

Assignment 4 - Systemes linéaires

May 5, 2025

1 Assignment 4: Méthodes itératives pour systèmes linéaires

Avant de voir le code disponible de ce test et avant de commencer à rédiger vos réponses, prenez le temps de réfléchir à la manière dont vous pouvez organiser le travail.

- Pensez à quelles parties du test utiliseront des fonctions écrites/résultats obtenus dans les parties précédentes.
- Réfléchissez à la structure de votre code (vous pouvez faire un brouillon sur papier).
- Réfléchissez aux sections du cours qui vous seront utiles pour l'analyse de vos résultats.

On considère les systèmes linéaires paramétrisés de dimension n de la forme $A_\beta \mathbf{x} = \mathbf{b}_\beta$, où

$$A_\beta = \begin{pmatrix} f(1)g(0) & f(\beta)g(0) & f(\beta^2)g(0) & 0 & \dots & 0 \\ f(\beta)g(1) & f(1)g(1) & f(\beta)g(1) & f(\beta^2)g(1) & 0 & 0 \\ f(\beta^2)g(2) & f(\beta)g(2) & f(1)g(2) & f(\beta)g(2) & f(\beta^2)g(2) & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \dots & 0 & f(\beta^2)g(n-1) & f(\beta)g(n-1) & f(1)g(n-1) \end{pmatrix} \in \mathbb{R}^{n \times n};$$

$\mathbf{b}_\beta \in \mathbb{R}^n$ tel que $\mathbf{b}_\beta = \mathbf{A}_\beta \mathbf{1}$ (c-à-d $\mathbf{x} = \mathbf{1}$ est la solution exacte).

On choisi $f(x) = 1 - \cos\left(\frac{\pi x}{2}\right)$, $g(x) = \frac{1}{n} \cosh\left(\frac{\pi x}{n}\right)$ et $\beta \in [0, 1]$.

Pour une taille n et une valeur beta de β données, on peut calculer A et b en utilisant la fonction `matrix(n, beta)` définie plus loin.

```
[ ]: # importing libraries used in this book
import numpy as np
import matplotlib.pyplot as plt
```

```
[ ]: def matrix(n, beta) :
    # Returns A_\beta and b given in the exercice.

    f = lambda x: 1.0 - np.cos(np.pi/2*x)
    g = lambda x: np.cosh(np.pi * x / n) / n
    vec = np.array([g(k) for k in range(n)])
```

```

# diagonals of the matrix
d = [f(1)*vec,
      f(beta**1)*vec[:-1],
      f(beta**2)*vec[:-2]]
A = np.diag(d[0],k=0) + \
    np.diag(d[1],k=1) + np.diag(d[1],k=-1) + \
    np.diag(d[2],k=2) + np.diag(d[2],k=-2)

b = A.dot(np.ones(n))

return A,b

```

1.1 Partie 1

Écrire une fonction Richardson, qui implemente la méthode de Richardson stationnaire préconditionnée. La fonction doit avoir la structure suivante:

```

def Richardson(A, b, x0, P, alpha, max_iter, tol) :
    # Stationary Richardson method to approximate the solution of Ax=b
    #
    # INPUTS:
    # A           : system matrix
    # b           : system vector
    # x0          : initial guess
    # P           : preconditioner
    # alpha       : constant relaxation parameter
    # max_iter    : maximum number of iterations
    # tol         : tolerance on the relative residual
    #
    # OUTPUTS
    # xk          : approximate solution to the linear system
    # rk          : list of relative norms of the residuals

```

```

[ ]: def Richardson(A, b, x0, P, alpha, max_iter, tol) :
    # Stationary Richardson method to approximate the solution of Ax=b
    #
    # INPUTS:
    # A           : system matrix
    # b           : system vector
    # x0          : initial guess
    # P           : preconditioner
    # alpha       : constant relaxation parameter
    # max_iter    : maximum number of iterations
    # tol         : tolerance on the relative residual
    #
    # OUTPUTS
    # xk          : approximate solution to the linear system

```

```

# res_norm          : list of relative norms of the residuals

xk = np.copy(x0)
rk = ## COMPLETE HERE !! (residual)
res_norm = [np.linalg.norm(rk)]
rel_res = res_norm[-1] / np.linalg.norm(b)

k = 0
while k < max_iter and rel_res > tol:

    ## COMPLETE HERE !! (few lines of code)

    res_norm.append(np.linalg.norm(rk))
    rel_res = res_norm[-1] / np.linalg.norm(b)

    k += 1

if k >= max_iter and rel_res >= tol:
    print(f'Preconditioned Gradient did not converge in {max_iter} iterations;
↪ '
        f'the relative residual is {rel_res:.3e}')
else:
    print(f'Preconditioned Gradient method converged in {k+1} iterations '
        f'with a relative residual of {rel_res:.3e}')

return ## COMPLETE HERE !!

```

1.2 Partie 2

On sait que si la matrice A_β est diagonale dominante stricte par ligne, alors la méthode de Gauss-Seidel converge. Établir si c'est le cas lorsqu'on fixe $\beta = 0.4$, pour une taille $n=20$.

Dans le cas affirmatif, calculer la solution du système par la commande **Richardson** en choisissant les méthodes de Jacobi et Gauss-Seidel. On fixe une tolérance $\text{tol} = 10^{-10}$ et le point de départ $\mathbf{x}^{(0)} = \mathbf{0}$.

Est-ce que les résultats obtenus sont en accord avec la théorie ? (*Commentaire 1*)

Aide: la somme des valeurs (en valeur absolue) dans toutes les lignes d'une matrice M peut être calculée de la manière suivante: $v = \text{np.sum(np.abs(M), axis=1)}$. L'output v est un `numpy.array`, dont longueur est égale au nombre des lignes de la matrice M .

```

[ ]: n = 20
     beta = 0.4
     [A, b] = matrix(n, beta)

     # COMPLETE HERE !! (check if A is diagonally dominant --> see AIDE in text)

```

```
[ ]: x0 = np.zeros(n)
      tol = 1e-10
      nmax = 500

      print('\nJacobi preconditioner')
      P = # COMPLETE HERE !! (Jacobi preconditioner)
      x, _ = # COMPLETE HERE !! (call Richardson method)
      err = # COMPLETE HERE !! (compute norm of error with exact solution)
      print(f'The error is: {err:.3e}')

      print('\nGauss-Seidel preconditioner')
      P = # COMPLETE HERE !! (Gauss-Seidel preconditioner)
      x, _ = # COMPLETE HERE !! (call Richardson method)
      err = # COMPLETE HERE !! (compute norm of error with exact solution)
      print(f'The error is: {err:.3e}')
```

Commentaire 1

1.3 Partie 3

Tracer le graphe du rayon spectral $\rho(\beta) = \max |\lambda_{B(\beta)}|$ (utiliser la commande `np.linalg.eig` pour les valeurs propres) des matrices d'itération de Jacobi $B_J(\beta)$ et de Gauss-Seidel $B_{GS}(\beta)$ en fonction de β , pour $\beta = 0, 0.1, 0.2, \dots, 1$.

D'après ce graphe, que peut-on dire sur la convergence des deux méthodes en fonction du paramètre β ? Quelle méthode utiliseriez-vous pour résoudre le système dans le cas $\beta = 0.7$? Pourquoi? Est-ce qu'on s'attend au même nombre d'itérations que dans le cas $\beta = 0.4$? (*Commentaire 2*)

```
[ ]: n = 20
      betas = np.linspace(0,1,11)

      rho_j, rho_gs = [], []
      for beta in betas :
          [A, b] = matrix(n,beta)

          # Jacobi
          Pj = # COMPLETE HERE !! (Jacobi preconditioner)
          Bj = # COMPLETE HERE !! (Jacobi iteration matrix)
          rho_j.append(# COMPLETE HERE !! (maximal singular value in modulus))

          # Gauss-Seidel
          Pgs = # COMPLETE HERE !! (Gauss-Seidel preconditioner)
          Bgs = # COMPLETE HERE !! (Gauss-Seidel iteration matrix)
          rho_gs.append(# COMPLETE HERE !! (maximal singular value in modulus))
```

```
[ ]: plt.figure()

      plt.plot(betas, rho_j, 'b:o', label='Jacobi')
```

```
plt.plot(betas, rho_gs, 'g:o', label='Gauss-Seidel')

plt.plot(np.linspace(0,1,50), np.ones(50), 'r-');

plt.xlabel(r'$\beta$', fontsize=15)
plt.ylabel(r'$\rho(B)$', fontsize=15)
plt.xlim([0,1])
plt.legend(fontsize=15)
plt.grid(linestyle='--', linewidth=.5)
plt.show()
```

Commentaire 2

1.4 Partie 4

Maintenant, on considère la matrice A_β et le vecteur $\mathbf{b}_\beta$ que l'on obtient pour $n=100$. Dans ce cas, pour $\beta = 0.25$ et $\beta = 0.50$, tracer et comparer les graphes **semilogy** de la norme du résidu $\mathbf{r}^{(k)}$ en fonction du nombre d'itérations pour la méthode de Jacobi.

Donner un commentaire, en considérant les pentes des courbes obtenues. Est-ce qu'on s'attend de résultats similaires pour la méthode de Gauss-Seidel ? (*Commentaire 3*)

```
[ ]: n = 100
x0 = np.zeros(n)
tol = 1e-10
nmax = 500

beta = 0.25
[A, b] = matrix(n,beta)
P = # COMPLETE HERE !! (Jacobi preconditioner)
_, res_beta25 = # COMPLETE HERE !! (call Richardson method)

beta = 0.5
[A, b] = matrix(n,beta)
P = # COMPLETE HERE !! (Jacobi preconditioner)
_, res_beta50 = # COMPLETE HERE !! (call Richardson method)
```

```
[ ]: plt.figure()

plt.semilogy(res_beta25, 'b:o', label=r'$\beta = 0.25$')
plt.semilogy(res_beta50, 'g:o', label=r'$\beta = 0.50$')

plt.legend(fontsize=15)
plt.xlabel(r'$k$', fontsize=15)
plt.ylabel('Absolute Residual', fontsize=15);
plt.grid(linestyle='--', linewidth=.5)
plt.show()
```

Commentaire 3

1.5 Partie 5

D'après la théorie vue en classe, est-il possible d'améliorer la convergence de la méthode de Richardson stationnaire avec le préconditionneur de Jacobi ? Si oui, comment ?

Vérifiez votre hypothèse numériquement, en choisissant $n=100$ et $\beta = 0.5$. Donner un commentaire sur les résultats obtenus. (*Commentaire 4*)

```
[ ]: beta = 0.5
[A, b] = matrix(n,beta)

P = # COMPLETE HERE !! (Jacobi preconditioner)

## COMPLETE HERE !! (your idea to speed up convergence, few lines of code)
```

Commentaire 4

1.6 Partie 6

La convergence peut être encore améliorée si on considère des méthodes de Richardson non-stationnaires, dans lesquels la valeur du paramètre α change au cours des itérations. L'exemple le plus simple est donné par la méthode du *Gradient Preconditionee*.

Écrire une nouvelle fonction `PrecGradient`, en modifiant de manière appropriée la fonction `Richardson` créée dans la *Partie 1*. Vérifier la convergence de la méthode en prenant $n=100$ et $\beta = 0.5$, et en considérant: * le cas pas preconditionné (c-à-d $P = \text{np.eye}(n)$); * le préconditionneur de Jacobi.

Donner un commentaire sur la base des résultats obtenus. (*Commentaire 5*)

```
[ ]: def PrecGradient(A, b, x0, P, max_iter, tol) :
    # Preconditioned gradient method to approximate the solution of Ax=b
    #
    # INPUTS:
    # A          : system matrix
    # b          : system vector
    # x0         : initial guess
    # P          : preconditioner
    # max_iter   : maximum number of iterations
    # tol        : tolerance on the relative residual
    #
    # OUTPUTS
    # xk         : approximate solution to the linear system
    # res_norm   : list of relative norms of the residuals

    ## COMPLETE HERE !! (copy-paste the Richardson function and change it where
    ↪needed --> it is a one-line update!)
```

```

    if k >= max_iter and rel_res >= tol:
        print(f'Preconditioned Gradient did not converge in {max_iter} iterations;
        ↪ '
            f'the relative residual is {rel_res:.3e}')
    else:
        print(f'Preconditioned Gradient method converged in {k+1} iterations '
            f'with a relative residual of {rel_res:.3e}')

    return xk, res_norm

```

```

[ ]: n = 100
    beta = 0.5
    [A, b] = matrix(n, beta)

    x0 = np.zeros(n)
    tol = 1e-10
    nmax = 500

    print("Without preconditioning")
    P = np.eye(n)
    _ = # COMPLETE HERE !! (call Gradient method)
    print("\n")

    print("Using Jacobi preconditioner")
    P = # COMPLETE HERE !! (Jacobi preconditioner)
    _ = # COMPLETE HERE !! (call Gradient method)

```

Commentaire 5

2 Quelques petites questions finales (pas évaluées)

- What types of collaboration strategies did your group use?
 - Work in pairs on different sections.
 - Work individually on different sections.
 - Work together on the same section with one notebook opened.
 - Work together on the same section with multiple notebooks opened.
 - Other (please specify).
- How effective was your collaboration strategy today? Please rate from 1 (not at all) to 5 (very effective).
- How supported did you feel by your TA during the session today? Please rate from 1 (not at all) to 5 (very effective).

Please report your answers here. **Thank you!**