2025

PRESENTATION DE PROJET

# SYSTEME LINEAIRE EN GRANDE DIMENSION

**CATTEAU Elsa**
**PROUZET Charlotte**

# SOMMAIRE

## 1 Matrices larges

> Jacobi dense avec 3 boucles
> Jacobi dense avec 2 boucles
> Comparaisons
> Rayon spectral

## 2 Matrices creuses

> Jacobi Sparse
> Comparaison Jacobi Sparse / Jacobi dense
> Gauss Seidel
> Comparaison Jacobi Sparse / Gauss Seidel
> Comparaison Jacobi Dense / Gauss Seidel

## 3 Méthode SOR

> Comparaison SOR / Gauss Seidel
> Best Omega

# 01 MATRICES LARGES

**A = D - L - U**

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \quad D = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \quad L = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad U = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

**> FORME ITERATIVE**

$$x^{(k+1)} = D^{-1}(L+U)\, x^{(k)} + D^{-1} b$$

**> FORME PAR COMPOSANTE**

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j\,!=\,i} a_{ij}\, x_j^{(k)}\right) \quad \text{pour i=1,2.....,n}$$

# JACOBI DENSE AVEC 3 BOUCLES

```python
def jacobi_method(A, b, x0, tol=1e-5, max_iter=1000):


    start_time = time.time()

    n = A.shape[0]
    x = x0.copy()
    errors = []
    for k in range(max_iter):
        x_new=np.zeros_like(x)
        for i in range(n):
            sum=0
            for j in range(n):
                if j!=i:
                    sum=sum+A[i,j]*x[j]
            x_new[i]=(1/A[i,i]) * (b[i] - sum)
        error = np.linalg.norm(x_new - x)
        errors.append(error)
        x = x_new
        if error < tol:
            break

    return x, k + 1, errors, time_taken, r
```

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j\,!=\,i} a_{ij}\, x_j^{(k)}\right) \quad \text{pour i=1,2.....,n}$$

# JACOBI DENSE AVEC 2 BOUCLES

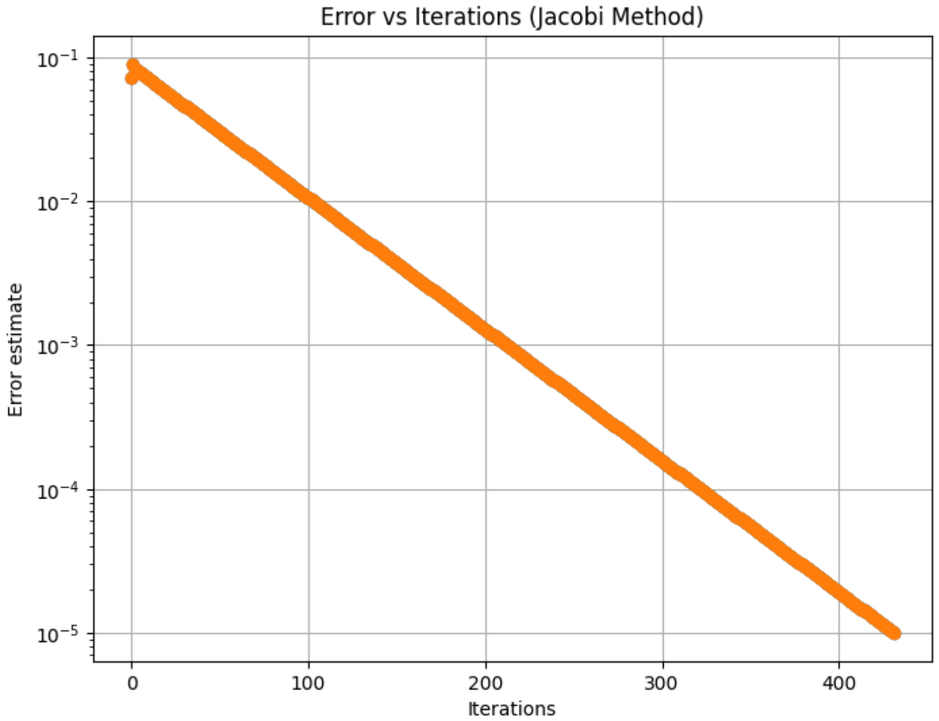
```python
def jacobi_method(A, b, x0, tol=1e-5, max_iter=1000):

  start_time=time.time()

  n = A.shape[0]
  x = x0.copy()
  errors = []
  for k in range(max_iter):
    x_new=np.zeros_like(x)
    for i in range(n):
        x_new[i]=(1/A[i,i]) * (b[i] - np.dot(A[i,:],x) + A[i,i]*x[i]) #on utilise le produit scalaire
    error = np.linalg.norm(x_new - x)
    errors.append(error)
    x = x_new
    if error < tol:
      break

  end_time = time.time()
  time_taken = end_time - start_time

  return x, k + 1, errors, time_taken, r
```

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j\,!=\,i} a_{ij}\, x_j^{(k)}\right) \quad \text{pour i=1,2.....,n}$$

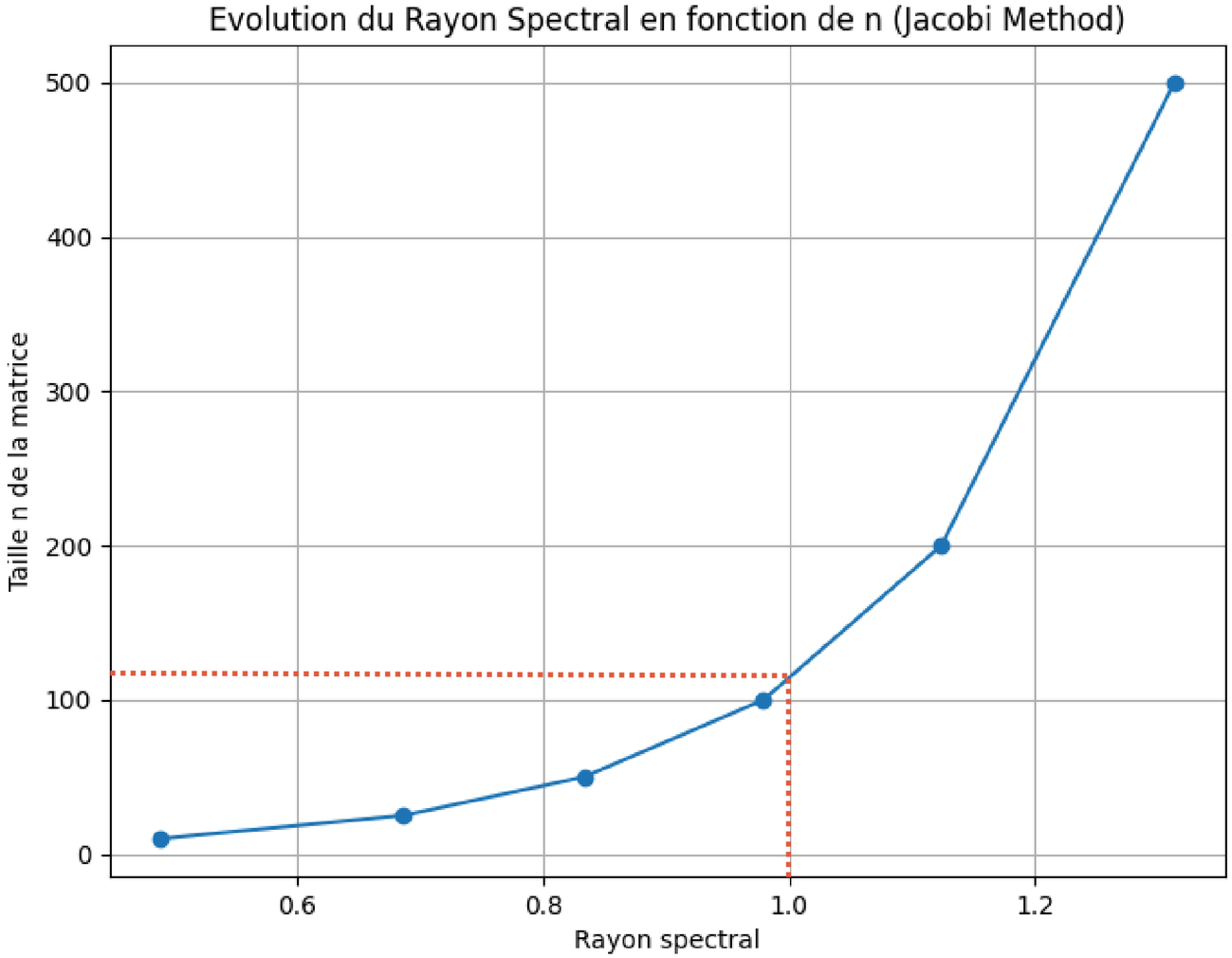
$$\sum_{j\,!=\,i} a_{ij}\, x_j^{(k)} = \sum a_{ij}\, x_j^{(k)} - a_{ii}\, x_i^{(k)}$$

# COMPARAISON

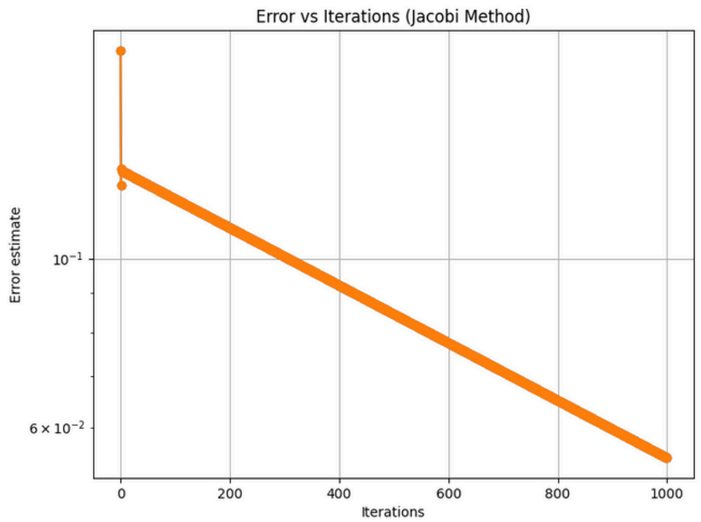
| | Nombre d'itérations | Temps d'exécution | Rayon Spectral |
|---|---|---|---|
| Jacobi dense avec 3 boucles | 443 | 7.34s | 0.979 |
| Jacobi dense avec 2 boucles | 446 | 0.54s | 0.979 |



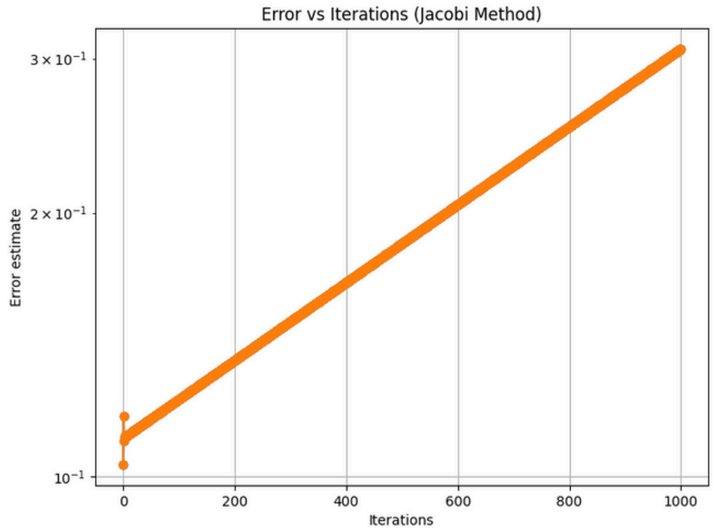
# RAYON SPECTRAL



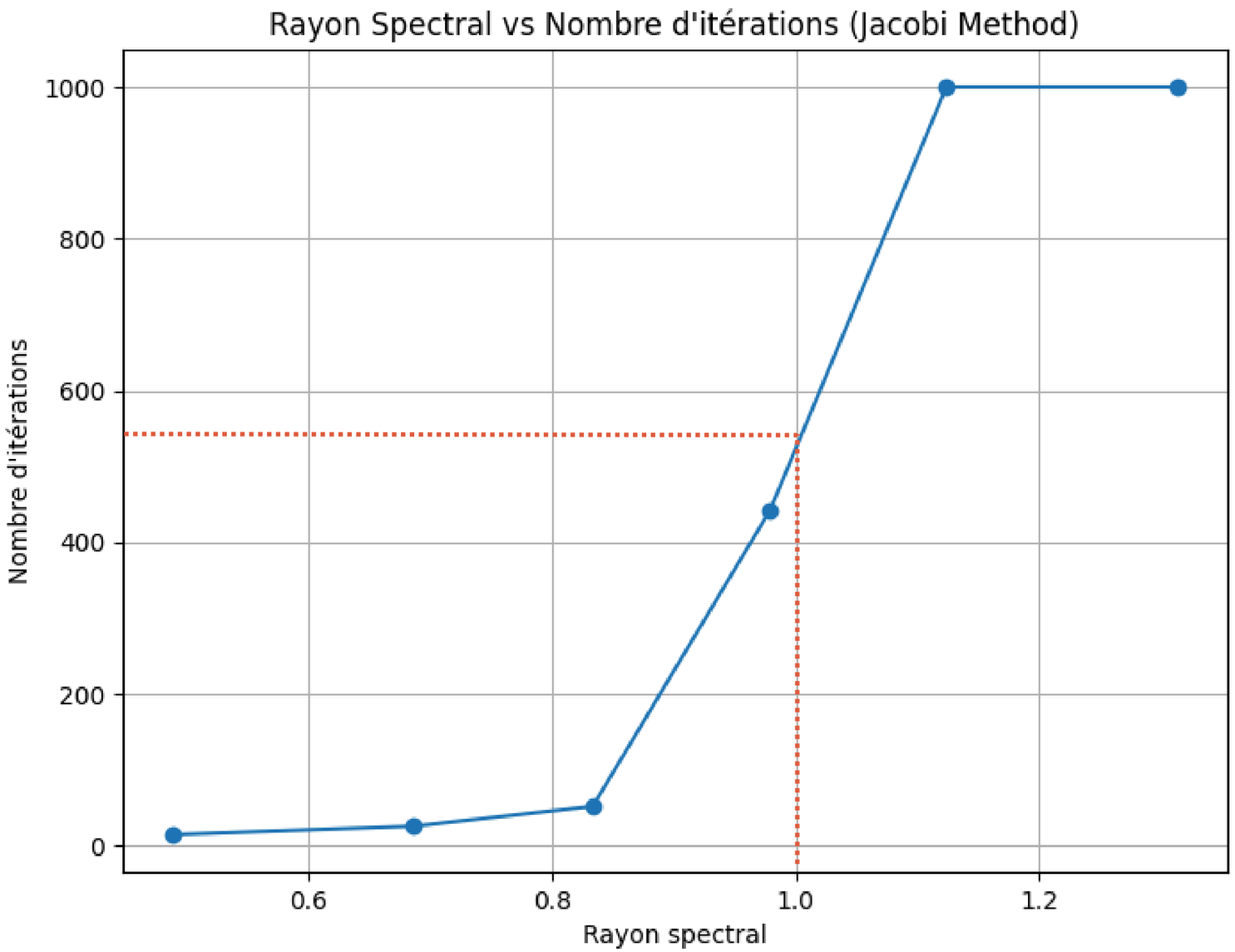

n= 110




n= 111



# RAYON SPECTRAL



# MATRICES CREUSES

02

# JACOBI SPARSE

```python
def jacobi_sparse(A, b, x0, tol=1e-7, max_iter=10000):
    start_time=time.time()
    x=x0.copy()
    D1=1/A.diagonal()
    LU=A-sparse.diags(A.diagonal())
    for k in range(max_iter):
        x_new = D1*(b-LU.dot(x))
        error = np.linalg.norm(x_new-x)
        if error < tol:
            break
        x = x_new
    end_time=time.time()
    time_taken=end_time-start_time
    return x, k + 1, time_taken
```

$$(L + U)$$

$$x = D^{-1}(b + (L + U)x)$$

# COMPARAISON JACOBI SPARSE VS JACOBI DENSE

En posant n=100:

| | Nombre d'itérations | Temps d'exécution |
|---|---|---|
| Jacobi Dense | 17 | 0.0336s |
| Jacobi Sparse | 19 | 0.0013s |

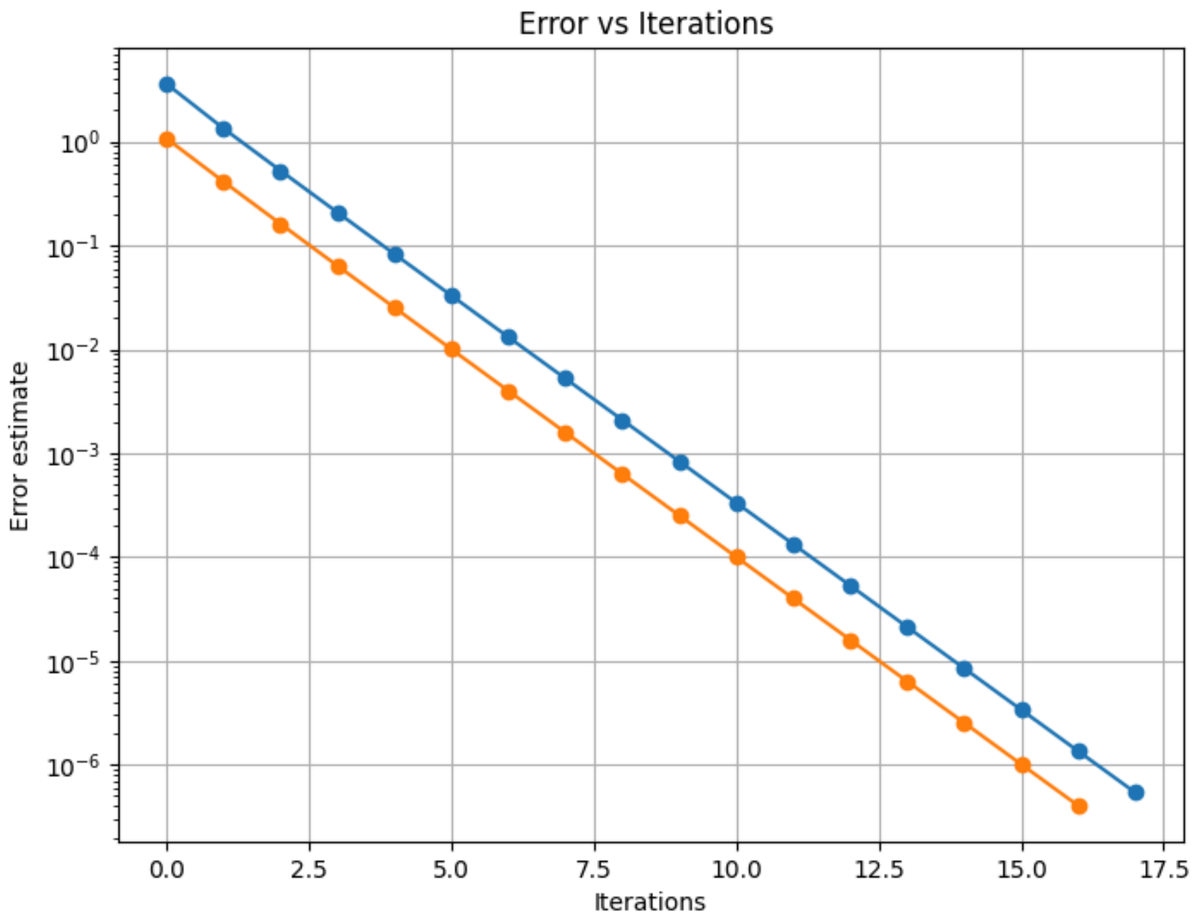
En posant n=1000:

| | Nombre d'itérations | Temps d'exécution |
|---|---|---|
| Jacobi Dense | 18 | 0.4359s |
| Jacobi Sparse | 20 | 0.0177s |

>> Jacobi Sparse est plus rapide que Jacobi Dense

# COMPARAISON JACOBI SPARSE VS JACOBI DENSE

## VITESSE DE CONVERGENCE




>> Jacobi Sparse a une vitesse de convergence plus rapide que celle de Jacobi Dense

# GÉNÉRER DES MATRICES TRIDIAGONALES SIMPLES

```python
def generate_simple_sparse_tridiagonal_matrix(n, diagonal_value=10, off_diagonal_value=4):
    main_diag = np.full(n, diagonal_value)
    off_diag = np.full(n-1, off_diagonal_value)

    # Construct sparse matrix
    data = np.concatenate([main_diag,off_diag,off_diag])
    rows = np.concatenate([np.arange(n), np.arange(n-1), np.arange(1,n)])
    cols = np.concatenate([np.arange(n), np.arange(1,n), np.arange(n-1)])
    As = csr_matrix((data, (rows, cols)), shape=(n, n))

    # Construct dense matrix for reference
    A_dense = np.zeros((n, n))
    A_dense[n-1,n-1]=diagonal_value
    for i in range(n-1):
        A_dense[i, i] = diagonal_value
        A_dense[i,i+1]=off_diagonal_value
        A_dense[i+1,i]=off_diagonal_value
    b = np.random.rand(n)
    return As, A_dense, b
```

# GÉNÉRER DES MATRICES TRIDIAGONALES CREUSES

```python
def generate_sparse_tridiagonal_matrix(n):
    h=1/(n+1)
    diagonal_value=2
    off_diagonal_value=-1
    main_diag = np.full(n, diagonal_value)
    off_diag = np.full(n-1, off_diagonal_value)
    data = np.concatenate([main_diag,off_diag,off_diag])
    rows = np.concatenate([np.arange(n), np.arange(n-1), np.arange(1,n)])
    cols = np.concatenate([np.arange(n), np.arange(1,n), np.arange(n-1)])
    A = (1/(h**2))*csr_matrix((data, (rows, cols)), shape=(n, n))

    # Construct dense matrix for reference
    A_dense = np.zeros((n, n))
    A_dense[n-1,n-1]=diagonal_value
    for i in range(n-1):
        A_dense[i, i] = diagonal_value
        A_dense[i,i+1]=off_diagonal_value
        A_dense[i+1,i]=off_diagonal_value
    A_dense=(1/(h**2))*A_dense
    b = np.random.rand(n)
    return A,  A_dense, b
```

UTILISATION DE LA MÉTHODE DU LAPLACIEN

# GAUSS SEIDEL

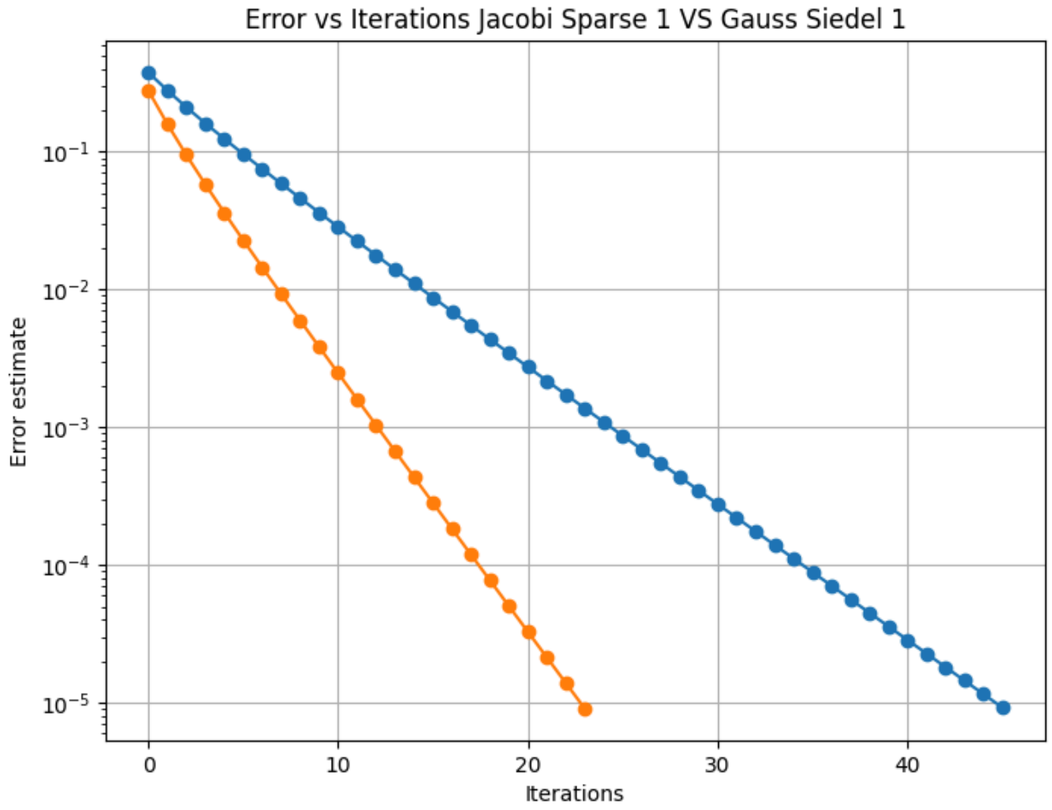
```python
def gauss_seidel_sparse_with_error(A, b, x0, x_exact, tol=1e-5, max_iter=1000):
    x=x0.copy()
    C=sparse.tril(A)  #correspond à D-L
    U=sparse.triu(A)-sparse.diags(A.diagonal())
    errors = []
    C1=sparse.linalg.inv(C)
    for k in range(max_iter):
        x_new = C1*(b-U.dot(x))
        error = np.linalg.norm(x_new-x_exact)
        errors.append(error)
        if error < tol:
            break
        x = x_new
    return x, k + 1, errors
```

$$x = C^{-1}(b + Ux)$$

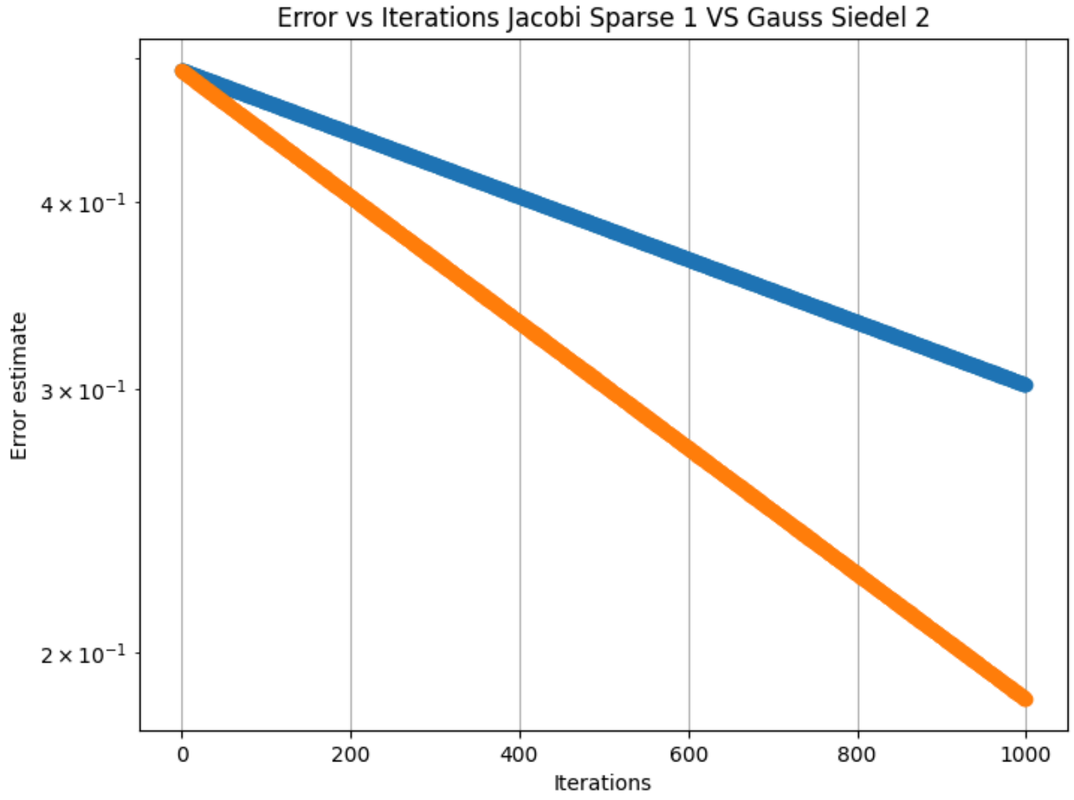
# COMPARAISON JACOBI SPARSE / GAUSS SEIDEL

## VITESSE DE CONVERGENCE

> Matrices tridiagonales simples




> Matrices tridiagonales creuses



# COMPARAISON JACOBI SPARSE / GAUSS SEIDEL

## TEMPS D'EXECUTION DU PROGRAMME

| | Jacobi Sparse | Gauss Seidel |
|---|---|---|
| generate_simple_sparse_tridiagonal_matrix | 0.0175s | 0.1136s |
| generate_sparse_tridiagonal_matrix | 0.0601s | 0.2383s |

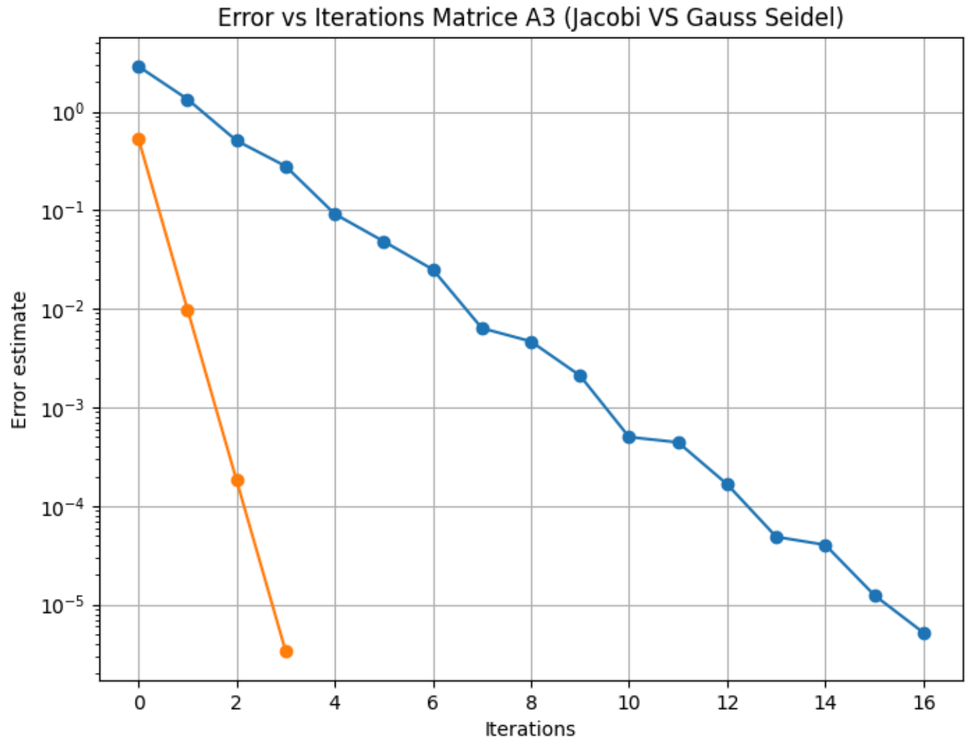
>Le temps d'exécution de la méthode de Gauss Seidel est plus long que celui de la méthode de Jacobi.

> Or la méthode de Gauss Siedel a une vitesse de convergence plus rapide que celle de Jacobi Sparse.

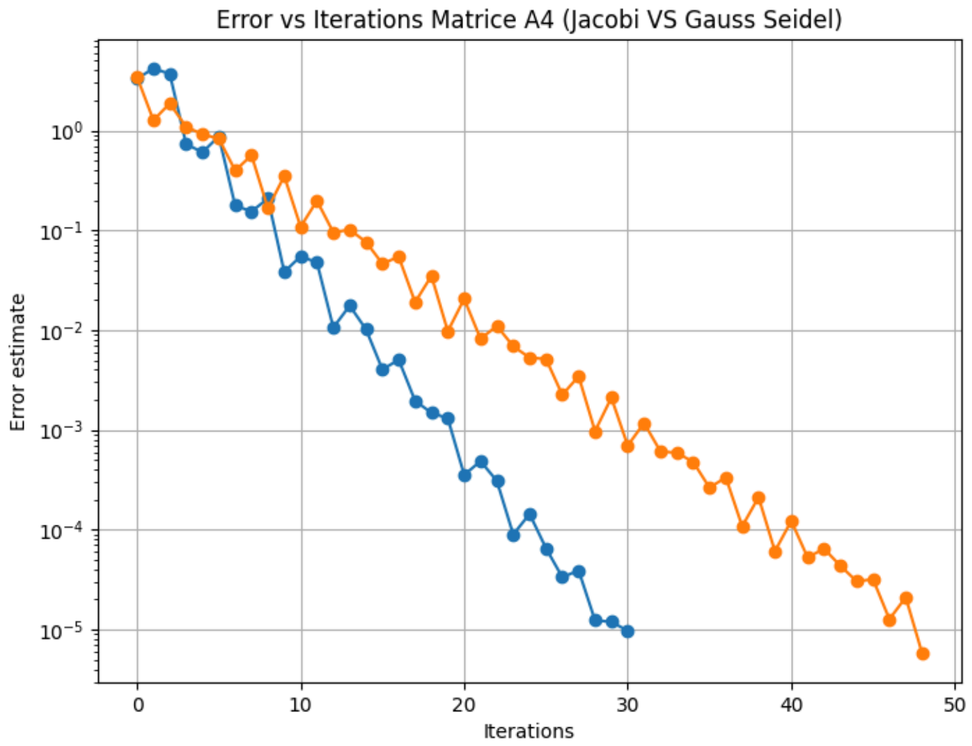
# COMPARAISON JACOBI DENSE / GAUSS SEIDEL DENSE

## APPLICATION AVEC LES MATRICES A3 ET A4

### MATRICE A3




### MATRICE A4



# METHODE
# SUCCESSIVE
# OVER
# RELAXATION

03

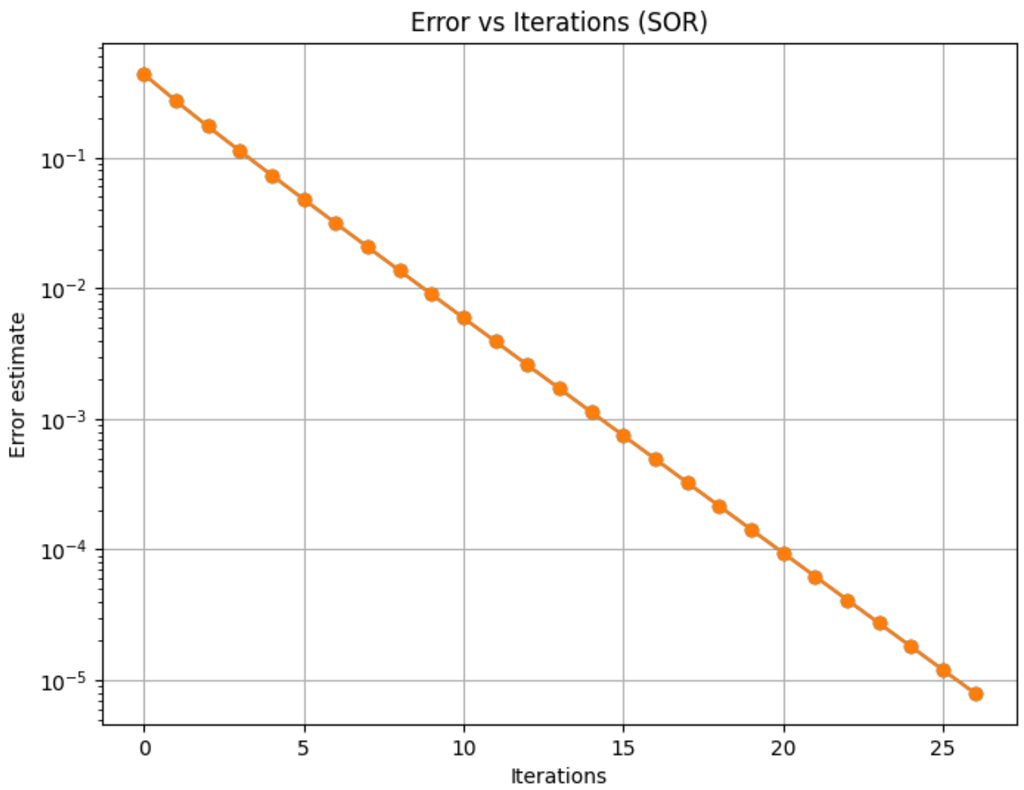
# SUCCESSIVE OVER RELAXATION (SOR)

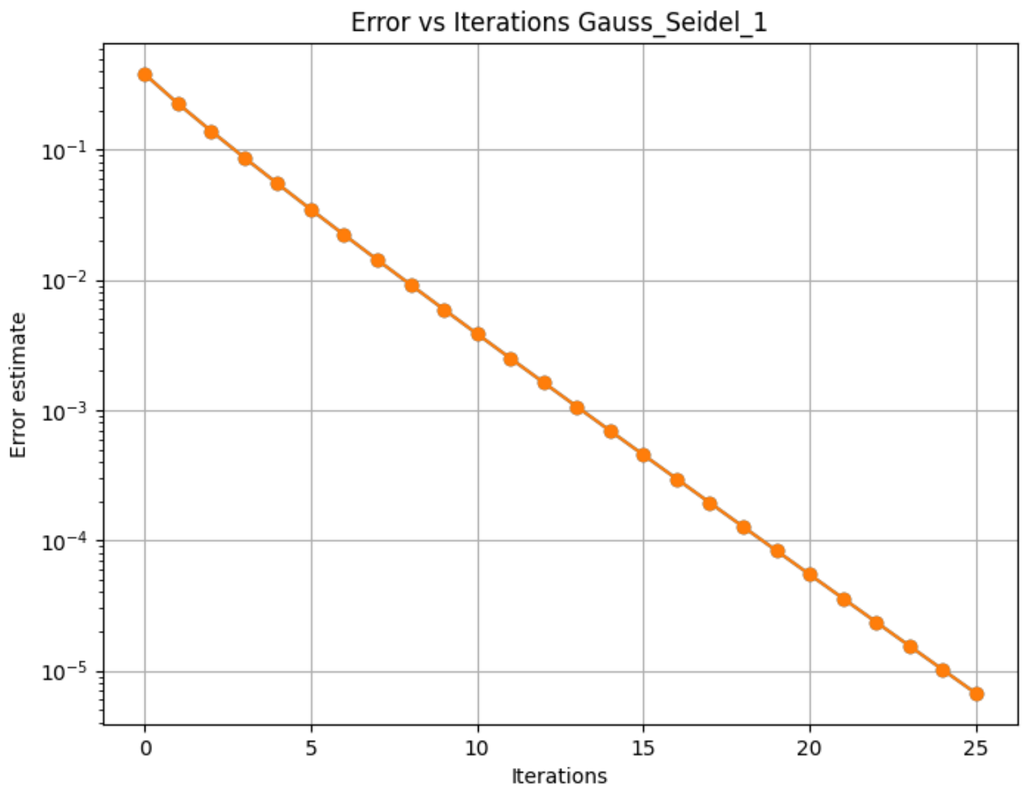
```
def successive_over_relaxation(A,b,x0, x_exact, tol=1e-5, max_iter=1000,w=1):
    x=x0.copy()
    D=sparse.diags(A.diagonal())
    L=sparse.tril(A,k=-1)
    U=sparse.triu(A,k=1)
    C=(1/w)*(D+w*L)
    errors = []
    C1=sparse.linalg.inv(C)
    for k in range(max_iter):
        x_new = C1.dot(b-(((w-1)/w)*D+U).dot(x))
        error = np.linalg.norm(x_new-x_exact)
        errors.append(error)
        if error < tol:
            break
        x = x_new
    return x, k + 1, errors
```

$$x = C^{-1}(b - (\frac{\omega-1}{\omega}D - U)x)$$

# COMPARAISON SOR / GAUSS SEIDEL

## AVEC OMEGA=1 POUR LA MÉTHODE SOR



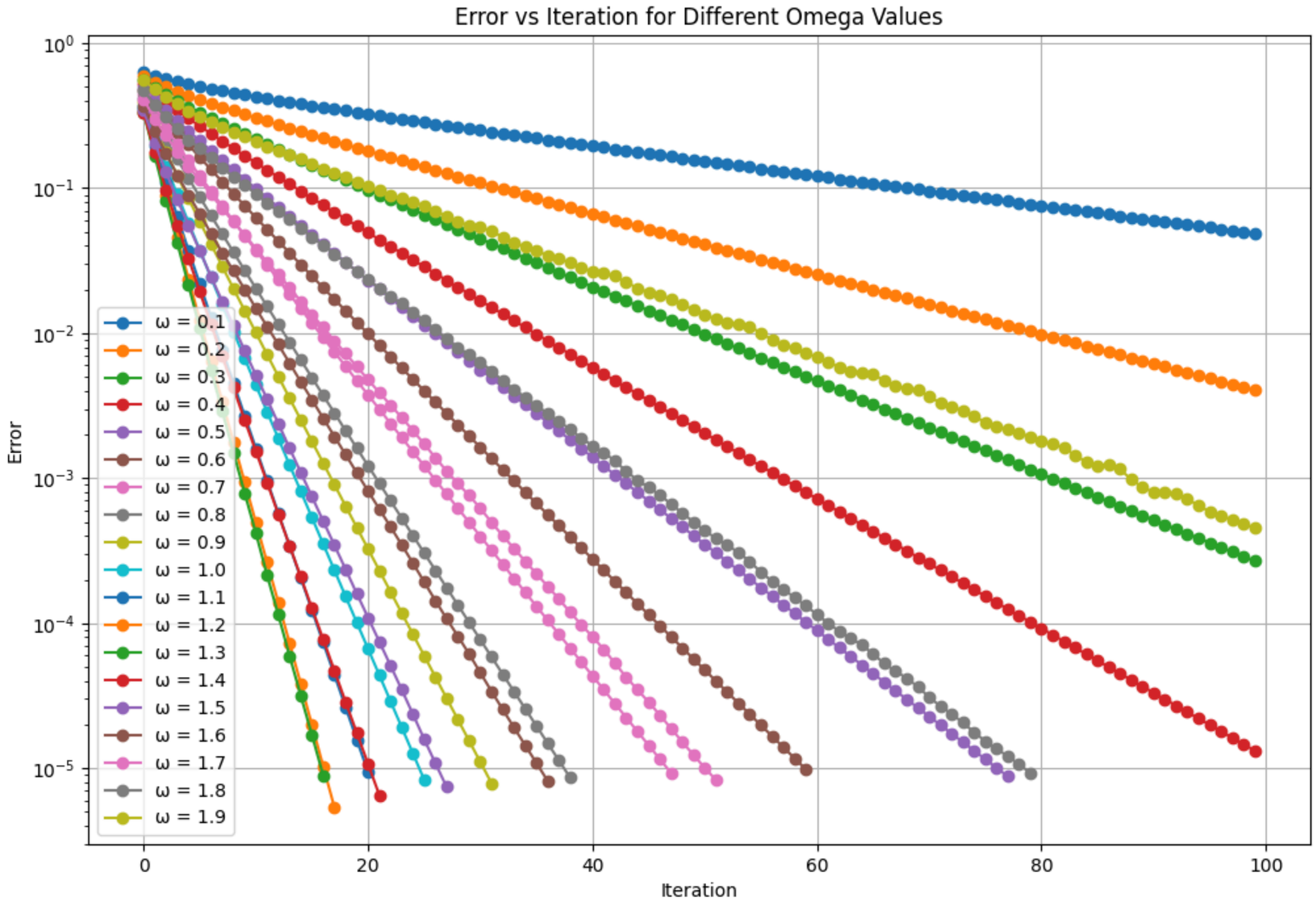



# FONCTION BEST OMEGA

```
def best_omega(As, b,x0,x_exact, tol=1e-5, max_iter=100):
    min_iter=max_iter
    best_w=0
    val=np.arange(0.1,2.0,0.1)
    for w in val:
        x, iter, errors = Sucessive_Over_Relaxation(As, b, w, x0, x_exact, tol=tol, max_iter=max_iter)
        if (iter<min_iter):
            min_iter=iter
            best_w=w
    return best_w
```

LA FONCTION RENVOIE 1.2 POUR N=100

# BEST OMEGA




> Best_Omega = 1.3

# CONCLUSION

SOR

GAUSS SEIDEL

JACOBI SPARSE

JACOBI DENSE