Tom Cormery
Clément Dumeril

TP 2 info

Groupe TP 31

Ce TP a pour but de programmer et comparer différentes méthodes de résolution numérique d'un système linéaire.
Nous programmerons deux méthodes directes : Gauss et LU, et quatre méthodes itératives : Jacobi, Gauss-Seidel, Jacobi relaxée et Gauss-Seidel relaxée.
On étudiera aussi l'influence du critère d'arrêt sur la précision de la solution et sur le nombre d'itérations nécessaires.

Partie A :

Question 1 :

On a le système suivant :

$$\begin{cases} y'' + y' + y = f(x) & x \in [0,2] \\ y(0) = 0.05, & y(2) = 0.05 \end{cases}$$
$$\text{où } f(x) = 1 - x^2$$

On veut le mettre sous la forme Ay = f
On peut trouver A par la méthode des différences finies :
Pour tout i entre 1 et N on a :

$$f_i = \frac{(2+h)y_{i+1} + (2-h)y_{i-1} + (2h^2-4)y_i}{2h^2} = Ay_i$$

D'où

```
def f(x): #calcul de le fonction f
    return 1-x**2

N = 100  #taille du système d'équations
h = 2/(N+1) #pas de discrétisation
X = np.linspace(0,2,N+2) #intervalle [0;2] discrétisé
f = [f(X[i]) for i in range(1,N+1)] #calcul des valeurs de la fonction f
f[0] -= 0.05*(2-h)/(2*h**2) #modification de la première valeur de la matrice des valeurs de f
f[N-1] -= 0.05*(2+h)/(2*h**2) #modification de la dernière valeur de la matrice des valeurs de f

A = 1/(2*h**2)*tridiag(2*h**2-4,2+h,2-h,N) #définition de la matrice A
```

Question 2 :

Méthode LU : on utilise la décomposition LU et les méthodes de remontée et de descente programmées au TP 1

```
In [39]: solve_lu(A,f)
Out[39]:
array([0.05165888, 0.0536533 , 0.05597545, 0.05861724, 0.0615703 ,
       0.06482602, 0.06837553, 0.07220971, 0.07631919, 0.08069441,
       0.08532556, 0.09020262, 0.09531539, 0.10065343, 0.10620616,
       0.1119628 , 0.11791239, 0.1240438 , 0.13034578, 0.13680689,
       0.14341557, 0.15016012, 0.1570287 , 0.16400936, 0.17109005,
       0.17825859, 0.18550271, 0.19281005, 0.20016815, 0.20756449,
       0.21498646, 0.2224214 , 0.22985655, 0.23727915, 0.24467635,
       0.25203528, 0.25934301, 0.2665866 , 0.27375308, 0.28082945,
       0.2878027 , 0.29465982, 0.30138779, 0.30797358, 0.31440417,
       0.32066657, 0.32674778, 0.33263485, 0.33831481, 0.34377477,
       0.34900185, 0.3539832 , 0.35870602, 0.36315757, 0.36732514,
       0.37119609, 0.37475784, 0.37799785, 0.38090368, 0.38346292,
       0.38566327, 0.38749249, 0.38893842, 0.38998898, 0.39063217,
       0.3908561 , 0.39064896, 0.38999902, 0.38889468, 0.38732441,
       0.38527679, 0.38274051, 0.37970436, 0.37615726, 0.37208821,
       0.36748635, 0.36234091, 0.35664127, 0.35037689, 0.34353739,
       0.33611249, 0.32809204, 0.31946601, 0.31022451, 0.30035776,
       0.28985613, 0.2787101 , 0.26691031, 0.25444751, 0.24131258,
       0.22749657, 0.21299062, 0.19778603, 0.18187425, 0.16524685,
       0.14789553, 0.12981216, 0.11098872, 0.09141734, 0.0710903 ])
```

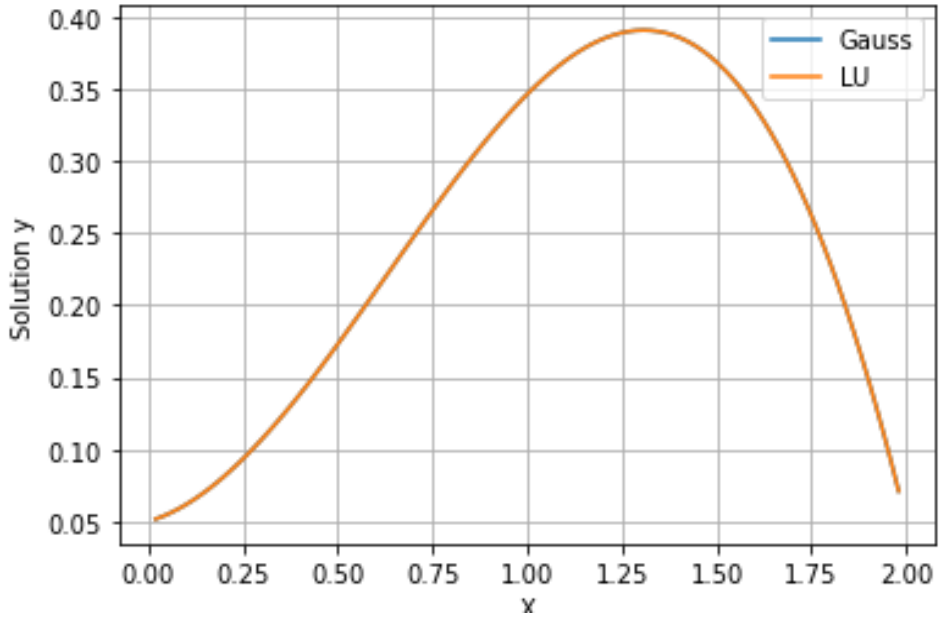
Méthode de Gauss : on utilise la fonction solve_gauss programmée au TP 1

```
In [40]: solve_gauss(A,f)
Out[40]:
array([0.05165888, 0.0536533 , 0.05597545, 0.05861724, 0.0615703 ,
       0.06482602, 0.06837553, 0.07220971, 0.07631919, 0.08069441,
       0.08532556, 0.09020262, 0.09531539, 0.10065343, 0.10620616,
       0.1119628 , 0.11791239, 0.1240438 , 0.13034578, 0.13680689,
       0.14341557, 0.15016012, 0.1570287 , 0.16400936, 0.17109005,
       0.17825859, 0.18550271, 0.19281005, 0.20016815, 0.20756449,
       0.21498646, 0.2224214 , 0.22985655, 0.23727915, 0.24467635,
       0.25203528, 0.25934301, 0.2665866 , 0.27375308, 0.28082945,
       0.2878027 , 0.29465982, 0.30138779, 0.30797358, 0.31440417,
       0.32066657, 0.32674778, 0.33263485, 0.33831481, 0.34377477,
       0.34900185, 0.3539832 , 0.35870602, 0.36315757, 0.36732514,
       0.37119609, 0.37475784, 0.37799785, 0.38090368, 0.38346292,
       0.38566327, 0.38749249, 0.38893842, 0.38998898, 0.39063217,
       0.3908561 , 0.39064896, 0.38999902, 0.38889468, 0.38732441,
       0.38527679, 0.38274051, 0.37970436, 0.37615726, 0.37208821,
       0.36748635, 0.36234091, 0.35664127, 0.35037689, 0.34353739,
       0.33611249, 0.32809204, 0.31946601, 0.31022451, 0.30035776,
       0.28985613, 0.2787101 , 0.26691031, 0.25444751, 0.24131258,
       0.22749657, 0.21299062, 0.19778603, 0.18187425, 0.16524685,
       0.14789553, 0.12981216, 0.11098872, 0.09141734, 0.0710903 ])
```

Question 3 :

```
In [8]: trace(solve_gauss(A,f),solve_lu(A,f))
```

On obtient :



Les deux courbes sont superposées donc la méthode de Gauss et la méthode LU semblent renvoyer le même résultat.

Question 4 :

```
Y = np.linalg.solve(A,f)
```

```
print(max(abs(solve_gauss(A,f)- Y)))

print(max(abs(solve_lu(A,f)- Y)))
```

```
In [9]: runfile('E:/TP 2 à rendre code.py', wdir='E:')
6.522560269672795e-15
2.220446049250313e-16
```

Les écarts calculés pour les deux méthodes sont très faibles, ces méthodes renvoient donc un résultat satisfaisant.
Pour la méthode LU, l'écart est légèrement plus faible mais cela reste négligeable. Ainsi les deux méthodes sont équivalentes.

Partie B :

Question 1 :
Par le cours on a les coefficients des matrices D, E, F suivants :

$$\begin{cases} D = a_{ij}\delta_{ij} \\ E_{ij} = \begin{cases} -a_{ij} & i > j \\ 0 & i \leq j \end{cases} \\ F_{ij} = \begin{cases} 0 & i \geq j \\ -a_{ij} & i < j \end{cases} \end{cases}$$

Question 2 :
Voir le programme

Question 3 :

Pour Y_0 = np.zeros(N), on a la suite (Y_k) qui converge pour GS et J. On regarde si elle converge bien vers A^{-1}*f pour les deux méthodes.

Gauss-Seidel :

```
In [73]: max(abs(np.linalg.inv(A)@f - gauss_seidel(A,f,10**(-4))))
Out[73]: 5.758851888881811e-05
```

Jacobi :

```
In [74]: max(abs(np.linalg.inv(A)@f - Jacobi2(A,f,10**(-4))))
Out[74]: 5.1873442651395596e-05
```

(Y_k) converge vers A^{-1}*f pour GS et J donc les deux méthodes sont **consistantes**.

Vérifions maintenant que $\boldsymbol{\rho(B) < 1}$ avec B la matrice d'itération de Gauss-Seidel puis de Jacobi.

Gauss-Seidel :

```
In [87]: rayon_spectralGS(A)
Out[87]: (0.9993266797613974+0j)
```

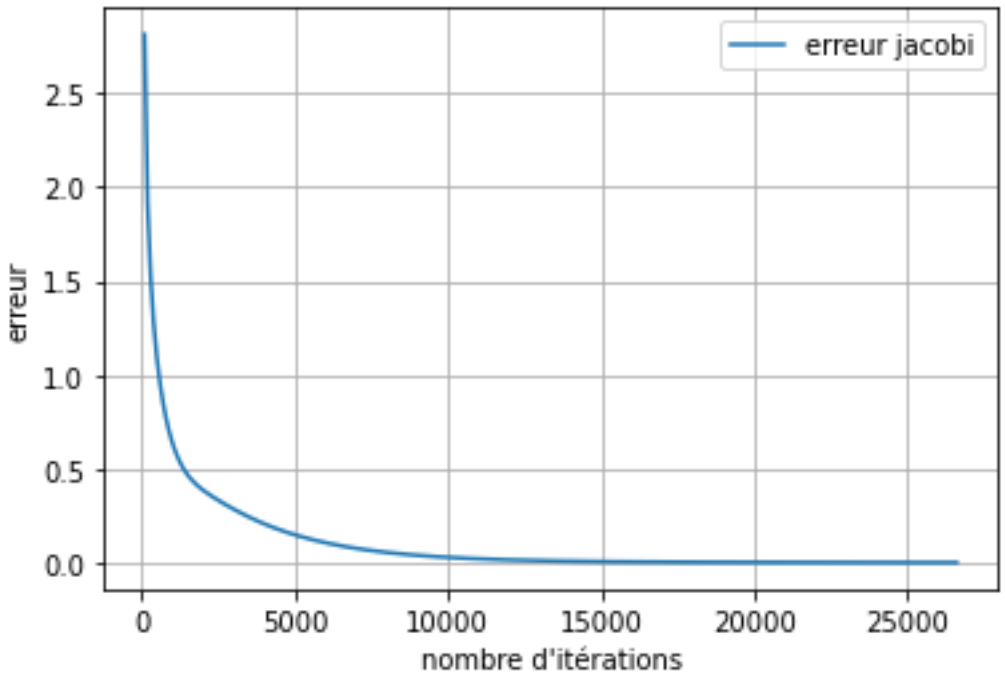
Jacobi :

```
In [86]: rayon_spectralJ(A)
Out[86]: 0.9996632831915941
```

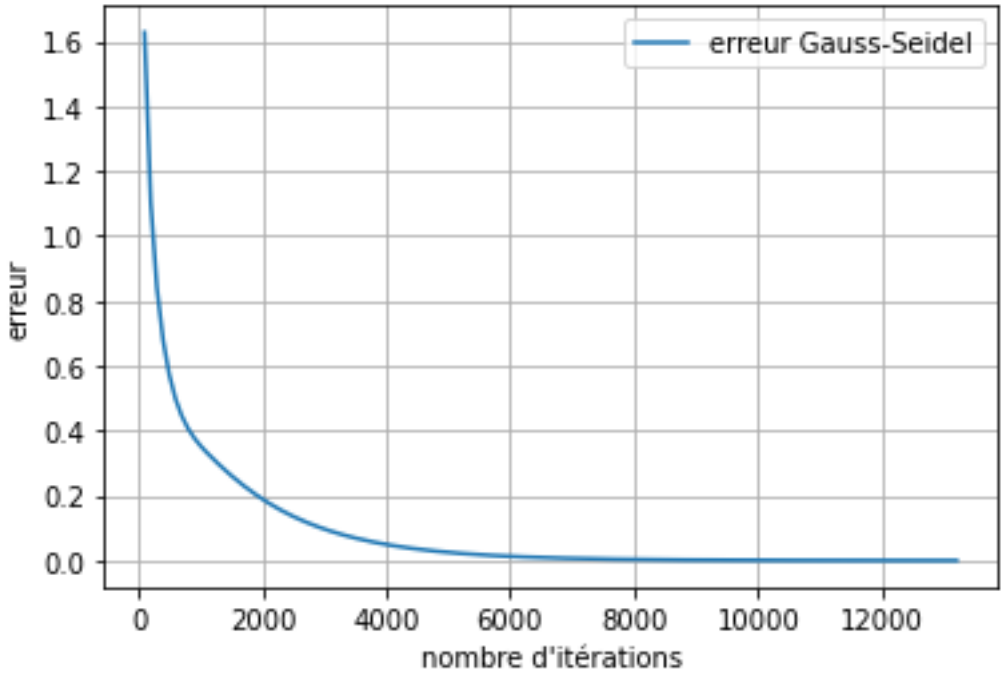
Dans les deux cas la condition est bien vérifiée.
Donc les méthodes J et GS **convergent**.

Question 4 :

Graphes de l'erreur pour la méthode de Jacobi en fonction du nombre d'itérations
Epsilon = 10^{-4}



Graphes de l'erreur pour la méthode de Gauss-Seidel en fonction du nombre d'itérations
Epsilon = 10^{-4}



Les deux méthodes convergent au bout d'un certain nombre d'itérations. On remarque que pour la méthode de Jacobi il faut environ le double du nombre d'itérations nécessaires pour la méthode de Gauss-Seidel.

Question 5 :

On a Y = np.linalg.solve(A,f)

Jacobi :

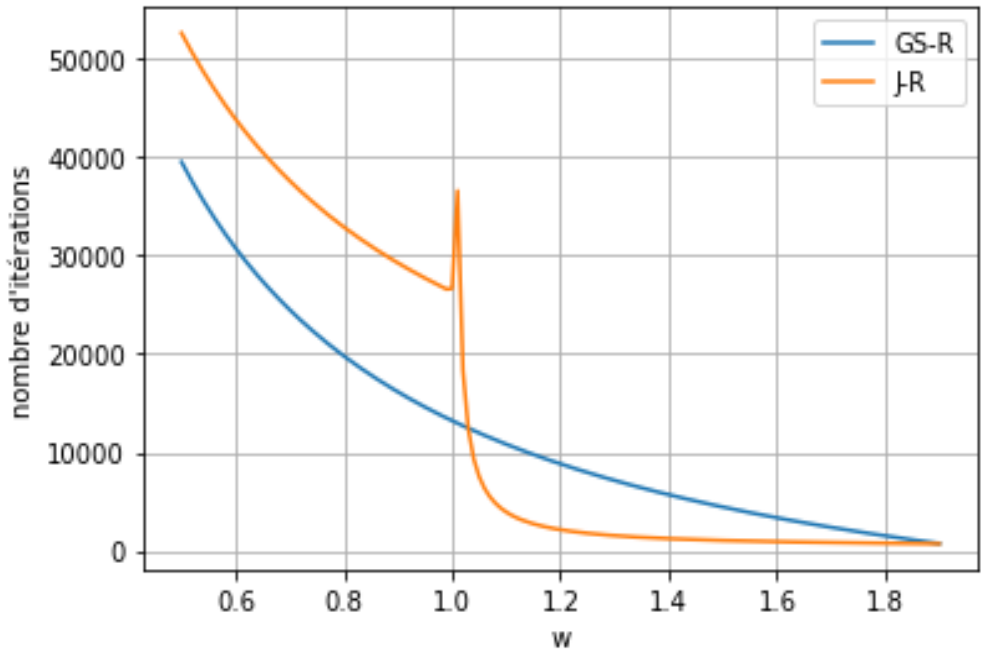
```
In [100]: max(abs(Jacobi2(A,f,10**(-4)) - Y))
Out[100]: 5.187344265150662e-05
```

Gauss-Seidel :

```
In [101]: max(abs(gauss_seidel(A,f,10**(-4)) - Y))
Out[101]: 5.758851888898464e-05
```

Question 6 :

Graphe du nombre d'itérations en fonction du paramètre de relaxation w pour les méthodes de Gauss-Seidel relaxée et de Jacobi relaxée



Pour w>1, la méthode de Jacobi relaxée diverge, d'où le nombre d'itérations qui diminue brusquement à partir de w=1.

Pour Gauss-Seidel, plus le paramètre de relaxation w est grand, plus le nombre d'itérations est faible. Ainsi pour résoudre le système donné dans ce TP, il est préférable de choisir un w proche de 1,9.

Estimation du nombre d'itérations nécessaires pour la convergence :

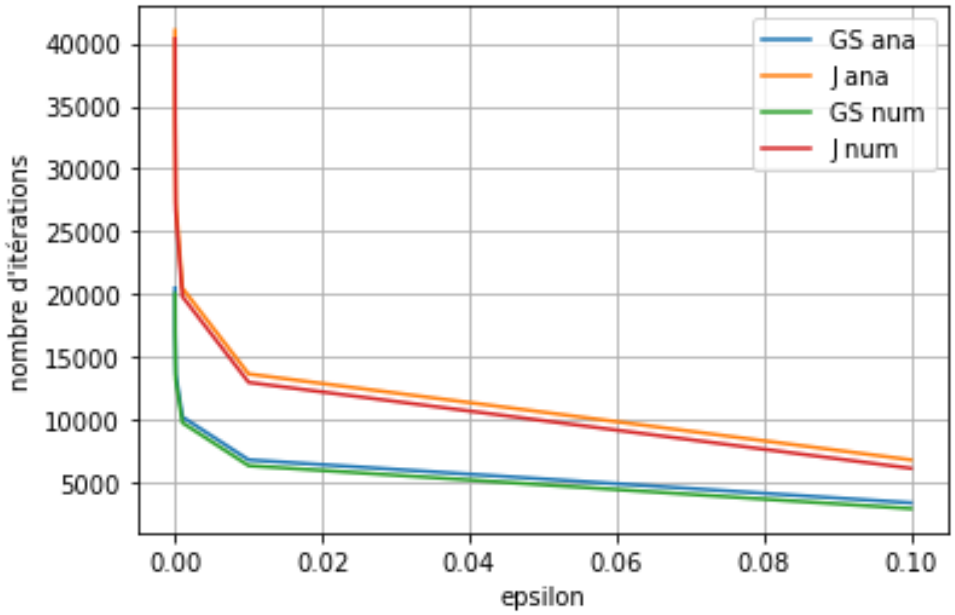
a) Jacobi : tableau du nombre d'itérations analytique pour epsilon de 10^{-1} à 10^{-6}

```
In [11]: ItJ
Out[11]:
array([ 6837.19183067, 13674.38366135, 20511.57549202, 27348.7673227 ,
       34185.95915337, 41023.15098405])
```

Gauss-Seidel : tableau du nombre d'itérations analytique pour epsilon de 10^{-1} à 10^{-6}

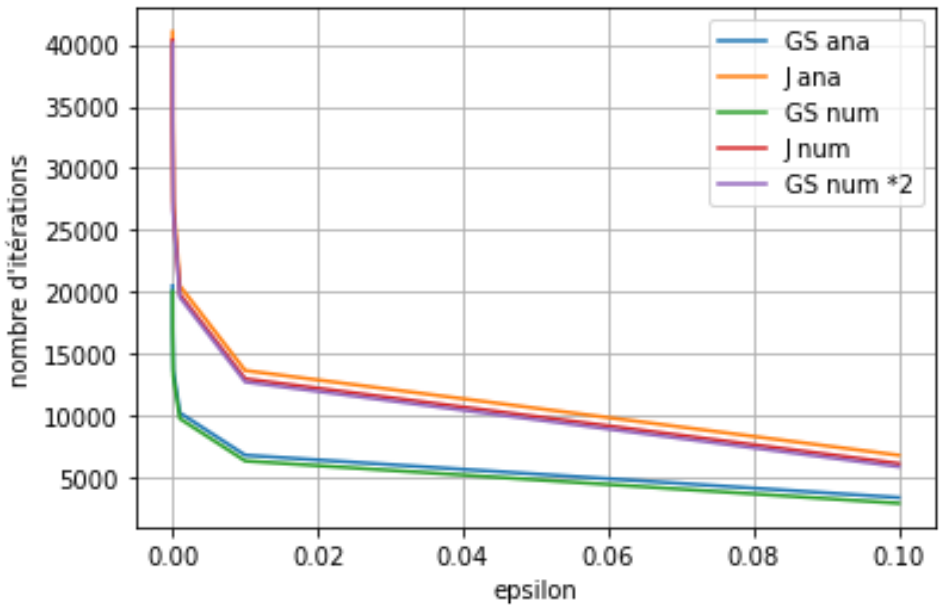
```
In [12]: ItGS
Out[12]:
array([ 3418.59591534-0.j,  6837.19183068-0.j, 10255.78774601-0.j,
       13674.38366135-0.j, 17092.97957669-0.j, 20511.57549203-0.j])
```

Nombre d'itérations en fonction d'epsilon pour les méthodes de Jacobi et Gauss-Seidel :



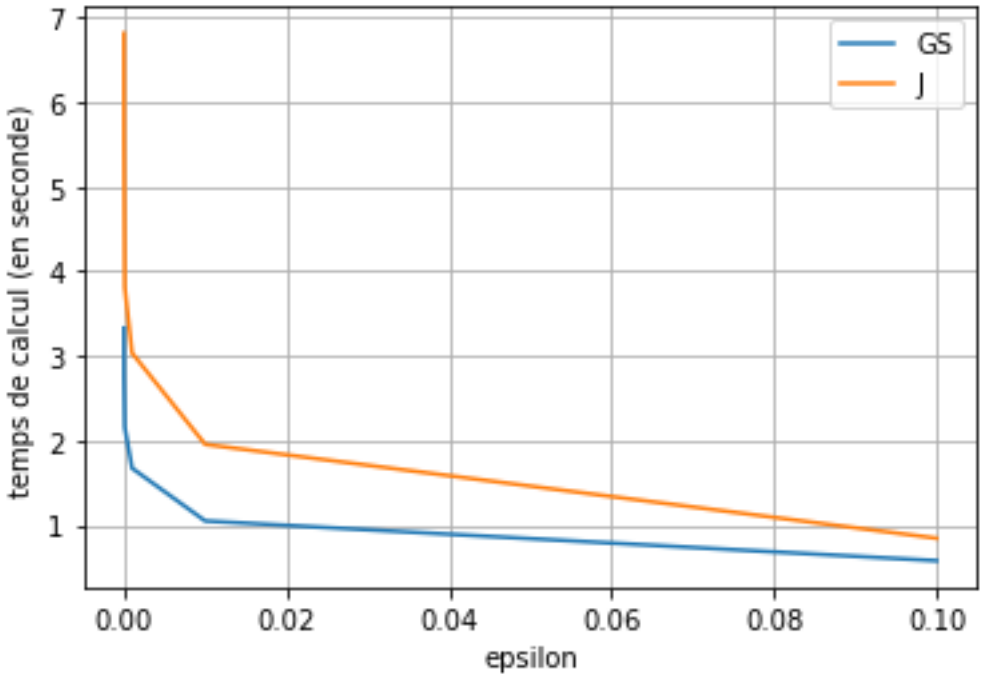
Le nombre d'itérations analytique est légèrement supérieur au nombre d'itérations numérique pour les deux méthodes. C'est un résultat satisfaisant.

On ajoute la courbe du double du nombre d'itérations de Gauss-Seidel :



On voit que la courbe du nombre d'itérations numérique de Gauss-Seidel est superposée avec la courbe du nombre d'itérations numérique de Jacobi. On en déduit que la méthode de Gauss-Seidel a besoin deux fois moins d'itérations que la méthode de Jacobi.

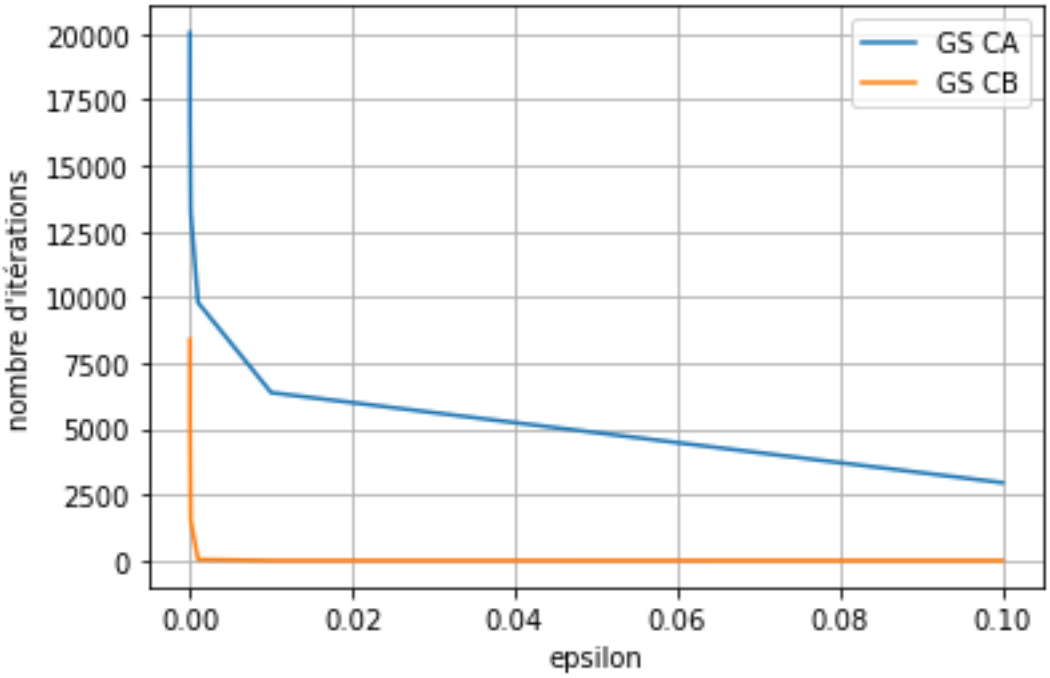
Temps de calcul en fonction de la tolérance epsilon :



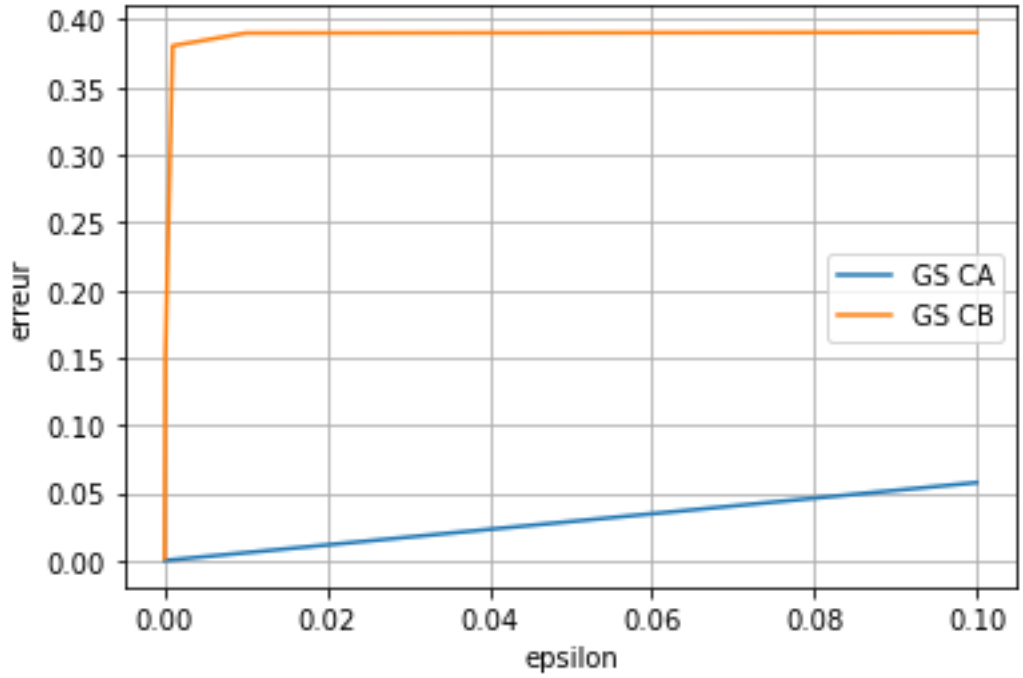
On remarque que la méthode de Gauss-Seidel va deux fois plus vite que celle de Jacobi.

b) <u>On définit un deuxième critère d'arrêt pour Gauss-Seidel que l'on compare au premier :</u>

Evolution du nombre d'itérations en fonction de la tolérance epsilon pour les critères A et B



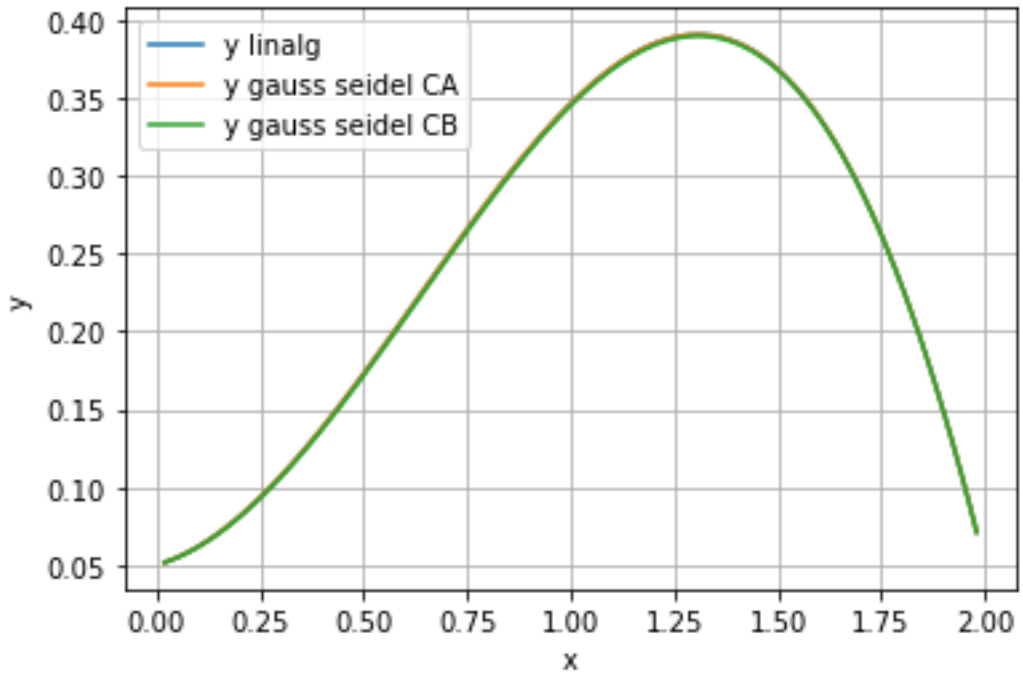
Evolution de l'erreur en fonction de la tolérance epsilon pour les critères A et B



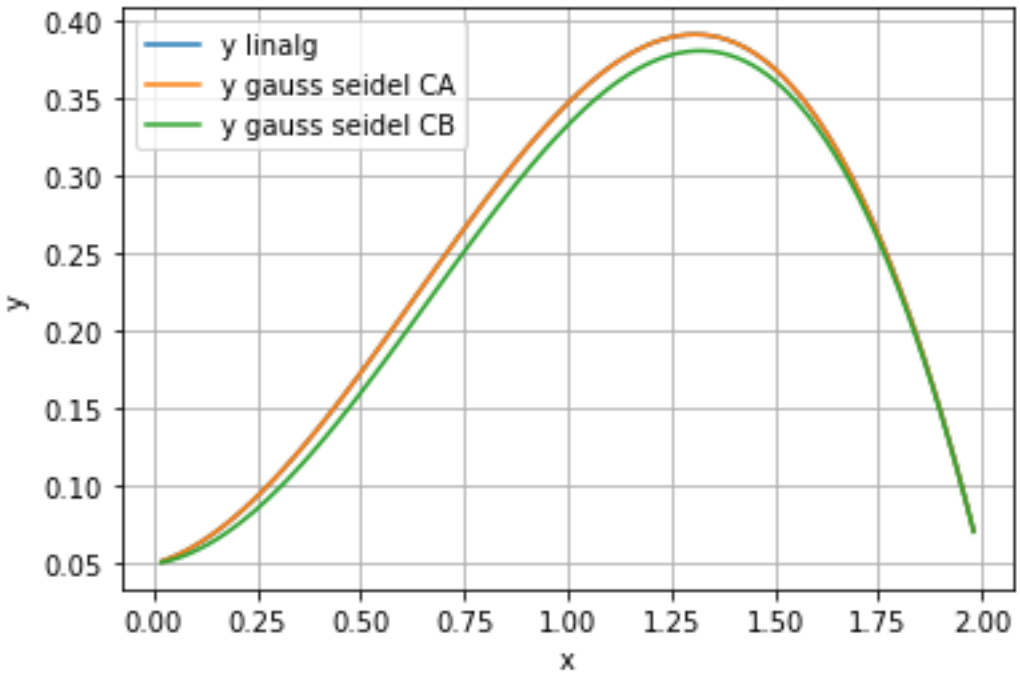
Plus epsilon est petit, plus il y a d'itérations pour les deux critères.
On voit que l'erreur est d'autant plus faible qu'epsilon est petit pour le critère A. Cependant pour le critère B l'erreur devient acceptable seulement lorsqu'epsilon est plus petit que 10^{-5}.

Graphe de la solution ylinalg et des solutions calculées par notre méthode de Gauss-Seidel pour les critères A et B en fonction de x pour chaque valeur d'epsilon

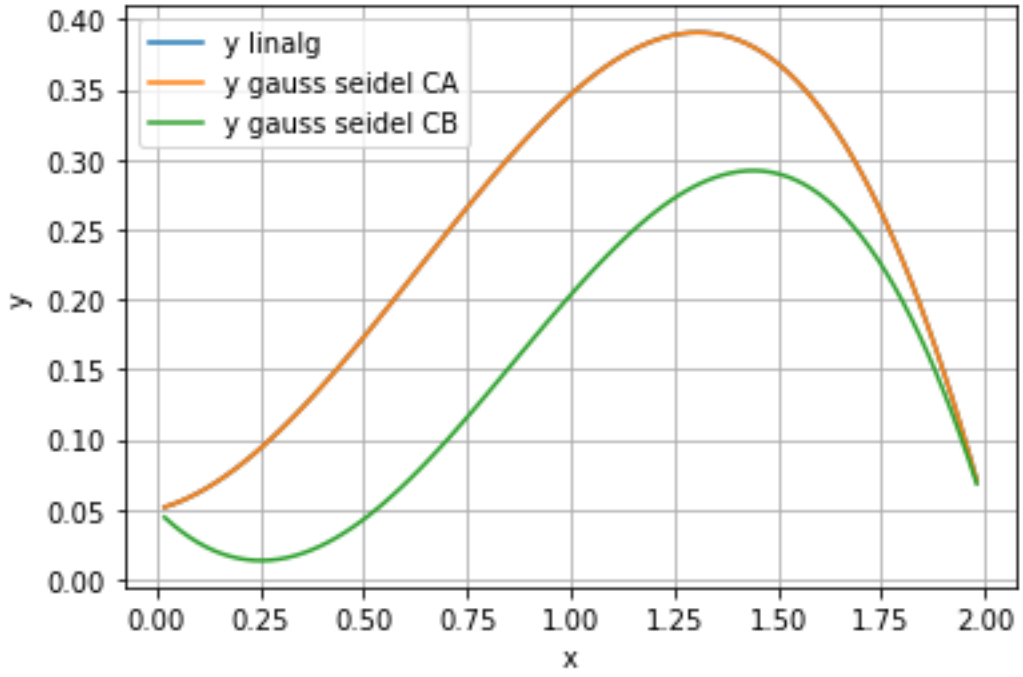
Epsilon = 10^{-6} :



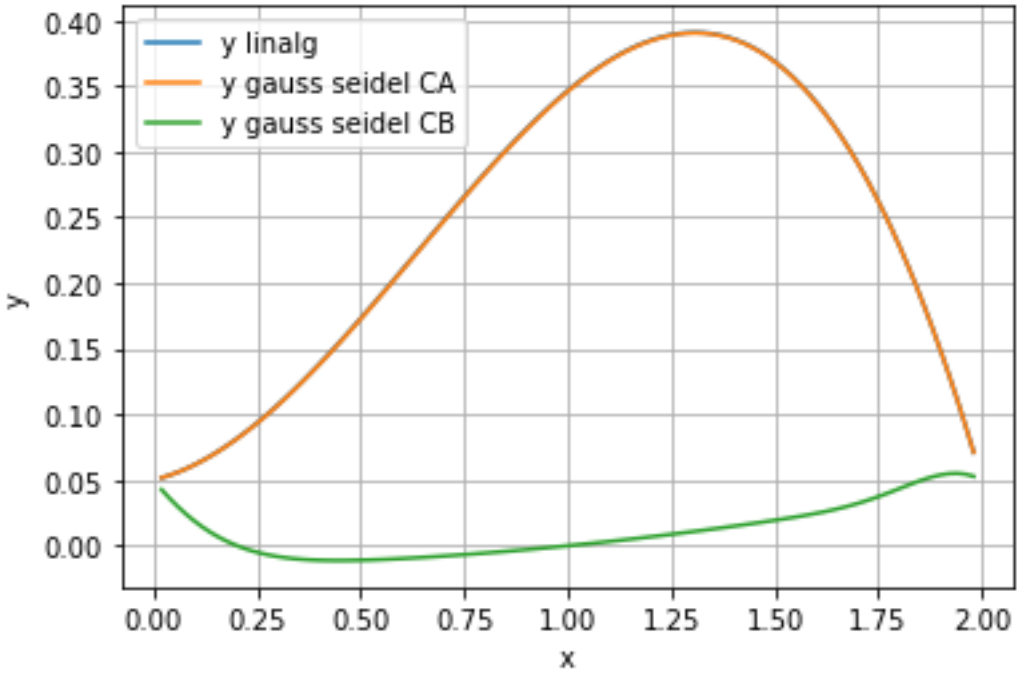
Epsilon = 10^{-5} :



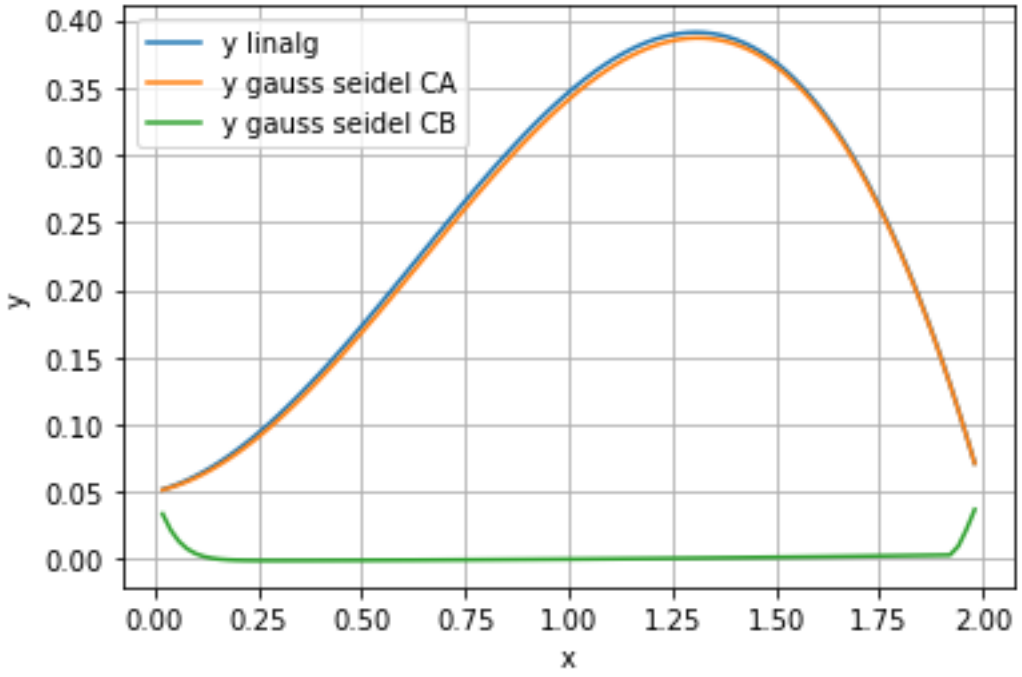
Epsilon = 10^{-4} :



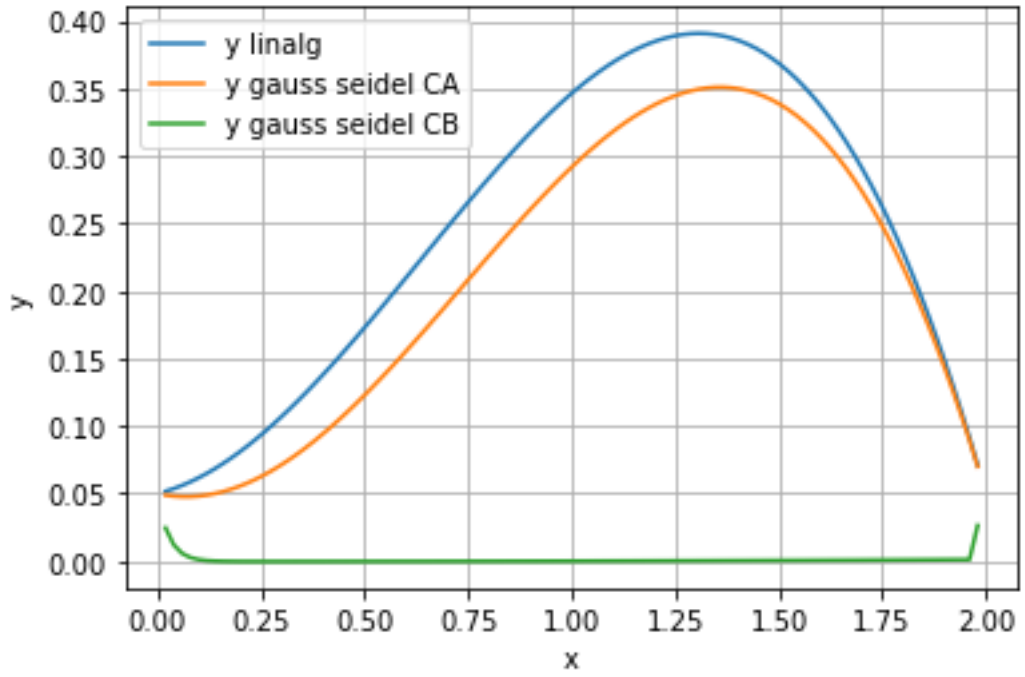
Epsilon = 10^{-3} :



Epsilon = 10^{-2} :



Epsilon = 10^{-1} :



On remarque que plus epsilon est petit plus les solutions sont précises. Cela se vérifie d'autant plus avec le critère B dont la solution est totalement erronée avec un epsilon trop grand.
Pour le critère A, la solution de Gauss Seidel commence à devenir satisfaisante pour un epsilon inférieur à 10^{-2}. Alors que pour le critère B, la solution de Gauss Seidel commence à devenir satisfaisante pour un epsilon inférieur à 10^{-5} voire 10^{-6}.

c) Conclusion :

Il existe plusieurs critères d'arrêt. Dans ce TP nous avons testé le critère A pour les méthodes de Jacobi et Gauss-Seidel et le critère B uniquement pour Gauss-Seidel.
Pour le critère A, Jacobi doit faire deux fois plus d'itérations que Gauss-Seidel et a un temps de calcul deux fois plus élevé. Gauss-Seidel semble donc être la méthode à utiliser parmi les deux si on veut optimiser le temps de calcul.
Nous avons ensuite comparé le critère A avec le critère B pour la méthode de Gauss Seidel et nous pouvons conclure que le critère B n'est pas adapté à notre problème. En effet, son utilisation nécessite une tolérance très faible et donc beaucoup d'itérations ce qui implique un temps de calcul plus long.