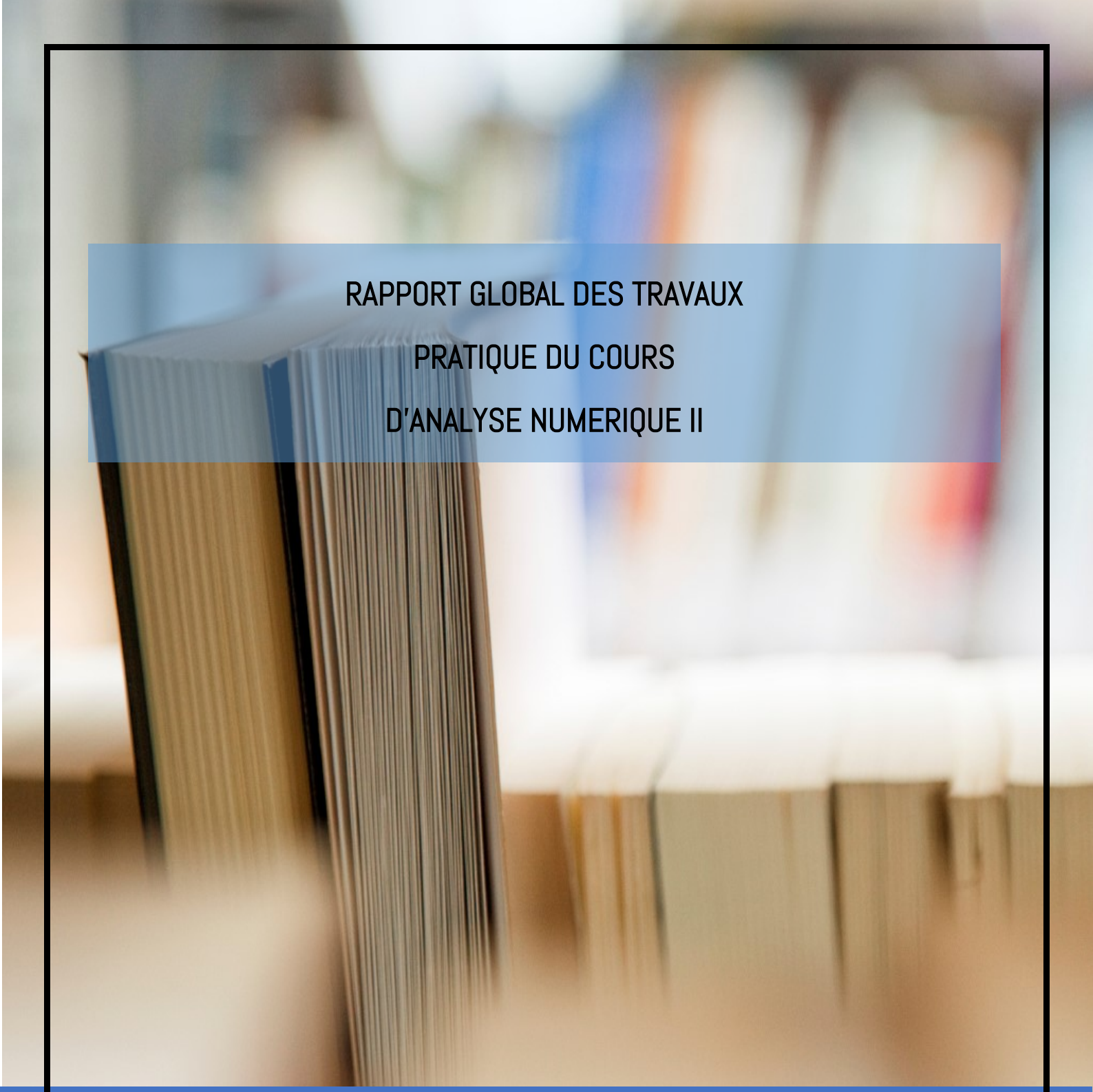


RAPPORT GLOBAL DES TRAVAUX PRATIQUE DU COURS D'ANALYSE NUMERIQUE II

Membres du Groupe : GROUPE 4

- DAGANG CHESSE FRED
- FANDIO ESDRAS (Chef)
- FOSSO RUDY
- KOUETCHOU JORDAN
- MBATHE PAUL
- SINGNING EDDY

Supervise par : Dr MOUKOUOP

Introduction

Dans le cadre de l'UE d'Analyse numérique, il nous a été donné à faire Tout au long de ce semestre plusieurs Travaux Pratique portant notamment sur les tests de résolution d'équation, et les résolutions d'équation différentielle par les méthodes des Différences Finies en dimension 1, Différences Finies en dimension 2, Volume Finis en dimension 1 et Volumes Finis en dimension 2. Ce rapport sert de rapport général de tous ces travaux précédemment effectués et en présentera les résultats

I. RAPPORT DES MEMBRES

Membre	Contribution
FANDIO	Fonction solver et initialisation matrice des différence fini 2D, élaboration de l'architecture des projets et des fonctions de test pour les différences finis 2D, rédaction du rapport
MBATHE	Initialisation des matrices volume fini 2D, fonction solver différence finis 1D, graphe d'erreur différence finis 2D
KOUETCHOU	Génération de tous les cas de tous dans les projets, fonctions de test, graphe d'erreur volumes finis 2D, rédaction du rapport
DAGANG	Initialisation des matrices volume fini 1D, fonction de test pour les polynômes, solver volume fini 1D, rapport sur les tests
SINGNING	Génération de tous les cas de test dans les projets, rédaction du rapport
FOSSO RUDY	Fonctions de test pour la racine, volumes finis

II. TEST DE RÉSOLUTION D'ÉQUATIONS

Il était question ici de produire les données de test et d'analyser les résultats sur les fonctions de test de nos différentes fonctions à tester.

Parmi nos fonctions à tester nous avons donc :

- Somme
- Racine carrée
- Equation de second degré

A. Somme

- **Fonction à tester :**
- **Dimension : 8 8**
- **Nombres de cas testés : 64,** pour cette partie, on a utilisé les scenarios générés par jenny que nous avons enregistrés dans un fichier puis nous avons généré les valeurs correspondantes à ces différents scénarios en matchant les scénarios de chaque variable avec des valeurs prédéfinis dans deux tableaux de taille 8 où chaque élément correspondait à la valeur à prendre pour un scénario d'une donnée précise.
- **Nombre de cas échoués : 9/64** : Ces cas correspondent aux scénarios où entraînant
- **Fonction de test :**

Pour cette fonction, on a calculé pour notre test la valeur de g(f(x)) qu'on a ensuite comparé à y qui est le résultat attendu pour cette valeur.

- **Tolérance :**
- **Résultat attendu :** le résultat attendu ici est y

B. Résultats du test

****

*

* Donnees de test pour la résolution de la somme f(x,y) = x + y *

*

****

x	y	Tolérance	Résultat attendu	Résultat obtenu	Oracle
0.0	1.0E26	1.0E-6	1.0E26	1.0E26	true
1.0E-6	2.718281828459045	1.0E-6	2.718281828459045	2.718281828459045	true
-1.0E-5	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true
1.0E19	-1.0E-7	1.0E-6	-1.0E-7	0.0	false
-1.0E15	97.0	1.0E-6	97.0	97.0	true
3.141592653589793	3.142857142857143	1.0E-6	3.142857142857143	3.1428571428571423	true
4.583333333333333	0.0	1.0E-6	0.0	0.0	true
30.0	1.0E-6	1.0E-6	1.0E-6	1.0000000010279564E-6	true
0.0	0.0	1.0E-6	0.0	0.0	true
1.0E-6	0.0	1.0E-6	0.0	0.0	true
-1.0E-5	0.0	1.0E-6	0.0	0.0	true
1.0E19	0.0	1.0E-6	0.0	0.0	true

-1.0E15	0.0	1.0E-6	0.0	0.0	true
3.141592653589793	0.0	1.0E-6	0.0	0.0	true
4.583333333333333	1.0E-6	1.0E-6	1.0E-6	1.000000000139778E-6	true
30.0	0.0	1.0E-6	0.0	0.0	true
0.0	-1.0E-7	1.0E-6	-1.0E-7	-1.0E-7	true
1.0E-6	1.0E26	1.0E-6	1.0E26	1.0E26	true
0.0	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true
0.0	2.718281828459045	1.0E-6	2.718281828459045	2.718281828459045	true
0.0	3.142857142857143	1.0E-6	3.142857142857143	3.142857142857143	true
0.0	97.0	1.0E-6	97.0	97.0	true
-1.0E-5	97.0	1.0E-6	97.0	97.0	true
1.0E19	97.0	1.0E-6	97.0	0.0	false
-1.0E15	3.142857142857143	1.0E-6	3.142857142857143	3.125	false
3.141592653589793	97.0	1.0E-6	97.0	97.0	true
4.583333333333333	97.0	1.0E-6	97.0	97.0	true
30.0	97.0	1.0E-6	97.0	97.0	true

30.0	1.0E26	1.0E-6	1.0E26	1.0E26	true
30.0	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true
30.0	2.718281828459045	1.0E-6	2.718281828459045	2.718281828459048	true
1.0E-6	97.0	1.0E-6	97.0	97.0	true
-1.0E-5	3.142857142857143	1.0E-6	3.142857142857143	3.142857142857143	true
1.0E19	3.142857142857143	1.0E-6	3.142857142857143	0.0	false
-1.0E15	2.718281828459045	1.0E-6	2.718281828459045	2.75	false
3.141592653589793	2.718281828459045	1.0E-6	2.718281828459045	2.7182818284590446	true
4.583333333333333	3.142857142857143	1.0E-6	3.142857142857143	3.1428571428571432	true
3.141592653589793	1.0E-6	1.0E-6	1.0E-6	1.000000000139778E-6	true
4.583333333333333	-1.0E-7	1.0E-6	-1.0E-7	-1.0000000028043132E-7	true
4.583333333333333	1.0E26	1.0E-6	1.0E26	1.0E26	true
4.583333333333333	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true
1.0E-6	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true
-1.0E-5	2.718281828459045	1.0E-6	2.718281828459045	2.718281828459045	true
1.0E19	2.718281828459045	1.0E-6	2.718281828459045	0.0	false

-1.0E15	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true
-1.0E15	1.0E-6	1.0E-6	1.0E-6	0.0	false
3.141592653589793	-1.0E-7	1.0E-6	-1.0E-7	-9.999999983634211E-8	true
3.141592653589793	1.0E26	1.0E-6	1.0E26	1.0E26	true
1.0E-6	3.142857142857143	1.0E-6	3.142857142857143	3.142857142857143	true
-1.0E-5	1.0E26	1.0E-6	1.0E26	1.0E26	true
1.0E19	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true
1.0E19	1.0E-6	1.0E-6	1.0E-6	0.0	false
-1.0E15	-1.0E-7	1.0E-6	-1.0E-7	0.0	false
1.0E-6	-1.0E-7	1.0E-6	-1.0E-7	-1.0E-7	true
-1.0E-5	-1.0E-7	1.0E-6	-1.0E-7	-1.0000000000000074E-7	true
-1.0E-5	1.0E-6	1.0E-6	1.0E-6	1.0000000000000006E-6	true
-1.0E15	1.0E26	1.0E-6	1.0E26	1.0E26	true
1.0E-6	1.0E-6	1.0E-6	1.0E-6	1.0E-6	true
1.0E19	1.0E26	1.0E-6	1.0E26	1.0E26	true
3.141592653589793	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true

1.0E19	1.0E26	1.0E-6	1.0E26	1.0E26	true
3.141592653589793	-1.0E24	1.0E-6	-1.0E24	-1.0E24	true
4.583333333333333	2.718281828459045	1.0E-6	2.718281828459045	2.7182818284590455	true
30.0	3.142857142857143	1.0E-6	3.142857142857143	3.142857142857146	true

```
**********************************************************************************************************************************************
*                                                                                                                                            *
*                              Total Tests Ratés: 9/ 64            Total Tests Réussis: 55/ 64                                           *
*                                                                                                                                            *
**********************************************************************************************************************************************
```

Les 64 scénarios ayant été testés nous avons obtenu comme vous pouvez le constater sur les images précédentes 55 cas réussis sur les 64 prévus ; soit environ **85.9%** de réussite pour le calcul de f(x).

C. Racine Carrée

- Fonction à tester : racine de x
- Dimension : 8
- **Nombres de cas testés : 8,** les données à tester ont été enregistré dans une liste
- Nombre de cas échoués : 0
- Fonction de test : x2
- Tolérance : 10-8
- Résultat attendu : x si x>= 0 ; null sinon

Résultats du test

```
**********************************************************************************************************************
*                                                                                                                    *
*                          Donnees de test pour la recherche de la racine de x                                       *
*                                                                                                                    *
**********************************************************************************************************************

+---------------------+---------------------+---------------------+--------------------------------+---------+
| x                   | Tolérance           | Résultat attendu    | Résultat obtenu                | Oracle  |
+---------------------+---------------------+---------------------+--------------------------------+---------+
| 0.0                 | 1.0E-8              | 0.0                 | 0.0                            | true    |
+---------------------+---------------------+---------------------+--------------------------------+---------+
| 1.0E-6              | 1.0E-8              | 1.0E-6              | 1.0E-6                         | true    |
+---------------------+---------------------+---------------------+--------------------------------+---------+
| -1.0E-5             | 1.0E-8              | null                | null                           | true    |
+---------------------+---------------------+---------------------+--------------------------------+---------+
| 1.0E41              | 1.0E-8              | 1.0E41              | 1.0E41                         | true    |
+---------------------+---------------------+---------------------+--------------------------------+---------+
| -1.0E15             | 1.0E-8              | null                | null                           | true    |
+---------------------+---------------------+---------------------+--------------------------------+---------+
| 3.1415926535889793  | 1.0E-8              | 3.1415926535889793  | 3.1415926535897927             | true    |
+---------------------+---------------------+---------------------+--------------------------------+---------+
| 4.583333333333333   | 1.0E-8              | 4.583333333333333   | 4.583333333333333              | true    |
+---------------------+---------------------+---------------------+--------------------------------+---------+
| 30.0                | 1.0E-8              | 30.0                | 30.0                           | true    |
+---------------------+---------------------+---------------------+--------------------------------+---------+

**********************************************************************************************************************
*                                                                                                                    *
*                      Total Tests Ratés: 0/ 8            Total Tests Réussis: 8/ 8                              *
*                                                                                                                    *
**********************************************************************************************************************
```

Les 8 scénarios ayant été testés nous avons obtenu comme vous pouvez le constater sur l'image précédentes 8 cas réussis sur les 8 prévus ; soit **100%** de réussite pour le calcul de f(x).

D. Equation Second Degré

- Fonction à tester : ax2 + bx + c = 0
- Dimension : 8 8 8
- **Nombre de cas testés : 68,** pour cette partie, nous avons utilisé les scenarios générés par jenny que nous avons enregistrés dans un fichier puis nous avons généré les valeurs correspondantes à ces différents scénarios en matchant les scénarios de chaque variable avec des valeurs prédéfinies dans trois tableaux de taille 8 où chaque élément correspondait à la valeur à prendre pour un scénario d'une donnée précise.
- **Nombre de cas échoués : 20 / 68 soit un taux de réussite de 70 %** et les 20 cas ratés ici correspondent aux cas ou b très grand devant 4ac
- Fonction de test :

  Cette fonction prend en paramètre les solutions obtenues avec la fonction à tester ainsi que les paramètres :

  - Si c <> 0 cette fonction calcule ax2 + bx et trouve l'erreur relative |(ax2+bx+c)/c| et la compare à la tolérance
  - Si c = 0 , cette fonction compare l'une des solutions à 0 et pour l'autre calcule et trouve l'erreur relative |(ax+b)/b| et la compare à la tolérance
  - Si c = 0 et a<>0 ,b<>0 , cette fonction compare la solution obtenue avec la fonction à tester avec 0 (tolérance nulle).
  - Si a = b = c = 0 , cette fonction vérifie si la solution obtenue avec la fonction à tester est nulle.

- Tolérance : 10^-5

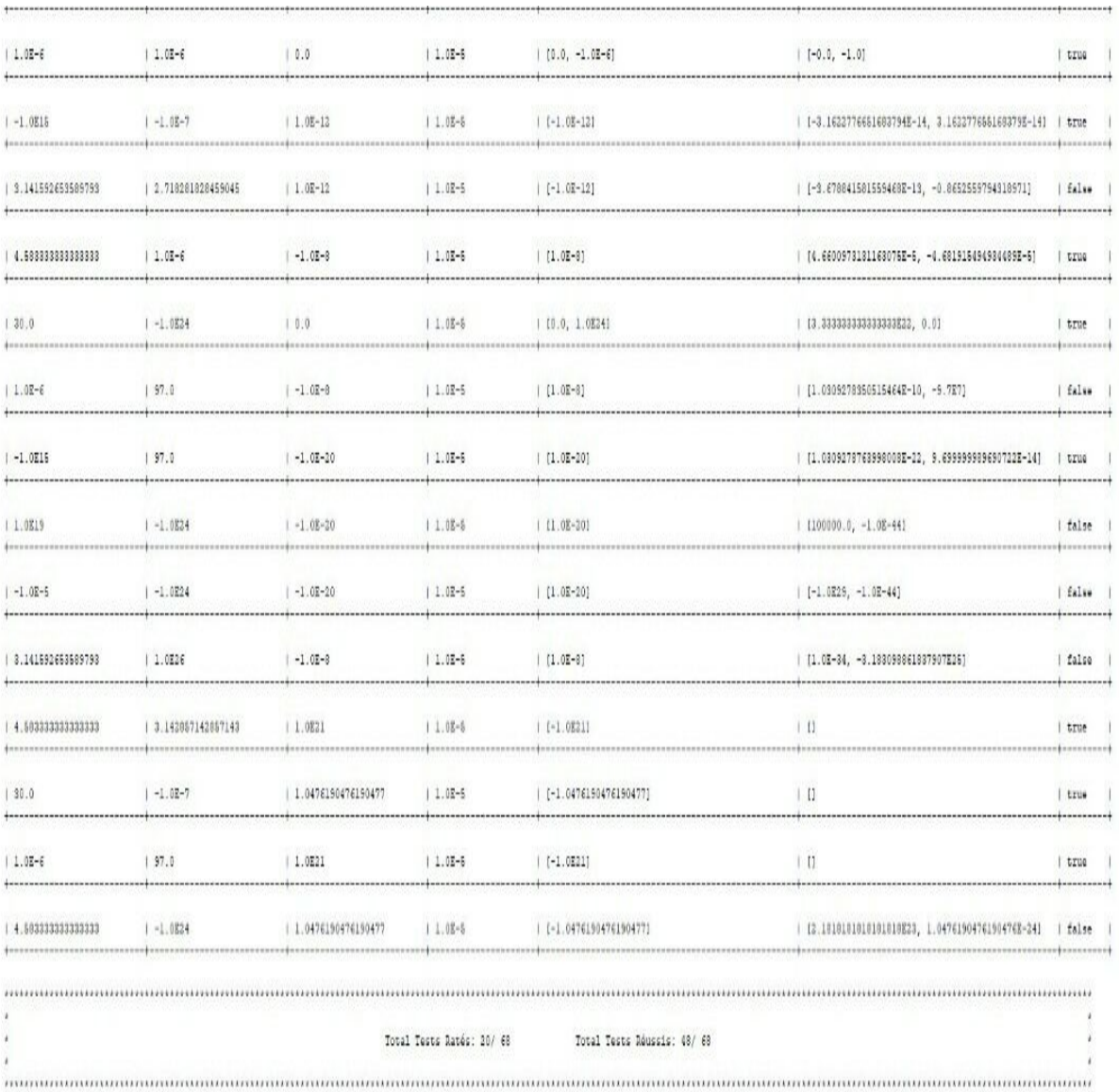
- Résultat attendu :

  - ❖ un arrayList contenant -c si c#0
  - ❖ un arrayList contenant {0.0,-b} si c=0 et a, b # 0
  - ❖ un arrayList contenant 0.0 si a#0 et b=c=0
  - ❖ null si a=b=c=0

Résultat des tests

Donnees de test pour la résolution de l'éqution aX^2+bX+c=0

a	b	c	Tolérance	Résultat attendu	Résultat obtenu	Oracle
0.0	1.0E26	-1.0E-20	1.0E-6	[1.0E-20]	[9.999999999999998E-47]	true
1.0E-6	-1.0E-7	2.6457613110645907	1.0E-6	[-2.6457613110645907]	[]	true
-1.0E-5	0.0	0.0	1.0E-6	[0.0]	[0.0]	true
1.0E19	1.0E-6	1.0E-12	1.0E-6	[-1.0E-12]	[]	true
-1.0E15	-1.0E24	-1.0E-8	1.0E-6	[1.0E-8]	[-1.0E9, -1.0E-32]	false
3.1415926535897793	3.1428571428571428	143.0	1.0E-6	[-143.0]	[]	true
4.583333333333333	97.0	1.0E21	1.0E-5	[-1.0E21]	[]	true
30.0	2.7182818284590045	1.0476190476190477	1.0E-6	[-1.0476190476190477]	[]	true
0.0	0.0	1.0E-12	1.0E-6	[-1.0E-12]	[]	true
1.0E-6	0.0	-1.0E-8	1.0E-5	[1.0E-8]	[0.1, -0.1]	true
-1.0E-5	1.0E-6	2.6457613110645907	1.0E-6	[-2.6457613110645907]	[-514.3106747912036, 514.4106747912036]	true
1.0E19	0.0	143.0	1.0E-6	[-143.0]	[]	true



1.0E-6	1.0E-6	0.0	1.0E-6	[0.0, -1.0E-6]	[-0.0, -1.0]	true
-1.0E16	-1.0E-7	1.0E-13	1.0E-6	[-1.0E-121	[-3.162277666168379E-14, 3.162277666168379E-14]	true
3.141592653589793	2.718281828459045	1.0E-12	1.0E-5	[-1.0E-12]	[-3.678841581559468E-13, -0.8652559794319571]	false
4.583333333333333	1.0E-6	-1.0E-8	1.0E-6	[1.0E-8]	[4.660097131116307SE-6, -4.681516494984689E-5]	true
30.0	-1.0E24	0.0	1.0E-6	[0.0, 1.0E24]	[3.333333333333333E22, 0.0]	true
1.0E-6	97.0	-1.0E-8	1.0E-5	[1.0E-8]	[1.0309278350515464E-10, -9.7E7]	false
-1.0E16	97.0	-1.0E-20	1.0E-6	[1.0E-20]	[1.0309278776898003SE-22, 9.699999898690722E-14]	true
1.0E19	-1.0E24	-1.0E-20	1.0E-6	[1.0E-201	[100000.0, -1.0E-441	false
-1.0E-5	-1.0E24	-1.0E-20	1.0E-5	[1.0E-20]	[-1.0E26, -1.0E-44]	false
3.141592653589793	1.0E26	-1.0E-8	1.0E-6	[1.0E-8]	[1.0E-84, -3.183098861837907E25]	false
4.583333333333333	3.142857142857143	1.0E21	1.0E-6	[-1.0E311	[]	true
30.0	-1.0E-7	1.0476190476190477	1.0E-5	[-1.0476190476190477]	[]	true
1.0E-6	97.0	1.0E21	1.0E-6	[-1.0E21]	[]	true
4.583333333333333	-1.0E24	1.0476190476190477	1.0E-6	[-1.0476190476190477]	[2.181818181818181E23, 1.0476190476190476E-24]	false

Total Tests Ratés: 20/ 68 Total Tests Réussis: 48/ 68

➢ Observations :

· Sur la figure présentation les tests, le résultat obtenu que nous avons affiché est l'ensemble des solutions obtenues tandis que le résultat attendu est selon les cas -c ou {0, -b} ou 0. Cela a été fait sciemment afin de pouvoir vérifier à la main ou à l'aide de calculatrices en ligne si les solutions obtenues étaient correctes.

· Sur les 68 cas de tests effectués, 20 cas de tests ont échoué. Au départ, nous avions 29 cas de tests échoués. Ces cas ont été résolu en mettant la parenthèse au niveau du dénominateur (2*a) des formules des racines des équations. Les 20 cas de tests échoués restant étaient dû au fait que lorsque b2>>4ac, racine(b2-4ac)=b2 , dans l'ensemble des pseudo réels de l'ordinateur. Ceci a pour conséquence d'induire que 0 est une solution de l'équation ce qui est généralement faux (lorsque a<>0 et c<>0) ce qui fait donc rater le test. Afin de corriger cela nous avons procédé à un développement limité c'est-à-dire en prenant le cas b>0 (sachant que le cas b<0) s'en déduit :

$$\frac{-b+\sqrt{b^2-4ac}}{2a} = \frac{-b+b\sqrt{1-\frac{4ac}{b^2}}}{2a} = \frac{b\left(-1+\sqrt{1-\frac{4ac}{b^2}}\right)}{2a} \approx \frac{b\left(-1+1-\frac{2ac}{b^2}\right)}{2a} \approx \frac{\left(-\frac{2ac}{b}\right)}{2a} \approx -\frac{c}{b}$$

Nous avons donc conclu qu'il est possible d'approximer l'une des solutions (celle qui était égale à 0) par -(c/b) . Après avoir implémenté cela dans les cas où 4ac/b2 < 10-20 , nous avons obtenu des solutions plus cohérentes. Ces solutions étaient correctes car lorsque nous avons testé avec des calculatrices en ligne, en remplaçant par ces dernières dans les équations nous obtenons des nombres autour de 10-16 au lieu de 0 et qui étaient donc inférieure à la tolérance. Néanmoins l'oracle a continué à donner un résultat négatif (False). Cela était cette fois ci dû au fait que dans la fonction de test, nous calculons |(ax2+bx+c)/c| et nous comparons cela à la tolérance or en général lorsque b2>> 4ac , l'une des deux solutions est très grande en valeur absolue et l'autre est très très faible. Cela fait en général que ax2 = -bx (ax2 et -bx sont en valeur absolues plus grands que 10^40 et égales d'après les calculs de la machine. On a donc |(ax2+bx+c)/c| = 1.0 en général, ce qui fausse le calcul.

La solution est donc de changer de fonction de tests mais nous n'avons pas trouvé de fonction de tests qui gère ces cas-là.

III. DIFFÉRENCE FINIES EN DIMENSION 1

Problème : Résolution de l'équation $-u'' = f$ sur $[0,1]$avec $f\ de\ Classe\ C^2\ sur\ ]0{,}1[, f(0) = a, f(1) = b$ et test de ses solutions.

Fonction à Tester : la fonction resolve (String fonction, double a, double b, int n) qui permet de déterminer les solutions de l'équation différentielle : $-u'' = f$

Donnée d'entrée : Les données d'entrée sont "fonction" qui représente la fonction f (écrite dans un formalisme que nous décodons afin d'en extraire les valeurs en différents points, $a$et b qui valent respectivement $f(a)\ et\ f(b)$ et n qui représente la taille du maillage (et implique qu'on a $n+1$points)

- **Nombre de cas testés :** Dans le cadre de la résolution des équations différentielles par différence finie, l'on distingue **deux** principaux cas de test : les fonctions polynômes de degré inférieur ou égal à 2 et toutes les autres fonctions de classe C^2(en dehors des polynômes de degré inférieur ou égal à 2. Afin de s'assurer de l'efficacité de la fonction Solver, nous avons généré 28 fonctions à tester (15 et 12 respectivement). Pour introduire ces fonctions dans le programme, nous avons utilisé une méthode (Java) permettant de

générer des fonctions en lisant une chaîne de caractère dont le formatage a été au préalable définit.

Méthode de Résolution : la méthode que nous utiliserons pour résoudre le système matriciel ici est la méthode de Gauss-Seidel qui est une méthode itérative.

Fonction de Tests : nous avons utilisé deux fonctions de tests :

- ➔ l'erreur relative basée sur la norme Infinie: elle est utilisée faisant la différence relative entre la norme infinie de la solution réelle et de la solution. Elle a été utilisée pour tester les cas où f est une fonction polynôme de degré inférieur ou égal à 2. Un cas de test est réussi pour un polynôme de degré inférieur ou égal à 2 lorsque l'erreur relative pour ce cas est inférieure à 10^{-10}.
- ➔ l'erreur absolue basée sur la norme infinie: elle est utilisée uniquement pour tester le résultat de la fonction avec comme donnée d'entrée la fonction nulle.
- ➔ convMail : Cette fonction de test est utilisée pour les cas de fonctions de classe C^2 n'étant pas des polynômes de degré inférieurs ou égal à 2. C'est une fonction permettant de calculer l'ordre de convergence de la méthode pour une fonction (à partir de la formule : $\frac{log(err2) - log(err1)}{log(h2) - log(h1)}$). Le test est réussi par cette fonction si l'ordre de convergence est supérieur ou égal à $2 - 10^{-3}$.

Tolérance : 10^{-10}

Résultat Attendu : Le résultat attendu est une matrice U de taille $n - 1$ tel que $\forall\, i \in 0 \ldots (n-2), U[i] = u((i+1)/n)$. Mais afin d'automatiser sa construction, nous avons passé dans le fichier Excel, l'expression écrite suivant notre standard de la fonction u que nous décodons et remplissons ainsi U.

Oracle : L'oracle a été donné lors de la présentation des différentes méthodes de test.

RÉSULTATS

- **1er cas de test :** Nous avons effectué des tests sur 15 fonctions polynômes de degré inférieur ou égal à 2. Ces fonctions ont été générées par combinaisons linéaires des vecteurs de base de l'espace des polynômes de degré inférieur ou égal à 2. Nous avons aussi testé la fonction nulle.

Les tests ont été effectué avec $n = 100$.

Résultat : 15 tests réussis sur 15.

Ci-dessous le tableau présentant les données de test correspondant au premier cas

OMI = Oracle Méthode Itérative

f	a	b	Tolérance	Résultat attendu	Fonction de test	O.M.I
DRnulle;	00.00E00	00.00E00	1.0E-10	nulle	erreurRel	true
0.0000001:DRcte;	10.00E-08	10.00E-08	1.0E-10	0.0000001:cte	erreurRel	true
-0.0000001:DRcte;	-10.00E-08	-10.00E-08	1.0E-10	-0.0000001:cte	erreurRel	true
1:DRcte;	10.00E-01	10.00E-01	1.0E-10	1:cte	erreurRel	true
1.000000001:DRcte;	10.00E-01	10.00E-01	1.0E-10	1.000000001:cte	erreurRel	true
-1.000000001:DRcte;	-10.00E-01	-10.00E-01	1.0E-10	-1.000000001:cte	erreurRel	true
8:DRcte;	80.00E-01	80.00E-01	1.0E-10	8:cte	erreurRel	true
-1:DRcte;	-10.00E-01	-10.00E-01	1.0E-10	-1:cte	erreurRel	true
19:DRsimplex;	00.00E00	19.00E00	1.0E-10	19:simplex	erreurRel	true
-3:DRsimplex;	00.00E00	-30.00E-01	1.0E-10	-3:simplex	erreurRel	true
-2:DRcarrex;	00.00E00	-20.00E-01	1.0E-10	-2:carrex	erreurRel	true
6:DRcarrex;	00.00E00	60.00E-01	1.0E-10	6:carrex	erreurRel	true
7:DRcte;12:DRnulle;	70.00E-01	70.00E-01	1.0E-10	7:cte;12:nulle	erreurRel	true
8:DRcarrex;-2:DRcarrex;	00.00E00	60.00E-01	1.0E-10	8:carrex;-2:carrex	erreurRel	true

- **2ème cas de test :** Afin de tester le deuxième cas, nous avons réalisé des combinaisons linéaires de fonctions polynôme de degré supérieur à 2, Cosinus, Sinus, Tangente, Exponentielle, Logarithme (ln (1+x)) et réalisé aussi des produits entre ces dernières. Le calcul de l'ordre de convergence s'est fait avec $n_1 = 100\ et\ n_2 = 300$.

  Résultats : 11 tests réussis sur 12.

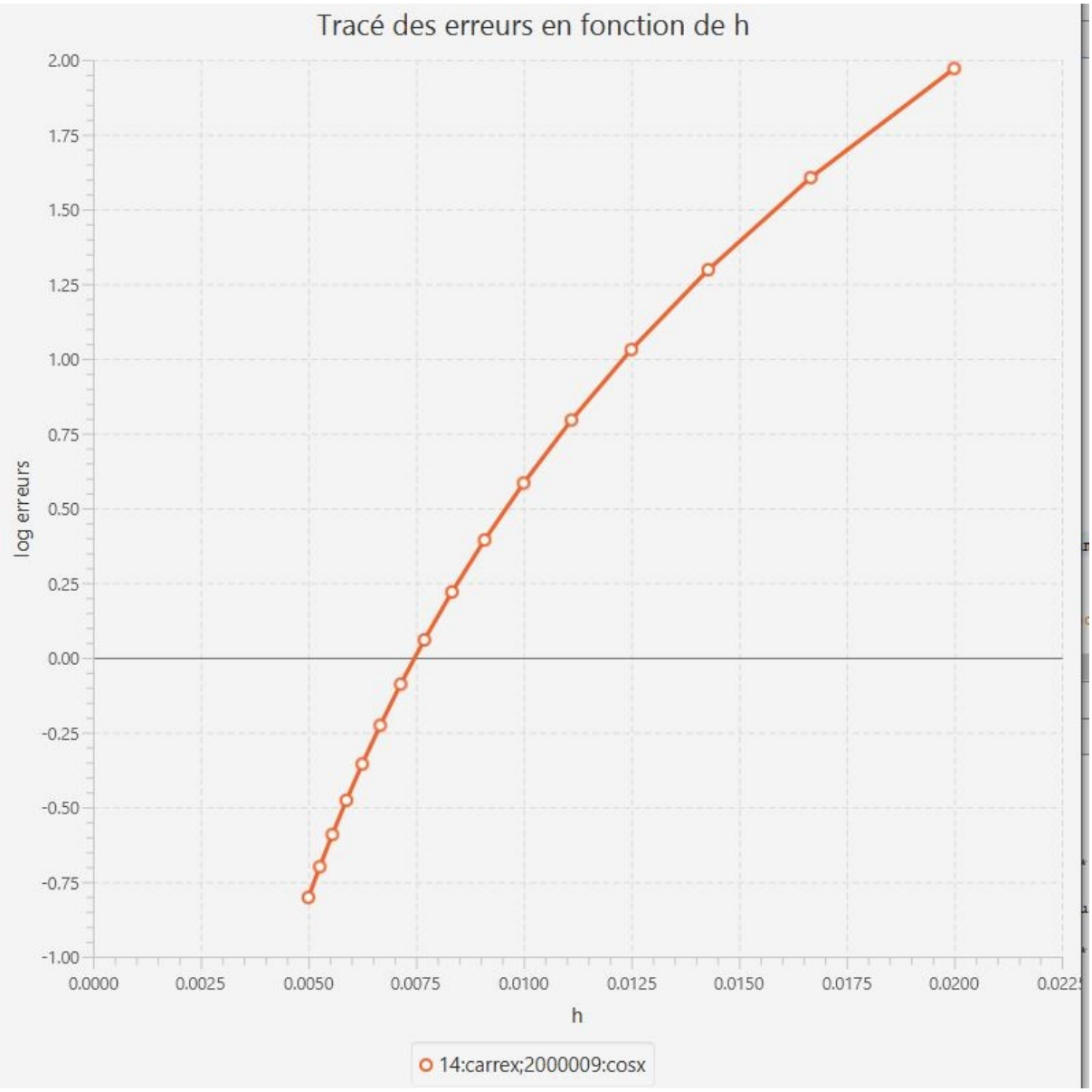
  Le test qui n'a pas réussi correspond à la fonction $3\ ln(1+x)\ +\ 4\ exp(x)$.

  Ci-dessous le tableau présentant les différentes données de test.

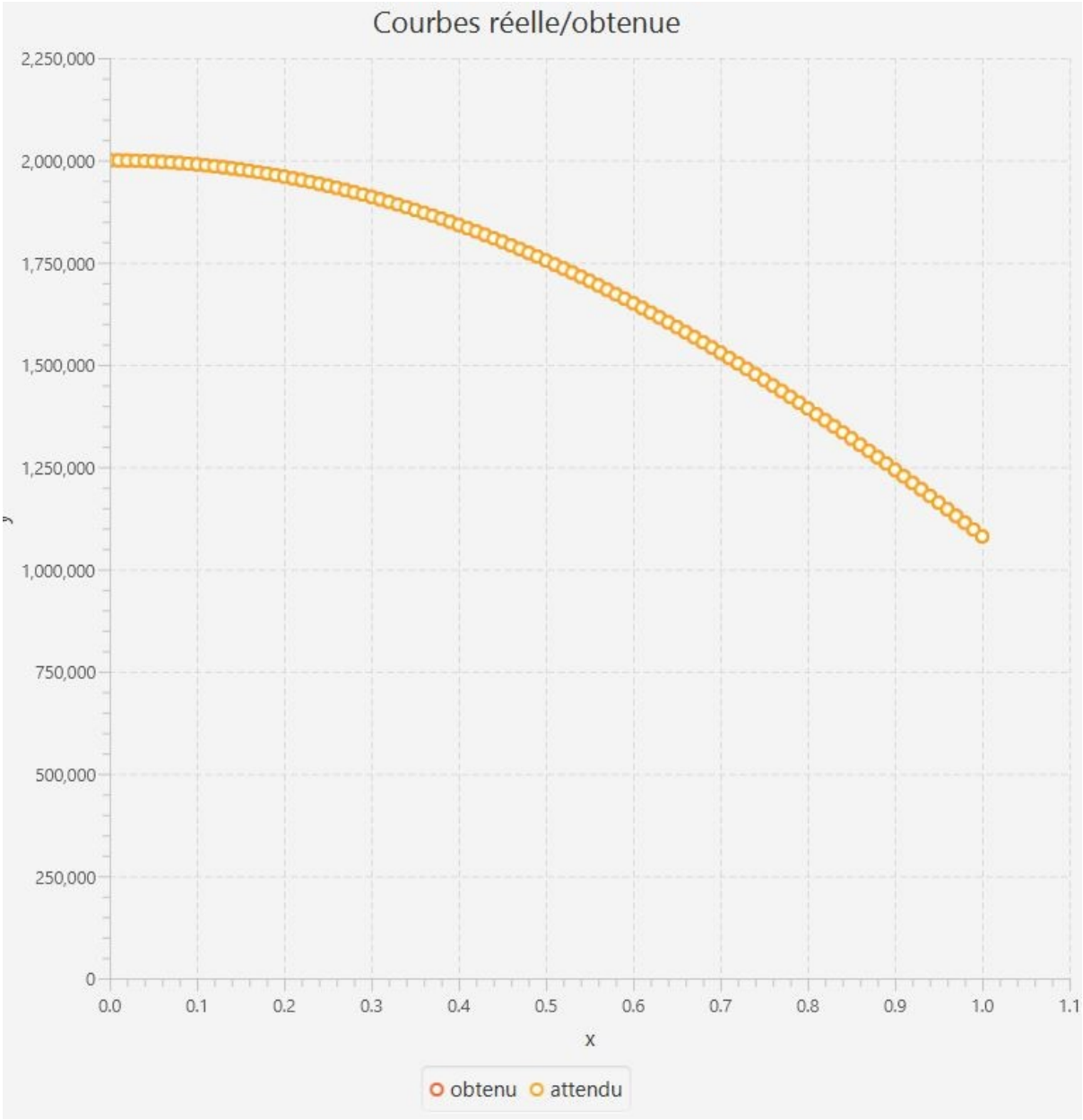
2:DRp5x;-3:DRcubex;DRsimplex;	00.00E00	00.00E00	0.001	2:p5x;-3:cubex;simplex	convMail	true
DRcosx;	10.00E-01	54.03E-02	0.001	cosx	convMail	true
6:DRexp;	60.00E-01	16.31E00	0.001	6:exp	convMail	true
4:DRsinx;4:DRcte;	40.00E-01	73.66E-01	0.001	4:sinx;4:cte	convMail	true
-2:DRcte;4:DRexp;	20.00E-01	88.73E-01	0.001	-2:cte;4:exp	convMail	true
9:DRcarrex;22:DRexp;	22.00E00	68.80E00	0.001	9:carrex;22:exp	convMail	true
14:DRcarrex;2000009:DRcosx;	20.00E05	10.81E05	0.001	14:carrex;2000009:cosx	convMail	true
2:DRsinx;-4:DRcosx;	-40.00E-01	-47.83E-02	0.001	2:sinx;-4:cosx	convMail	true
3:DRlnUn;4:DRexp;	40.00E-01	12.95E00	0.001	3:lnUn;4:exp	convMail	false
3:DRsinx;-3:DRexp;	-30.00E-01	-56.30E-01	0.001	3:sinx;-3:exp	convMail	true
3:DRcubex;4:DRsinx;	00.00E00	63.66E-01	0.001	3:cubex;4:sinx	convMail	true
-2:DRcosx;-9:DRexp;	-11.00E00	-25.55E00	0.001	-2:cosx;-9:exp	convMail	true

Cas de la fonction $14x^2 + 2000009\ cos\ x$

Ordre de convergence : Le calcul de l'ordre de convergence (avec $n_1 = 50\ et\ n_2 = 200$) pour la fonction $14x^2 + 2000009\ cos\ x$ a donné 2.00001.



Cette courbe a été obtenue après avoir fait varier n de 50 à 200 en pas de 10.



Cette courbe représente le schéma du résultat attendu compare au résultat réel pour n = 100. Sur cette dernière, on peut constater que les deux fonctions sont indistinguables.

IV. DIFFÉRENCES FINIES DIMENSION 2

Problème : Résolution de l'équation $-u'' = f$ sur $[0,1] * [0,1]$avec $f\ de\ Classe\ C^2\ sur\ ]0,1[*]0,1[, f(0,) = a, f(1) = b$ et test de ses solutions.

Fonction à Tester : la fonction solveSystem (matrice csc, maillage b) ou csc est la matrice du système au format ccs et b le maillage au sens des différences finies relatif au problème.

Donnée d'entrée : Les données d'entrée sont "fonction" qui représente la fonction f (écrite dans un formalisme que nous décodons afin d'en extraire les valeurs en différents points, $a$et b qui valent respectivement $f(a)\ et\ f(b)$ et $n\ et\ m$ qui représente la taille du maillage (et implique qu'on a $(n+1) * (m+1)$points)

- **Nombre de cas testés :** Dans le cadre de la résolution des équations différentielles par différence finie, l'on distingue **deux** principaux cas de test : les fonctions polynômes de degré inférieur ou égal à 2 et toutes les autres fonctions de classe C^2(en dehors des polynômes de degré inférieur ou égal à 2. Afin de s'assurer de l'efficacité de la fonction Solver, nous avons généré 27 fonctions à tester (10 et 17 respectivement). Pour introduire ces fonctions dans le programme, nous avons utilisé les fonctions lambda en python, fonction qui permettent de créer des fonctions au sens mathématique en spécifiant les variables et l'expression de ladite fonction. Exemple : la fonction calculant le carré d'un nombre est : $lambda\ x: x ** 2$.

Méthode de Résolution : la méthode que nous utiliserons pour résoudre le système matriciel ici est la méthode de Gauss-Seidel qui est une méthode itérative.

Fonction de Tests : nous avons utilisé deux fonctions de tests :

➔ L'erreur relative basée sur la norme Infinie : elle est utilisée faisant la différence relative entre la norme infinie de la solution réelle et de la solution. Elle a été utilisée pour tester les cas où f est une fonction polynôme de degré inférieur ou égal à 2. Un cas de test est réussi pour

un polynôme de degré inférieur ou égal à 2 lorsque l'erreur relative pour ce cas est inférieure à 10^{-8}.

➔ ConvMail : Cette fonction de test est utilisée pour les cas de fonctions de classe C^2 n'étant pas des polynômes de degré inférieurs ou égal à 2. C'est une fonction permettant de calculer l'ordre de convergence de la méthode pour une fonction (à partir de la formule : $\frac{log(err2) - log(err1)}{log(h2) - log(h1)}$). Le test est réussi par cette fonction si l'ordre de convergence est supérieur ou égal à $2 - 10^{-3}$.

Tolérance : 10^{-8}

Résultat Attendu : Le résultat attendu est une matrice U de taille $n - 1$ tel que $\forall\, i \in 0 \ldots (n-2), U[i] = u((i+1)/n)$. Mais afin d'automatiser sa construction, nous avons passé dans le fichier Excel, l'expression écrite suivant notre standard de la fonction u que nous décodons et remplissons ainsi U.

Oracle : L'oracle a été donné lors de la présentation des différentes méthodes de test.

RÉSULTATS

- **1er cas de test :** Nous avons d'abord testé le cas nul. Il a été testé avec un maillage pour n=100 et m=200

  Résultat :

  Test Réussi : 1/1

  Cas de test fonction nulle

  ```
  vit1 = testFunctionNulle(lambda x,y : 0,lambda x,y:0,100,200,1E-8)
  vit1
  ```

  ```
  True
  ```

- **2eme cas de test:** Nous avons effectué des tests sur 12 fonctions polynômes de degré inférieur ou égal à 2. Ces fonctions ont été générée à partir de

combinaison de polynôme de degré <= 2 et stockée dans un tableau. Les tests ont été effectué avec $n = 100\ et\ m = 200$.

Résultat :

Test Réussi : 12/12

Cas de test fonction Polynome de degre < 2

```
DE   = [{'f': lambda x,y : -24, 'u': lambda x,y: 12*x**2 , 'n': 100, 'm':200 , 'tol': 1E-8 },
        {'f': lambda x,y : 12*y, 'u': lambda x,y: -6*(x**2)*y , 'n': 100, 'm':200 , 'tol': 1E-8 },
        {'f': lambda x,y: 0, 'u': lambda x,y: 3*x, 'n': 100 , 'm':200 , 'tol': 1E-8},
        {'f': lambda x,y : -24, 'u': lambda x,y: 12*y**2 , 'n': 100, 'm':200 , 'tol': 1E-8 },
        {'f': lambda x,y : 12*x, 'u': lambda x,y: -6*(y**2)*x , 'n': 100, 'm':200 , 'tol': 1E-8},
        {'f': lambda x,y: 0, 'u': lambda x,y: 3*y, 'n': 100 , 'm':200 , 'tol': 1E-8 },
        {'f': lambda x,y : -24, 'u': lambda x,y: 12*x**2 , 'n': 200, 'm':100 , 'tol': 1E-8 },
        {'f': lambda x,y : 12*y, 'u': lambda x,y: -6*(x**2)*y , 'n': 200, 'm':100 , 'tol': 1E-8 },
        {'f': lambda x,y: 0, 'u': lambda x,y: 3*x, 'n': 200 , 'm':100 , 'tol': 1E-8},
        {'f': lambda x,y : -24, 'u': lambda x,y: 12*y**2 , 'n': 200, 'm':100 , 'tol': 1E-8 },
        {'f': lambda x,y : 12*x, 'u': lambda x,y: -6*(y**2)*x , 'n': 200, 'm':100 , 'tol': 1E-8},
        {'f': lambda x,y: 0, 'u': lambda x,y: 3*y, 'n': 200 , 'm':100 , 'tol': 1E-8 },

      ]

boolResult = []
print(len(DE))
for i in range(0, len(DE)):
    boolResult.append(testFunctionSimple(DE[i]['f'], DE[i]['u'], DE[i]['n'],DE[i]['m'],DE[i]['tol'] ))
print(len(boolResult))
print(boolResult)
```

```
12
12
[True, True, True, True, True, True, True, True, True, True, True, True]
```

- **3eme cas de test :** Nous avons effectué 16 tests de fonction complexe tels que exp, sin, cos, ln, et les fonctions polynôme de degré > 2. Ces fonctions ont été générée et stockée dans un tableau. Les tests ont été effectué avec $n = 100\ et\ m = 200$.

Résultat :

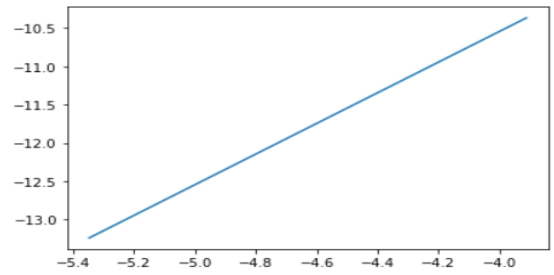
Test Réussi : 15/16

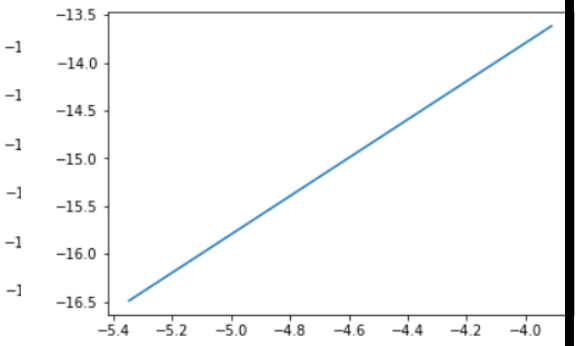
Cas de test fonction Complexes

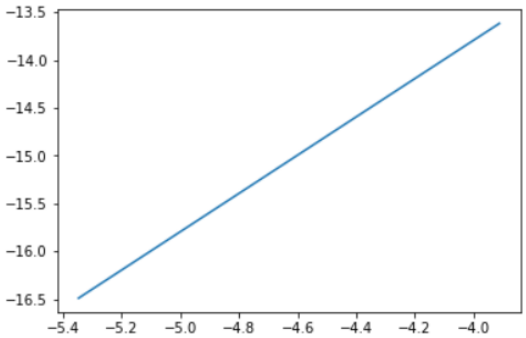
```
DE2 = [{'f': lambda x,y : -12*(x**2)*y, 'u': lambda x,y : (x**4)*y , 'n': 100, 'm':200, 'tol': 1E-8},
       {'f': lambda x,y : -12*(y**2)*x, 'u': lambda x,y : (y**4)*x , 'n': 100, 'm':200, 'tol': 1E-8},
       {'f': lambda x,y : -12*(x**2)*(y**4)-12*(x**4)*(y**2), 'u': lambda x,y : (x**4)*(y**4) , 'n': 100, 'm':200, 'tol': 1E-8
       {'f': lambda x,y: -2/((x+1)**3), 'u': lambda x,y: 1/(x+1) , 'n': 100, 'm':200 , 'tol': 1E-8},
       {'f': lambda x,y: sin(x), 'u': lambda x,y: sin(x) , 'n': 100, 'm':200 , 'tol': 1E-8},
       {'f': lambda x,y: -2/((y+1)**3), 'u': lambda x,y: 1/(y+1) , 'n': 100, 'm':200 , 'tol': 1E-8 },
       {'f': lambda x,y: sin(y), 'u': lambda x,y: sin(y) , 'n': 100, 'm':200 , 'tol': 1E-8 },
       {'f': lambda x,y: sin(x)+cos(y), 'u': lambda x,y: sin(x)+cos(y) , 'n': 100, 'm':200,  'tol': 1E-8},
       {'f': lambda x,y : -12*(x**2)*y, 'u': lambda x,y : (x**4)*y , 'n': 200, 'm':100, 'tol': 1E-8},
       {'f': lambda x,y: -2/((x+1)**3), 'u': lambda x,y: 1/(x+1) , 'n': 200, 'm':100 , 'tol': 1E-8},
       {'f': lambda x,y: sin(x), 'u': lambda x,y: sin(x) , 'n': 200, 'm':100 , 'tol': 1E-8},
       {'f': lambda x,y: -2/((y+1)**3), 'u': lambda x,y: 1/(y+1) , 'n': 200, 'm':100 , 'tol': 1E-8 },
       {'f': lambda x,y: sin(y), 'u': lambda x,y: sin(y) , 'n': 200, 'm':100 , 'tol': 1E-8 },
       {'f': lambda x,y: sin(x)+cos(y), 'u': lambda x,y: sin(x)+cos(y) , 'n': 200, 'm':100,  'tol': 1E-8},
       {'f': lambda x,y: 1/(x+1.1)**2, 'u': lambda x,y: log(x+1.1) , 'n': 200, 'm':100 , 'tol': 1E-8 },
       {'f': lambda x,y: exp(x)+exp(y), 'u': lambda x,y: exp(x)+exp(y) , 'n': 100, 'm':100 , 'tol': 1E-8 },

      ]

vit = []
print(len(DE2))
for i in range(0, len(DE2)):
    vit.append(testFunctionComplex(DE2[i]['f'], DE2[i]['u'], 50,200 )[0])
print(vit)
```

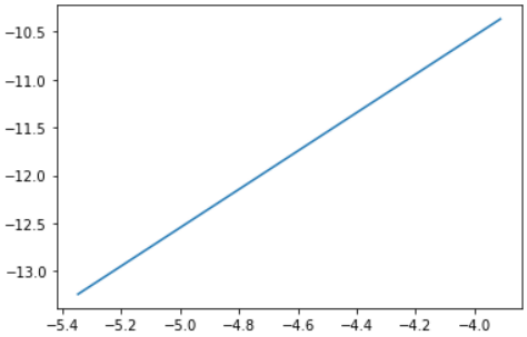
```
16
[True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, False]
```

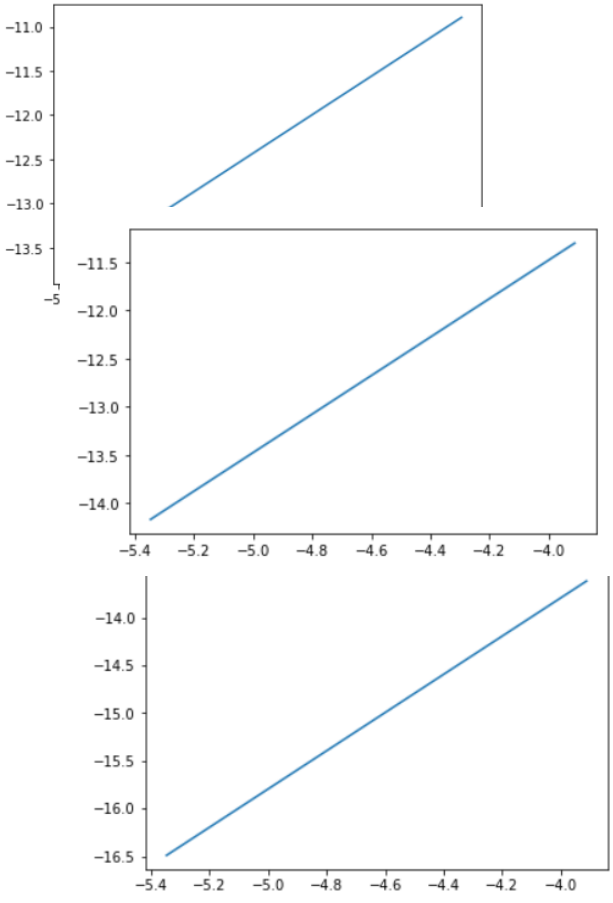
Graphe de convergence pour les fonctions complexe

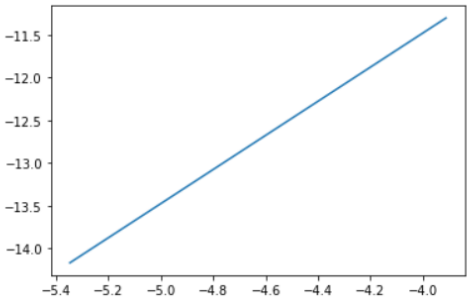


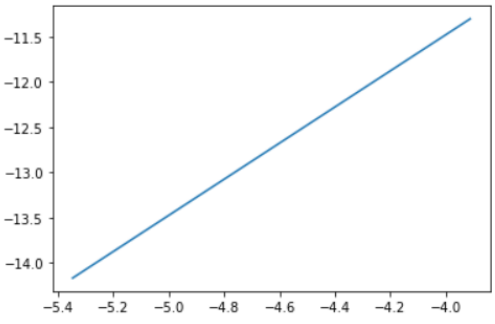


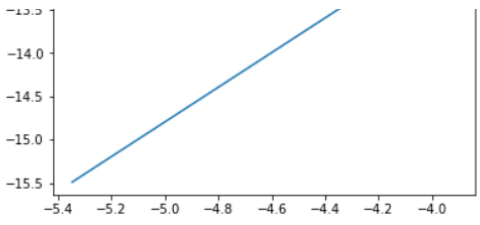


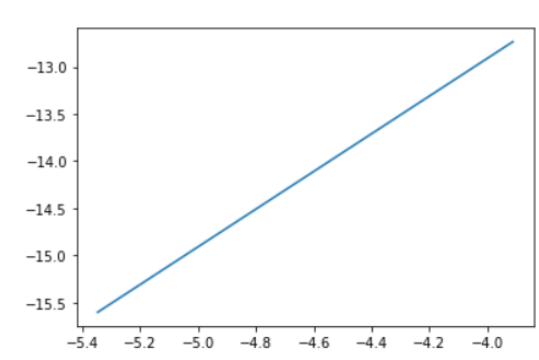


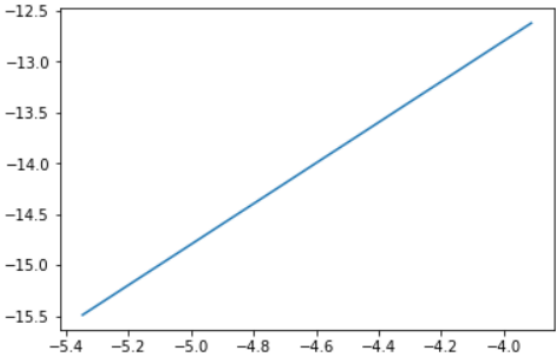


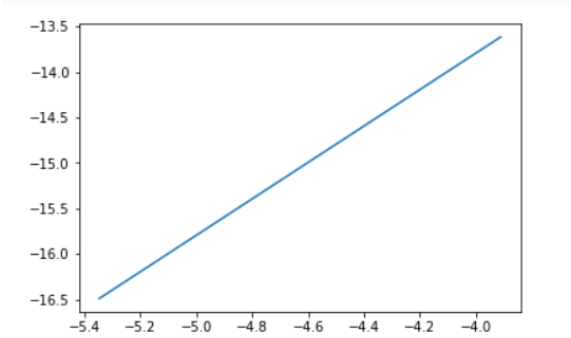


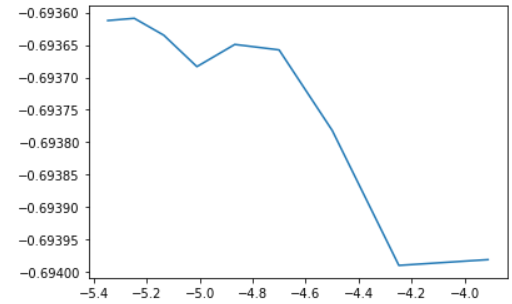












V. VOLUMES FINIS DIMENSION 1

Problème : Résolution de l'équation $-(\lambda u')'(x) + a.u'(x) + b.u(x) = f(x), x \in [0,1], u(0) = c, u(1) = d, b > 0,$

$$\lambda \in L^{\infty}([0,1]), f \in L^{2}([0,1])$$

Fonction à Tester : la fonction resolve (LambdaFunction fonction, LambdaFunction u, double a, double b, int n, LambdaFunction lambda) qui permet de déterminer les solutions de l'équation différentielle présentée plus haut (LambdaFunction en Python est un type qui permet de mettre directement l'expression d'une fonction)

Donnée d'entrée : Les données d'entrée sont "fonction" qui représente la fonction f, "u" qui représente la fonction u, a, b qui représentent les coefficients a et b de l'équation, lambda qui représente la fonction λ de l'équation et n qui représente la taille du maillage.

Nombre de cas testés : Dans le cadre de la résolution des équations différentielles par différence finie, l'on distingue **trois** principaux cas de test : la fonction nulle, les

fonctions polynômes de degré inférieur ou égal à 2 et toutes les autres fonctions de classe C^2(en dehors des polynômes de degré inférieur ou égal à 2.

Méthode de Résolution : la méthode que nous utiliserons pour résoudre le système matriciel ici est la méthode de Gauss-Seidel Parallèle qui est une méthode itérative.

Fonction de Tests : nous avons utilisé trois fonctions de tests :

- ➔ L'erreur relative basée sur la norme Infinie : elle est utilisée faisant la différence relative entre la norme infinie de la solution réelle et de la solution. Elle a été utilisée pour tester les cas où f est une fonction polynôme de degré inférieur ou égal à 2 et différent de la fonction nulle. Un cas de test est réussi pour un polynôme de degré inférieur ou égal à 2 lorsque l'erreur relative pour ce cas est inférieure à 10^{-10}.
- ➔ L'erreur absolue pour la fonction nulle
- ➔ ConvMail : Cette fonction de test est utilisée pour les cas de fonctions de classe C^2 n'étant pas des polynômes de degré inférieurs ou égal à 2. C'est une fonction permettant de calculer l'ordre de convergence de la méthode pour une fonction (à partir de la formule : $\frac{log(err2) - log(err1)}{log(h2) - log(h1)}$). Le test est réussi par cette fonction si l'ordre de convergence est supérieur ou égal à $2 - 10^{-3}$.

Tolérance : 10^{-10}

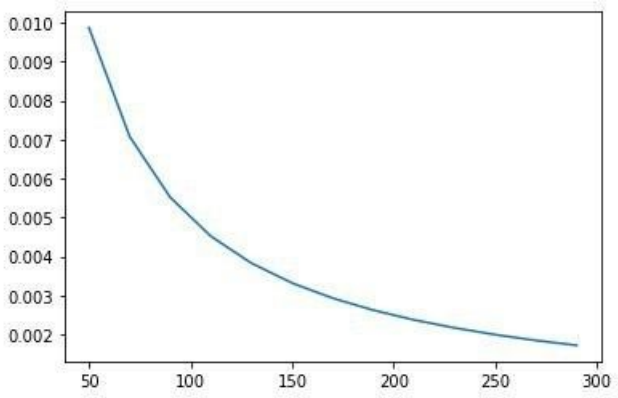
Résultat Attendu : Le résultat attendu est une matrice U de taille $n - 1$ tel que $\forall\, i \in 0 \ldots (n-2), U[i] = u((i+1)/n)$.

Oracle : L'oracle a été donné lors de la présentation des différentes méthodes de test.

Résultats :

La fonction testée n'a pas fourni de bons résultats. En effet, l'erreur obtenue sur les différents était de 10^-2 au lieu de 10^-8

Nous travaillons pour l'améliorer



Courbe Erreur 1: Courbe presentant l'evolution des erreurs avec n

VI. VOLUMES FINIS DIMENSION 2

Probleme: Resolution de l'equation $-(\lambda u')'(x) + a.u'(x) + b.u(x) = f(x), x \in [0,1], u(0) = c, u(1) = d, b > 0,$

$$\lambda \in L^{\infty}([0,1]), f \in L^{2}([0,1])$$

Fonction à Tester : la fonction resolve(LambdaFunction fonction, LambdaFunction u, double a, double b, int n, LambdaFunction lambda) qui permet de déterminer les solutions de l'équation différentielle présentée plus haut (LambdaFunction en Python est un type qui permet de mettre directement l'expression d'une fonction)

Donnée d'entrée : Les données d'entrée sont "fonction" qui représente la fonction f, "u" qui représente la fonction u, a, b qui représentent les coefficients a et b de l'équation, lambda qui représente la fonction λ de l'équation et n qui représente la taille du maillage.

Nombre de cas testés : Dans le cadre de la résolution des équations différentielles par différence finie, l'on distingue **trois** principaux cas de test : la fonction nulle, les fonctions polynômes de degré inférieur ou égal à 2 et toutes les autres fonctions de classe C^2(en dehors des polynômes de degré inférieur ou égal à 2.

Méthode de Résolution : la méthode que nous utiliserons pour résoudre le système matriciel ici est la méthode de Gauss-Seidel Parallèle qui est une méthode itérative.

Fonction de Tests : nous avons utilisé trois fonctions de tests:

- ➔ l'erreur relative basée sur la norme Infinie: elle est utilisée faisant la différence relative entre la norme infinie de la solution réelle et de la solution. Elle a été utilisée pour tester les cas où f est une fonction polynôme de degré inférieur ou égal à 2 et différent de la fonction nulle. Un cas de test est réussi pour un polynôme de degré inférieur ou égal à 2 lorsque l'erreur relative pour ce cas est inférieure à 10^{-10}.
- ➔ l'erreur absolue pour la fonction nulle
- ➔ convMail: Cette fonction de test est utilisée pour les cas de fonctions de classe C^2 n'étant pas des polynômes de degré inférieurs ou égal à 2. C'est une fonction permettant de calculer l'ordre de convergence de la méthode pour une fonction (à partir de la formule : $\frac{log(err2) - log(err1)}{log(h2) - log(h1)}$). Le test est réussi par cette fonction si l'ordre de convergence est supérieur ou égal à $2 - 10^{-3}$.

Tolérance: 10^{-10}

Résultat Attendu: Le résultat attendu est une matrice U de taille $n - 1$ tel que $\forall\ i\ \in 0 \ldots (n - 2), U[i] = u((i + 1)/n)$.

Oracle : L'oracle a été donné lors de la présentation des différentes méthodes de test.

Résultats

I. Volumes finis dimension 2

Problème : Résolution de l'équation $-div(gradU)$ sur $\Omega = [0,1] * [0,1]$avec $U = g\ sur\partial\Omega$ et$f$ continue et dérivable sur Ω test de ses solutions.

Structure du code:
Nous avons notre classe SolveurVF2D : qui définit les toutes les méthodes qui seront utilisés

Fonction à Tester : la fonction draw_courbe(fonction , n,m) où n, et m définissent la taille du maillage à utiliser pour résoudre le problème.

Donnée d'entrée :

- Les données d'entrée sont "fonction" qui représente la fonction f (écrite dans un formalisme que nous décodons afin d'en extraire les valeurs en différents points, $a$et b qui valent respectivement $f(a)\ et\ f(b)$
- $n\ et\ m$ qui représente la taille du maillage (et implique qu'on a $(n+1)*(m+1)$points)
- -laplacien de U qui représente la valeur du calcul du laplacien de u
- Nombre de cas testés : Dans le cadre de la résolution des équations en volume finie 2D, l'on distingue trois principaux cas de test : les fonctions polynômes et la fonction nul et les fonctions non polynomiale- . Afin de s'assurer de l'efficacité de la fonction draw_courbe, nous avons testé quelques fonctions dans chacun de ses cas de test pour apprécier les résultats. Pour introduire ces fonctions dans le programme, nous avons utilisé les fonctions lambda en python, fonction qui permettent de créer des fonctions au sens mathématique en spécifiant les variables et l'expression de ladite fonction. exemple: la fonction calculant le carré d'un nombre est: $lambda\ x,y: x**2+y**2$.

Méthode de Résolution : la méthode que nous utiliserons pour résoudre le système matriciel ici est la méthode de Gauss-Seidel qui est une méthode itérative.

Fonction de Tests : nous avons utilisé deux fonctions de tests :

➔ l'erreur relative basée sur la norme Infinie: elle est utilisée faisant la différence relative entre la norme infinie de la solution réelle et de la solution. Elle a été utilisée pour tester les cas où f est une fonction polynôme de degré inférieur ou égal à 2. Un cas de test est réussi pour un polynôme de degré inférieur ou égal à 2 lorsque l'erreur relative pour ce cas est inférieure à 10^{-8}.

➔ draw_error_courbe : Cette fonction de test est utilisée pour pour calculer les erreurs et afficher la courbe correspondante en fonction de $1/n$

Tolérance: 10^{-8}

Résultat Attendu: Le résultat attendu est une matrice U de taille $n-1$ tel que $\forall\, i \in 0\ldots(n-2), U[i] = u((i+1)/n)$. Mais afin d'automatiser sa construction, nous avons passé dans le fichier Excel, l'expression écrite suivant notre standard de la fonction u que nous décodons et remplissons ainsi U.

Oracle : L'oracle a été donné lors de la présentation des différentes méthodes de test.

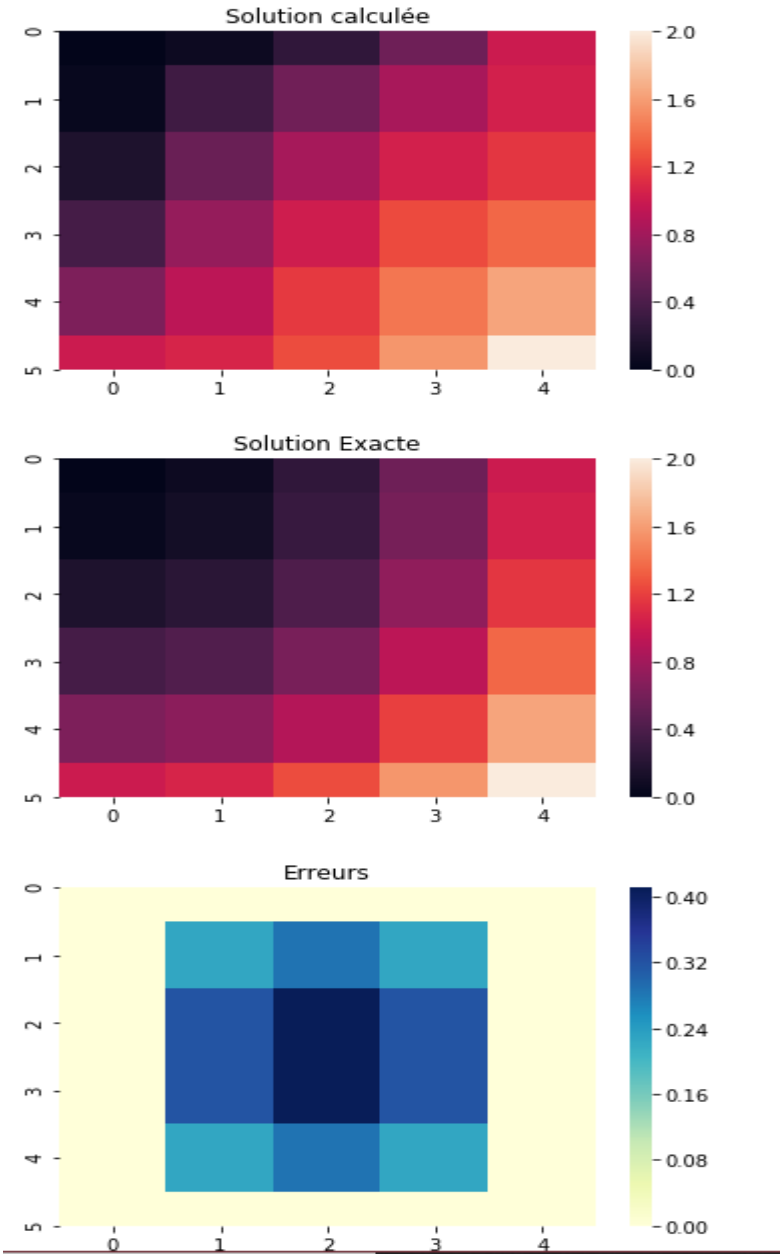
RÉSULTATS

Donnée de test pour un scénario d'une fonction polynomiale

SolveurVF2D().draw_courbe(lambda x,y:x2+y**2 , lambda x,y:-6*x-6*y, n,m)**

```
#Donnée de test pour un scénario d'une fonction polynômile
SolveurVF2D().draw_courbe(lambda x,y:x**2+y**2 , lambda x,y:-6*x-6*y, n,m)
```

```
[[ 4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.    0.  ]
 [-0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.    0.    0.    0.  ]
 [-1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.  ]
 [ 0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.  ]
 [ 0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.  ]
 [ 0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.  ]
 [ 0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.  ]
 [ 0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25]
 [ 0.    0.    0.    0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8 ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1 ]]
```

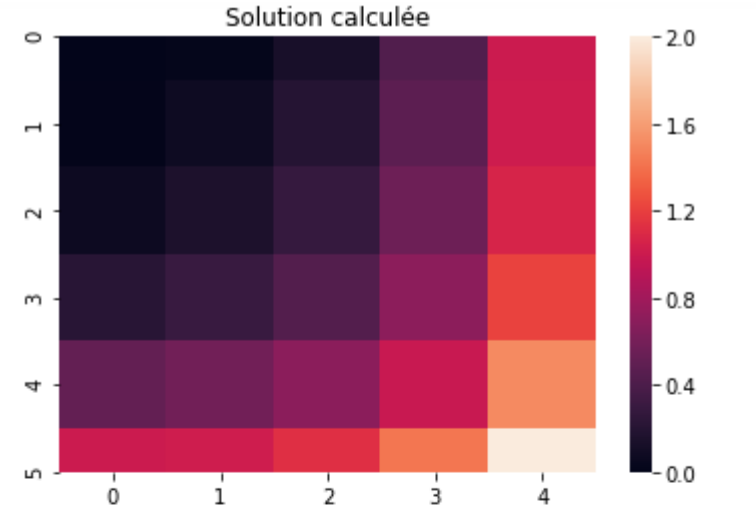


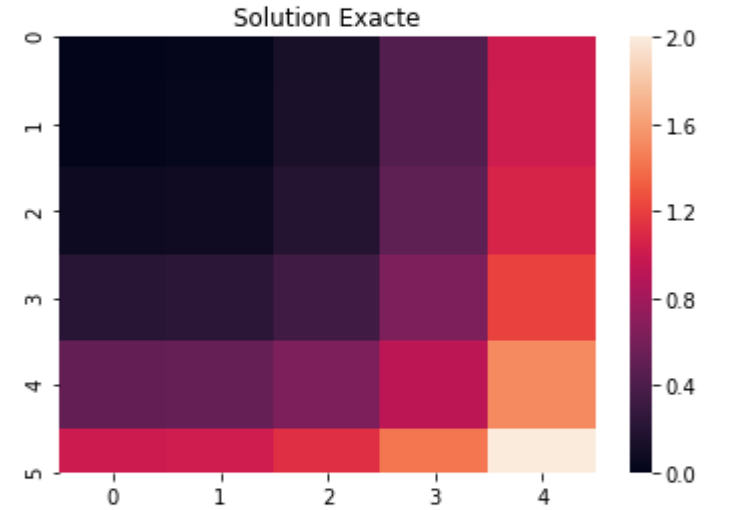
données pour le scénario d'une fonction polynômiale

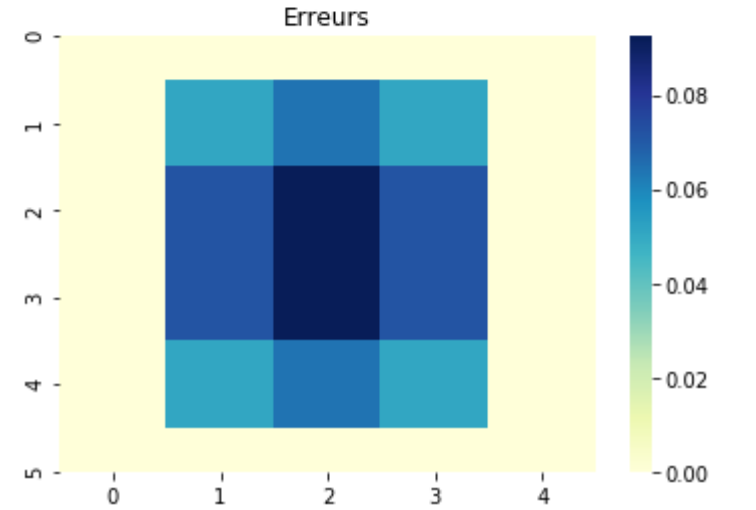
SolveurVF2D().draw_courbe(lambda x,y:x**3+y**3 , lambda x,y:-6*x-6*y, n,m)

```
#données pour le scénario d'une fonction polynômiale
SolveurVF2D().draw_courbe(lambda x,y:x**3+y**3 , lambda x,y:-6*x-6*y, n,m)
```

```
[[ 4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.    0.  ]
 [-0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.    0.    0.    0.  ]
 [-1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.  ]
 [ 0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.  ]
 [ 0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.  ]
 [ 0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.  ]
 [ 0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.  ]
 [ 0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25]
 [ 0.    0.    0.    0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8 ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1 ]]
```





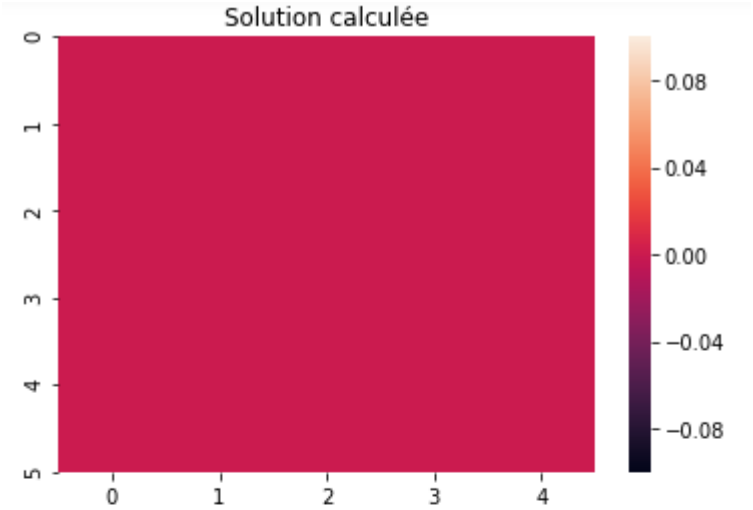


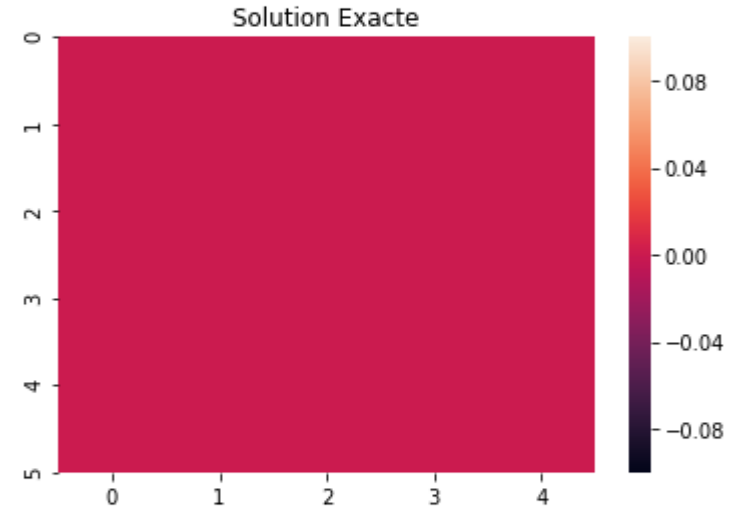
données pour le scénario d'une fonction nul

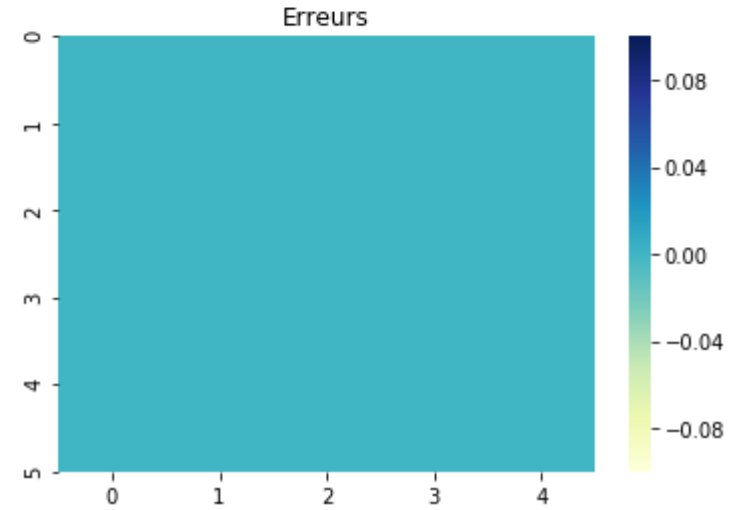
SolveurVF2D().draw_courbe(lambda x,y:0, lambda x,y:0, n,m)

```
#données pour le scénario d'une fonction nul
SolveurVF2D().draw_courbe(lambda x,y:0, lambda x,y:0, n,m)
```

```
[[ 4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.    0.  ]
 [-0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.    0.    0.    0.  ]
 [-1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.  ]
 [ 0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.  ]
 [ 0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.  ]
 [ 0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.  ]
 [ 0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.  ]
 [ 0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25]
 [ 0.    0.    0.    0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8 ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1 ]]
```





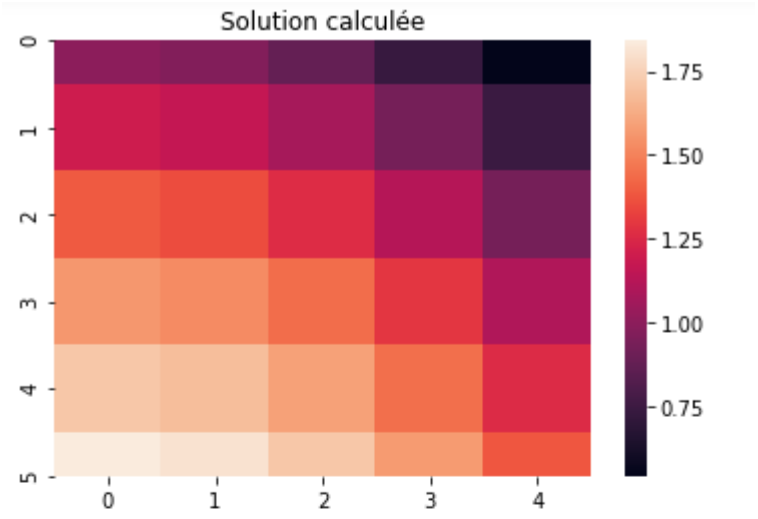


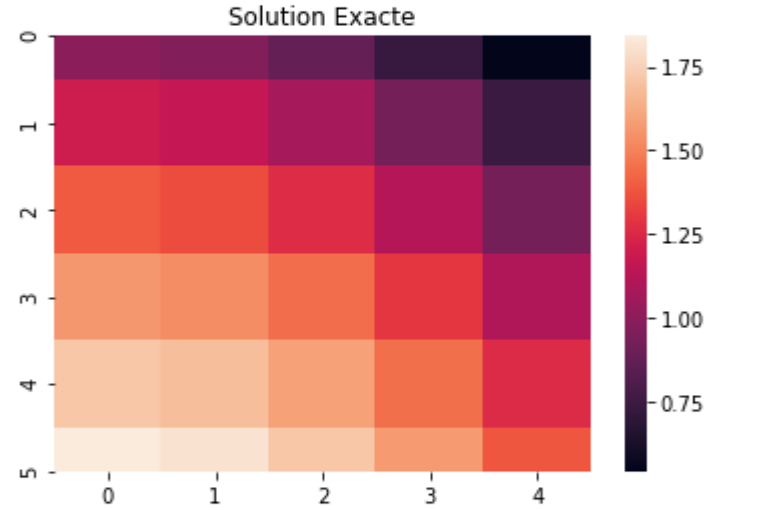
données pour le scénario d'une fonction non polynômiale

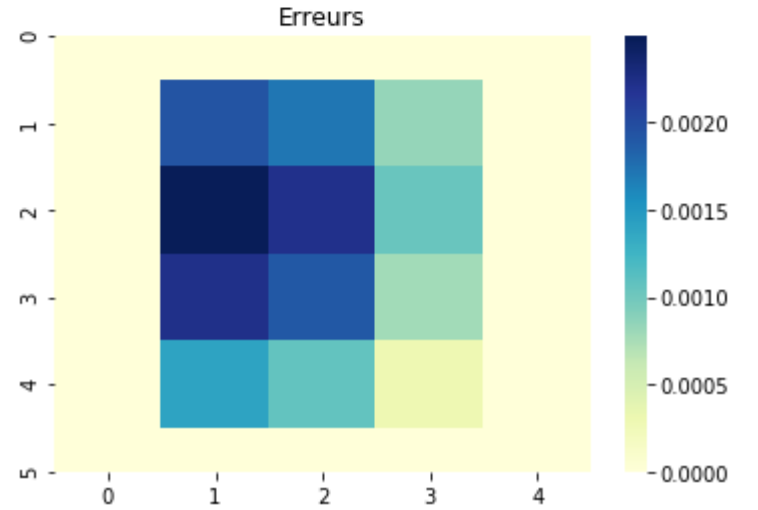
SolveurVF2D().draw_courbe(lambda x,y:np.cos(x)+np.sin(y), lambda x,y:np.cos(x)+np.sin(y), n,m)

```
#données pour le scénario d'une fonction non polynômiale
SolveurVF2D().draw_courbe(lambda x,y:np.cos(x)+np.sin(y), lambda x,y:np.cos(x)+np
```

```
[[ 4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.    0.  ]
 [-0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.    0.    0.    0.  ]
 [-1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.  ]
 [ 0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.  ]
 [ 0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.  ]
 [ 0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.  ]
 [ 0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.  ]
 [ 0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25]
 [ 0.    0.    0.    0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8 ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1 ]]
```


Solution calculée
0
1
2
3
4
5
0
1
2
3
4
1.75
1.50
1.25
1.00
0.75


Solution Exacte
0
1
2
3
4
5
0
1
2
3
4
1.75
1.50
1.25
1.00
0.75

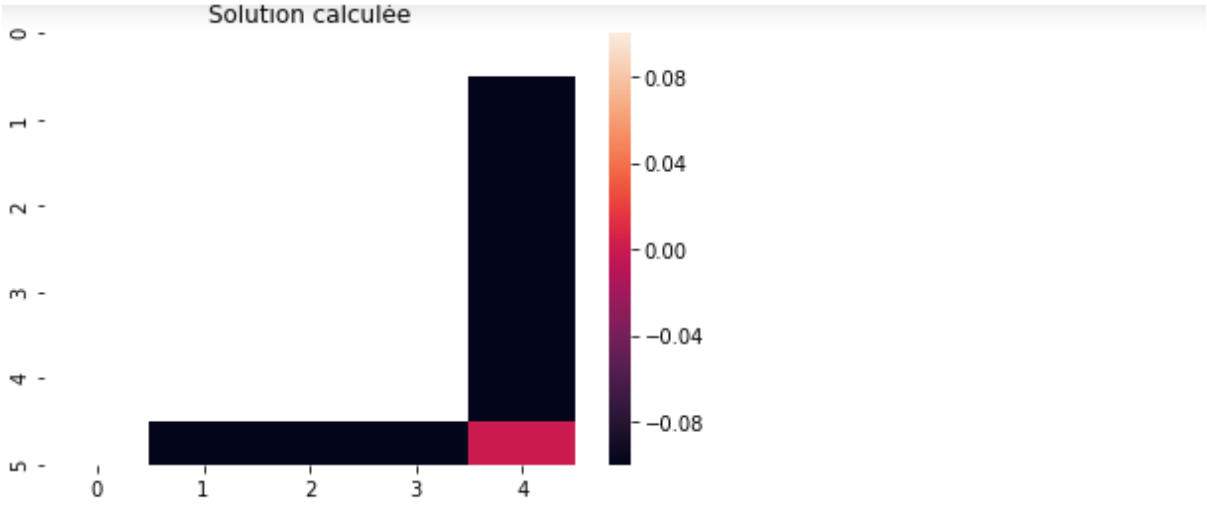

Erreurs
0
1
2
3
4
5
0
1
2
3
4
0.0020
0.0015
0.0010
0.0005
0.0000

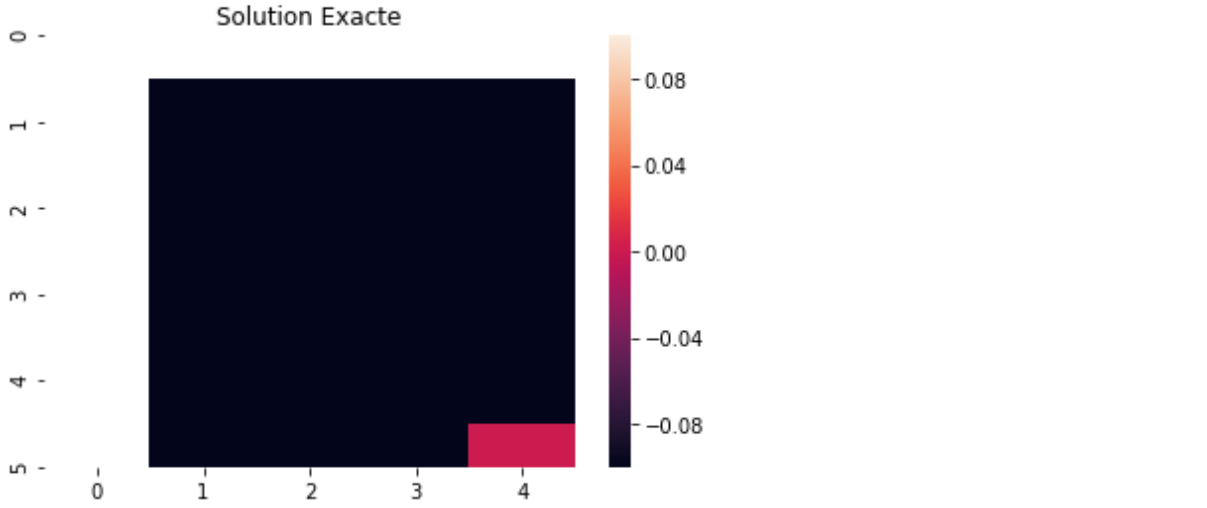
données pour le scénario d'une fonction logarithmique
SolveurVF2D().draw_courbe(lambda x,y:np.log(x)+np.log(y), lambda x,y:1/x**2+1/y**2, n,m)

```
#données pour le scénario d'une fonction logarithmique
SolveurVF2D().draw_courbe(lambda x,y:np.log(x)+np.log(y), lambda x,y:1/x**2+1/y**
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarn
ing: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarn
ing: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

```
[[ 4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.    0.  ]
 [-0.8   4.1  -0.8   0.   -1.25  0.    0.    0.    0.    0.    0.  ]
 [ 0.   -0.8   4.1   0.    0.   -1.25  0.    0.    0.    0.    0.  ]
 [-1.25  0.    0.    4.1  -0.8   0.   -1.25  0.    0.    0.    0.  ]
 [ 0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.    0.    0.  ]
 [ 0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25  0.    0.  ]
 [ 0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.   -1.25  0.  ]
 [ 0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8   0.   -1.25  0.  ]
 [ 0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1   0.    0.   -1.25]
 [ 0.    0.    0.    0.    0.    0.   -1.25  0.    0.    4.1  -0.8   0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1  -0.8 ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.   -1.25  0.   -0.8   4.1 ]]
```






```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:86: RuntimeWar
ning: invalid value encountered in subtract
```