -0.2189   -0.2189    0.2189    0.2189
-0.1447    0.1447    0.1447   -0.1447   -0.1447
-0.0141    0.0141    0.0141   -0.0141   -0.0141
-0.6007    0.6007    0.6007   -0.6007   -0.6007
-0.2503    0.2503    0.2503   -0.2503   -0.2503
 0.2445   -0.2445   -0.2445    0.2445    0.2445
-0.0524    0.0524    0.0524   -0.0524   -0.0524
```

FIGURE 5 – Résultats avec `puissance_iterree.m` modifié

2.2 Question 5

$A \in M_n(\mathbf{R})$ et $V \in M_{n,m}(\mathbf{R})$.
 Donc $Y = AV \in M_{n,m}(\mathbf{R})$ d'où $V^T \in M_{m,n}(\mathbf{R})$.
 Ainsi $H = V^T Y \in M_{m,m}(\mathbf{R})$.

Et comme m correspond au nombre max de couples propres calculés, ce n'est pas gênant de calculer tous les couples propres de H .

2.3 Question 6

(voir `subspace_iter_v0.m`)

2.4 Question 7

Les lignes du programme `subspace_iter_v1.m`, correspondantes à chaque étape, figurent en rouge dans l'image ci-dessous :

Algorithm 4 Subspace iteration method v1 with Raleigh-Ritz projection

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

Generate an initial set of m orthonormal vectors $V \in \mathbb{R}^{n \times m}$; $k = 0$; *PercentReached* = 0

repeat lignes 48-49 ligne 38 ligne 40

$k = k + 1$ ligne 54

Compute Y such that $Y = A \cdot V$ ligne 56

$V \leftarrow$ orthonormalisation of the columns of Y ligne 58

Rayleigh-Ritz projection applied on matrix A and orthonormal vectors V ligne 61

Convergence analysis step: save eigenpairs that have converged and update *PercentReached* lignes 64-112

until (*PercentReached* > *PercentTrace* or $n_{ev} = m$ or $k > \text{MaxIter}$) ligne 115

FIGURE 6 – Algo 4 avec lignes correspondantes à chaque étape

3 subspace_iter_v2 and subspace_iter_v3 : toward an efficient solver

3.1 Block approach (subspace_iter_v2)

3.1.1 Question 8

- Le coût :

Nombre d'opérations pour calculer A^p avec $A \in M_n(\mathbf{R})$: $2(p-1)n^3$

Nombre d'opérations pour calculer $A^p V$ avec $V \in M_{n,m}(\mathbf{R})$:

- $2(p-1)n^3$ pour A^p
 - $2n^2m$ pour faire un produit avec V
- $\Rightarrow 2n^2[m + (p-1)n]$ opérations au total pour $A^p V$

- Pour réduire ce coût :

- Sans considérer l'algorithme, seulement en terme de calcul matriciel :

Effectuer d'abord AV puis en multipliant par A^{p-1} à gauche on arrive à un nombre d'opérations de $2pn^2m$. Par exemple en prenant $n = 200$; $m = 100$; $p = 10$ ou 1000 on arrive à un nombre d'opérations 2 fois inférieur par cette façon.

- En pensant à l'aspect algorithmique :

On peut effectuer le calcul de A^p en dehors de la boucle.

3.1.2 Question 9

(voir `subspace_iter_v2.m`)

3.1.3 Question 10

```
>> test_v2

Matrice 200 x 200 - type 1

***** calcul avec subspace iteration v2 *****

Temps subspace iteration v2 = 4.200e-01
Nombre d'itérations : 242
Nombre de valeurs propres pour attendre le pourcentage = 11
Nombre d'itérations pour chaque couple propre
couple 1 : 132
couple 2 : 132
couple 3 : 137
couple 4 : 152
couple 5 : 152
couple 6 : 166
couple 7 : 179
couple 8 : 198
couple 9 : 198
couple 10 : 200
couple 11 : 242
Qualité des valeurs propres (par rapport au spectre de la matrice) = [5.585e-14 , 1.340e-13]
Qualité des couples propres = [6.990e-13 , 8.393e-08]
```

FIGURE 7 – Résultats de subspace_iter_v2 avec $p = 1$

```

>> test_v2

Matrice 200 x 200 - type 1

***** calcul avec subspace iteration v2 *****

Temps subspace iteration v2 = 9.000e-02
Nombre d'itérations : 24
Nombre de valeurs propres pour attendre le pourcentage = 11
Nombre d'itérations pour chaque couple propre
couple 1 : 14
couple 2 : 14
couple 3 : 15
couple 4 : 16
couple 5 : 17
couple 6 : 19
couple 7 : 19
couple 8 : 19
couple 9 : 22
couple 10 : 23
couple 11 : 24
Qualité des valeurs propres (par rapport au spectre de la matrice) = [1.914e-15 , 1.093e-13]
Qualité des couples propres = [8.725e-13 , 7.630e-08]

```

FIGURE 8 – Résultats de subspace_iter_v2 avec $p = 10$

```

>> test_v2

Matrice 200 x 200 - type 1

***** calcul avec subspace iteration v2 *****

Temps subspace iteration v2 = 4.000e-02
Nombre d'itérations : 3
Nombre de valeurs propres pour attendre le pourcentage = 11
Nombre d'itérations pour chaque couple propre
couple 1 : 2
couple 2 : 2
couple 3 : 2
couple 4 : 2
couple 5 : 2
couple 6 : 2
couple 7 : 2
couple 8 : 3
couple 9 : 3
couple 10 : 3
couple 11 : 3
Qualité des valeurs propres (par rapport au spectre de la matrice) = [0.000e+00 , 1.631e-14]
Qualité des couples propres = [9.576e-16 , 2.807e-08]

```

FIGURE 9 – Résultats de subspace_iter_v2 avec $p = 100$

Le temps d'exécution diminue et la qualité des couples propres augmentent lorsque p augmente. Pour des valeurs de $p > 133$, le pourcentage de la trace n'est plus atteint quelque soit le pourcentage voulu pour $m = \frac{n}{10}$ ou $\frac{n}{4}$ ou $\frac{n}{2}$ avec $A \in M_n(\mathbf{R})$ et $n = 200$.

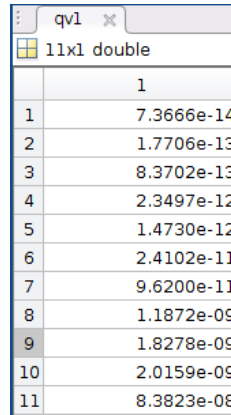
Explications :

En augmentant p , on réduit le coût de l'orthonormalisation à chaque itération ce qui diminue le temps de calcul.

Pour de grandes valeurs de p , le calcul de A^p est trop coûteux et l'algorithme ne converge plus donc le pourcentage de la trace exigé n'est jamais atteint.

3.2 Deflation method (subspace_iter_v3)

3.2.1 Question 11



	1
1	7.3666e-14
2	1.7706e-13
3	8.3702e-13
4	2.3497e-12
5	1.4730e-12
6	2.4102e-11
7	9.6200e-11
8	1.1872e-09
9	1.8278e-09
10	2.0159e-09
11	8.3823e-08

FIGURE 10 – Précisions de chaque paire avec subspace_iter_v1

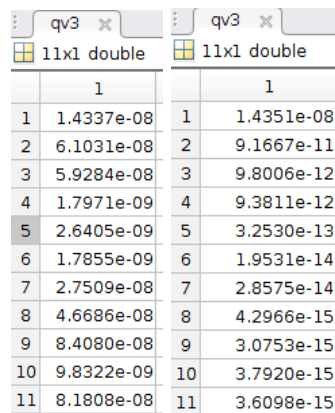
La précision sur les couples propres diminue au fur et à mesure des couples propres trouvés car, une fois qu'un vecteur propre a convergé, on le garde dans la matrice. Ainsi, en calculant le prochain vecteur convergeant à partir de cette matrice, une erreur s'ajoute et diminue la précision.

3.2.2 Question 12

On conjecture qu'avec **subspace iter v3.m** ça va être l'inverse, c'est-à-dire que la précision sur les couples augmente au fur et à mesure des couples propres trouvés. En effet, dès qu'un vecteur propre a convergé, on l'enlève de V_{nc} et on le place dans V_c . Puis, on réitère seulement sur V_{nc} donc la précision augmente.

Cependant, si m est petit, cette conjecture ne sera pas vérifiée et les vecteurs propres convergeront à peu près en même temps puisqu'il y en a peu.

On obtient alors sur MATLAB les résultats suivants qui confirment nos conjectures :



	1
1	1.4337e-08
2	6.1031e-08
3	5.9284e-08
4	1.7971e-09
5	2.6405e-09
6	1.7855e-09
7	2.7509e-08
8	4.6686e-08
9	8.4080e-08
10	9.8322e-09
11	8.1808e-08

	1
1	1.4351e-08
2	9.1667e-11
3	9.8006e-12
4	9.3811e-12
5	3.2530e-13
6	1.9531e-14
7	2.8575e-14
8	4.2966e-15
9	3.0753e-15
10	3.7920e-15
11	3.6098e-15

FIGURE 11 – Précisions de chaque paire avec subspace iter v3 pour $m = 20$ et $m = 100$

3.2.3 Question 13

(voir subspace_iter_v3.m)

4 Numerical experiments

4.1 Question 14

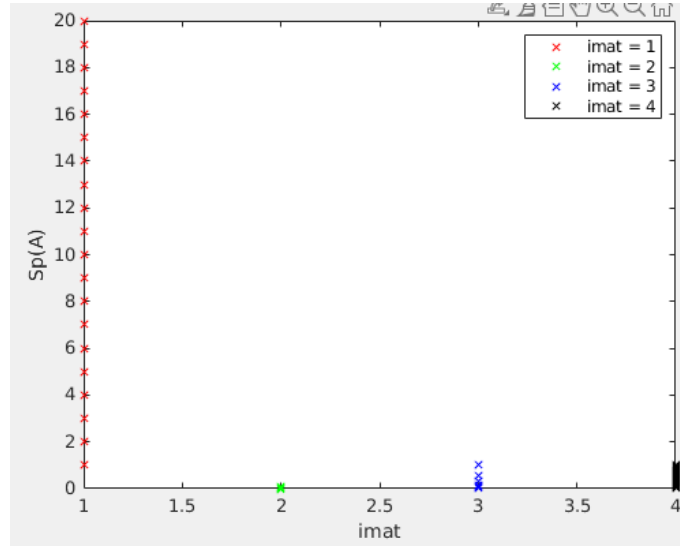


FIGURE 12 – Spectre en fonction du type des matrices pour $n = 20$

Pour $A \in M_n(\mathbf{R})$ avec $n = 20$, on observe que chaque type de matrice a une répartition de valeurs propres différentes :

- pour $imat = 1$: le spectre est réparti de 1 à 20 de manière uniforme
- pour $imat = 2$: le spectre est concentré en 0
- pour $imat = 3$: le spectre est réparti de 0 à 1 (s'approchant d'une échelle logarithmique)
- pour $imat = 4$: le spectre est réparti de 0 à 1 de manière uniforme

On observe les mêmes conclusions pour $n = 200$ mais cela est moins visible :

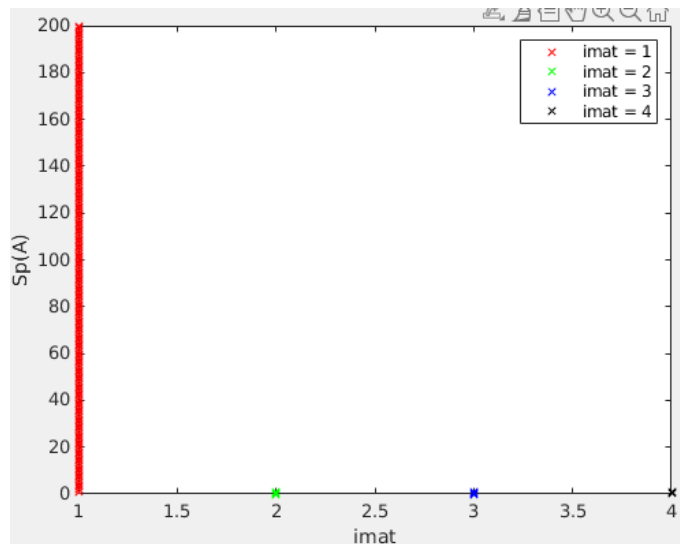


FIGURE 13 – Spectre en fonction du type des matrices pour $n = 200$

4.2 Question 15

Pour répondre à cette question nous avons trouvé pertinent de tracer le temps d'exécution de tous les algos en fonction de la taille la matrice et ce pour les 4 types de matrice. Il a été convenu d'utiliser une **échelle logarithmique** pour le temps afin de mieux observer les différences entre les algorithmes car ces derniers ont souvent de faibles temps d'exécution.

Les graphiques suivants ont été réalisés en prenant les paramètres suivants :

- tolérance $eps = 10^{-8}$
- nombre maximum d'itérations $maxit = 10^4$
- nombre maximum de couples propres calculés $m = 20$ si pour $A \in M_n(\mathbf{R}), n = 450$ et $m = 50$ sinon.
- pourcentage de la trace à atteindre de 10%.
- A est élevée à la puissance $p = 5$ pour **subspace_iter_v2.m** et **subspace_iter_v3.m**.

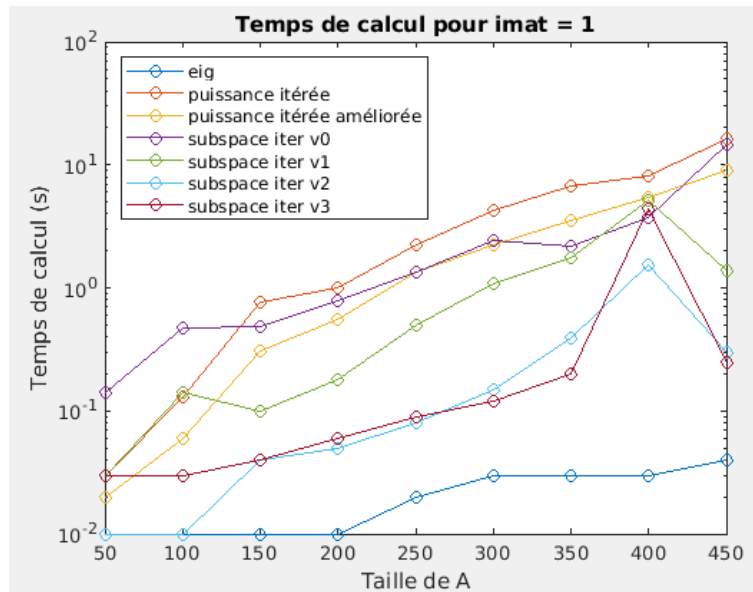


FIGURE 14 – $t = f(n)$ pour $imat = 1$

On remarque alors que pour $imat = 1$:

- **eig** est l'algo le plus rapide $\forall n$.
- **power_v11** est l'algo le moins rapide pour n assez grand.
- **power_v12** est plus rapide que **power_v11** d'un écart constant en échelle log donc qui augmente avec n en échelle normale.
- **subspace_iter_v0** est de même performance que **power_v12** sauf pour n petit où il l'est moins.
- **subspace_iter_v1** est toujours au moins plus rapide que **subspace_iter_v0**.
- **subspace_iter_v2** et **subspace_iter_v3** ont des performances très proches.

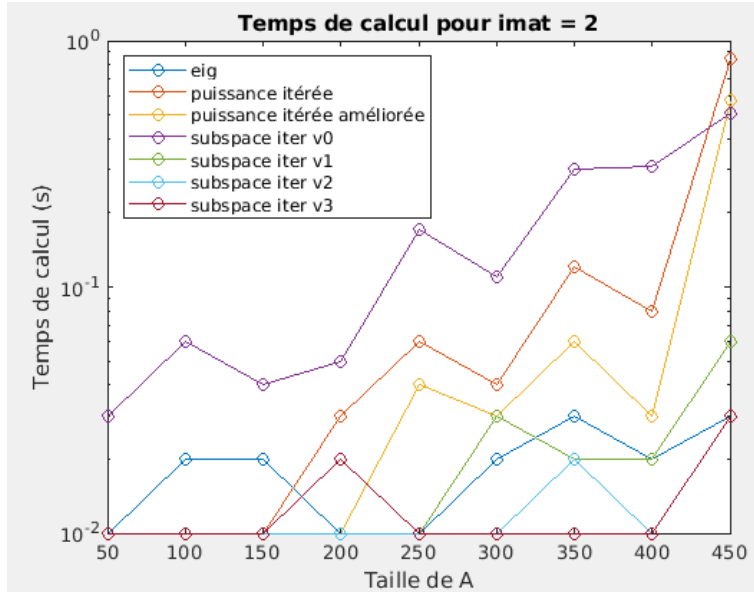


FIGURE 15 – $t = f(n)$ pour $\text{imat} = 2$

On remarque alors que pour $\text{imat} = 2$:

- **subspace_iter_v0** est l'algo le moins performant sauf en $n = 450$ où il est plus rapide que les **power**.
- **eig** et **subspace_iter_v1** ont des performances similaires, meilleures que celles de **power_v11**.
- **subspace_iter_v2** et **subspace_iter_v3** sont les plus rapides et ont des performances similaires sauf en $n = 200$ où **subspace_iter_v2** est 10% plus performant que **subspace_iter_v3** et en $n = 350$ où c'est le comportement inverse.

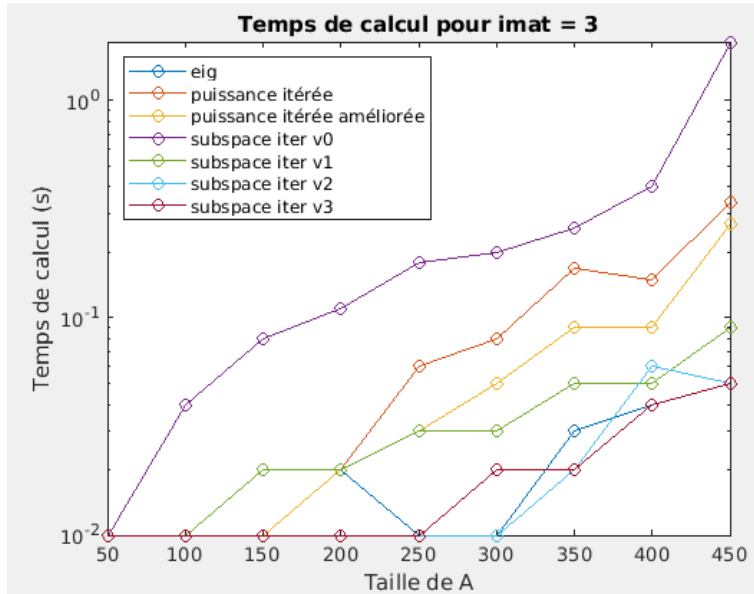


FIGURE 16 – $t = f(n)$ pour $\text{imat} = 3$

On remarque alors que pour $\text{imat} = 3$:

- **subspace_iter_v0** est l'algo le moins performant $\forall n$.
- **eig**, **subspace_iter_v2** et **subspace_iter_v3** ont des performances très similaires et lorsque n est grand, **subspace_iter_v3** est meilleur que **eig**, lui même meilleur que **subspace_iter_v2**
- **subspace_iter_v1** devient de plus en plus performant par rapport aux méthodes **power** lorsque n augmente.

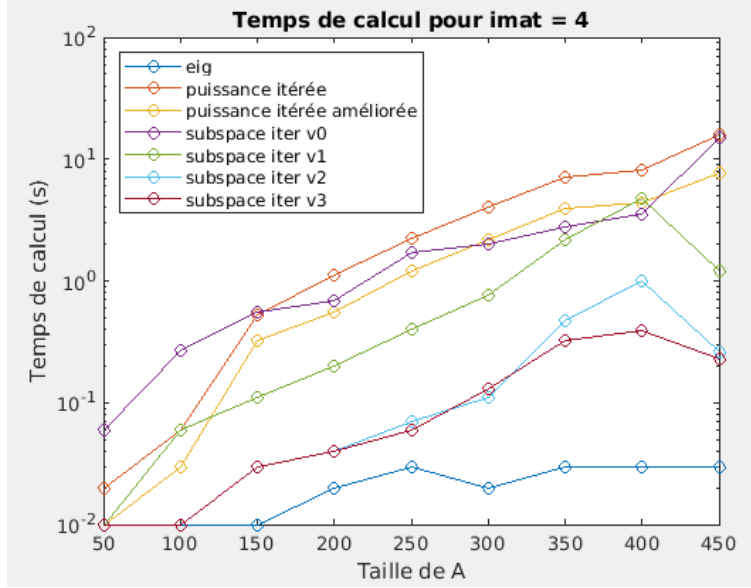


FIGURE 17 – $t = f(n)$ pour $\text{imat} = 4$

On remarque alors que pour $\text{imat} = 4$, on obtient les mêmes observations que pour $\text{imat} = 1$.