

1 大問2

1.1 (1)

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 %matplotlib inline
```

1.1.1 問題設定

Aは1000次とした。

bは1000次で各値は0から100までのfloat型の乱数を作成した。

乱数はseedをとって各手法で固定した。

また、許容誤差は $\text{tol} = 1 \times 10^{-6}$ とし、全ての計算で停止条件は $\|Ax - b\| < \text{tol}$ で取った。

```
In [2]: 1 # 次数の設定
2 degree = 1000
3
4 # x_{k-1}とx_kの差がtol未満なら収束したとみなす
5 tol = 1e-6
6
7 max_iter = 50000

In [3]: 1 # A: 本レポートではtridiagonal_matrix(2), tridiagonal_matrix(20)を用いる
2 def tridiagonal_matrix(c, n = degree):
3     A = np.zeros([n, n])
4     for i in range(n):
5         if i == 0:
6             A[i, i] = c
7             A[i, i + 1] = -1
8         elif i == n - 1:
9             A[i, i - 1] = -1
10            A[i, i] = c
11        else:
12            A[i, i - 1] = -1
13            A[i, i] = c
14            A[i, i + 1] = -1
15    return A
16
17 # b
18 def constant_vector(n = degree):
19     np.random.seed(23) # 23は適当な数字
20     return np.random.uniform(0.0, 100.0, [n, 1])
```

1.1.2 実装

逆行列は一度求めればその後掛け算するだけで使えるため、計算してしまっています

1.1.2.1 CG法

```
In [4]: 1 def cg(A, b = constant_vector()):
2     xs = np.array([b])
3
4     r = b - np.dot(A, xs[0])
5     p = r
6
7     k = 0
8     error = tol + 1
9
10    while error > tol:
11        alpha = np.dot(r.T, p) / (p.T @ A @ p)
12        xs = np.append(xs, [xs[k] + alpha * p], axis = 0)
13        r = r - alpha * np.dot(A, p)
14        beta = -1 * (r.T @ A @ p) / (p.T @ A @ p)
15        p = r + beta * p
16        k += 1
17        error = np.linalg.norm(np.dot(A, xs[k]) - b)
18
19    print("CG法はk={}で収束しました.".format(k))
20    return xs
```

1.1.2.2 Gauss-Seidel法 (c = 2 で収束しなかったため、上限を設けました)

```
In [5]: 1 def gs(A, b = constant_vector()):
2     xs = np.array([b])
3
4     LD_inv = np.linalg.inv(np.tril(A))
5     U = np.triu(A, 1)
6
7     H = -1 * np.dot(LD_inv, U)
8     c = LD_inv @ b
9
10    k = 0
11    error = tol + 1
12
13    while error > tol:
14        xs = np.append(xs, [np.dot(H, xs[k]) + c], axis = 0)
15        k += 1
16        error = np.linalg.norm(np.dot(A, xs[k]) - b)
17        if k > max_iter:
18            print("Gauss-Seidel法は{}回の反復では収束しませんでした.".format(k))
19            return xs
20
21    print("Gauss-Seidel法はk={}で収束しました.".format(k))
22    return xs
23
```

1.1.2.3 SOR法

```
In [6]: 1 def sor(A, b = constant_vector()):
2         xs = np.array([b])
3
4         D = np.diag(np.diag(A))
5         L = np.tril(A, -1)
6         U = A - L - D
7
8         Mj = np.dot(np.linalg.inv(D), -(L + U))
9         rho_Mj = max(abs(np.linalg.eigvals(Mj)))
10        w = 2 / (1 + np.sqrt(1 - rho_Mj ** 2))
11
12        T = np.linalg.inv(D + w * L)
13        H = np.dot(T, -w * U + (1 - w) * D)
14        c = np.dot(T, w * b)
15
16        k = 0
17        error = tol + 1
18
19        while error > tol:
20            xs = np.append(xs, [np.dot(H, xs[k]) + c], axis = 0)
21            k += 1
22            error = np.linalg.norm(np.dot(A, xs[k]) - b)
23
24        print("SOR法はk={}で収束しました。".format(k))
25        return xs
```

1.1.2.4 Jacobi法

c = 2で全く収束しなかったなので、反復回数に上限を設けています。

```
In [7]: 1 def jacobi(A, b = constant_vector()):
2         xs = np.array([b])
3
4         D = np.diag(np.diag(A))
5         LU = A - D
6         D_inv = np.linalg.inv(D)
7
8         H = -1 * np.dot(D_inv, LU)
9         c = np.dot(D_inv, b)
10
11        k = 0
12        error = tol + 1
13
14        while error > tol:
15            xs = np.append(xs, [np.dot(H, xs[k]) + c], axis = 0)
16            k += 1
17            error = np.linalg.norm(np.dot(A, xs[k]) - b)
18            if k > max_iter:
19                print("Jacobi法は{}回の反復では収束しませんでした。".format(k))
20                return xs
21
22        print("Jacobi法はk={}で収束しました。".format(k))
23        return xs
```

1.1.3 検証

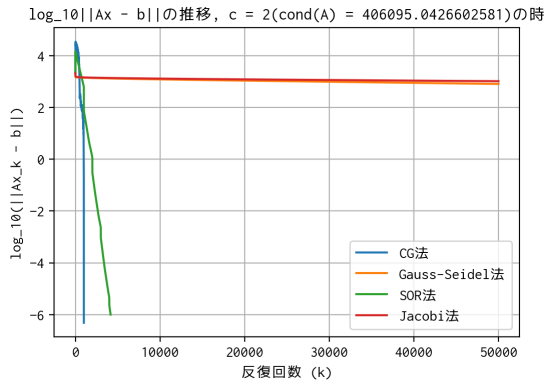
c = 2とc = 20で誤差ノルムの常用対数の推移を観察する

```
In [8]: 1 def plot_errors_by_log(c):
2         A = tridiagonal_matrix(c)
3
4         error = lambda x: np.log10(np.linalg.norm(A @ x - constant_vector()))
5
6         cg_error_transition = list(map(error, cg(A)))
7         gs_error_transition = list(map(error, gs(A)))
8         sor_error_transition = list(map(error, sor(A)))
9         jacobi_error_transition = list(map(error, jacobi(A)))
10
11        plt.plot(np.arange(len(cg_error_transition)), cg_error_transition, label='CG法')
12        plt.plot(np.arange(len(gs_error_transition)), gs_error_transition, label='Gauss-Seidel法')
13        plt.plot(np.arange(len(sor_error_transition)), sor_error_transition, label='SOR法')
14        plt.plot(np.arange(len(jacobi_error_transition)), jacobi_error_transition, label='Jacobi法')
15
16        plt.legend()
17        plt.title("log10||Ax - b||の推移, c = {}(cond(A) = {})の時".format(c, np.linalg.cond(A)))
18        plt.xlabel('反復回数 (k)')
19        plt.ylabel('log10(||Axk - b||)')
20        plt.grid()
21        plt.show()
```

(i) c = 2の時

```
In [9]: 1 plot_errors_by_log(2)
```

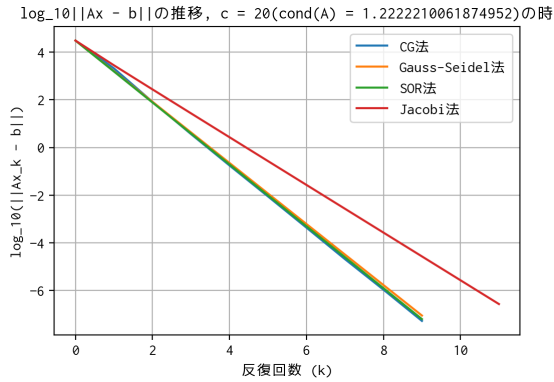
CG法はk=1000で収束しました。
Gauss-Seidel法は50001回の反復では収束しませんでした。
SOR法はk=4162で収束しました。
Jacobi法は50001回の反復では収束しませんでした。



(ii) c = 20の時

```
In [10]: 1 plot_errors_by_log(20)
```

CG法はk=9で収束しました。
Gauss-Seidel法はk=9で収束しました。
SOR法はk=9で収束しました。
Jacobi法はk=11で収束しました。



1.2 (2) 考察

1.2.0.1 c = 2 と c = 20 の比較

```
In [11]: 1 print("c = 2の時の条件数: ", np.linalg.cond(tridiagonal_matrix(2)))
2 print("c = 20の時の条件数: ", np.linalg.cond(tridiagonal_matrix(20)))
```

c = 2の時の条件数: 406095.0426602581
c = 20の時の条件数: 1.2222210061874952

c=20 では全手法で急速に||Ax - b||が0に近づく、許容誤差を下回るまでに10回程度の反復で十分だったのに対し、c = 2で||Ax - b||が50000回の反復で許容誤差を下回ったのは、CG法とSOR法だけであり、それでも反復回数は100倍以上必要だった。

上のように、c = 2 と c = 20 とで条件数が約100万倍異なるが、この現象はこの条件数の差が大きく影響を及ぼしていると考えられる。

1.2.0.2 各手法の比較

CG法とSOR法ではc = 2でも20でも収束した。

しかし、Jacobi法とGS法では、c = 2で単調に解に近づいてはいるようだが、その収束速度は極めて遅く、50000回の反復では、あまり解に近づかなかった。

また、定常反復法において、行列のスペクトル半径を観察する

```
In [12]: 1 def max_eigenvalue(A):
2     return np.sort(np.abs(np.linalg.eig(A)[0]))[-1]
3
4 def spectral_radiuses(c):
5     A = tridiagonal_matrix(c)
6
7     # GS
8     LD_inv = np.linalg.inv(np.tril(A))
9     U = np.triu(A, 1)
10    print("Gauss-Seidel法のスペクトル半径は{}".format(max_eigenvalue(-1 * LD_inv @ U)))
11
12    # SOR
13    D = np.diag(np.diag(A))
14    L = np.tril(A, -1)
15    U = A - L - D
16
17    Mj = np.dot(np.linalg.inv(D), -(L+U))
18    rho_Mj = max(abs(np.linalg.eigvals(Mj)))
19    w = 2/(1+np.sqrt(1-rho_Mj**2))
20    T = np.linalg.inv(D+w*L)
21    print("SOR法のスペクトル半径は{}".format(max_eigenvalue(np.dot(T, -w*U+(1-w)*D))))
22
23    # Jacobi
24    D = np.diag(np.diag(A))
25    LU = A - D
26    D_inv = np.linalg.inv(D)
27    print("Jacobi法のスペクトル半径は{}".format(max_eigenvalue(-1 * D_inv @ LU)))
28
```

c = 2

```
In [13]: 1 spectral_radiuses(2)
```

Gauss-Seidel法のスペクトル半径は0.9999901501375795
SOR法のスペクトル半径は0.9937427399987759
Jacobi法のスペクトル半径は0.9999950750566646

c = 20

```
In [14]: 1 spectral_radiuses(20)
```

Gauss-Seidel法のスペクトル半径は0.02009607880562554
SOR法のスペクトル半径は0.007596793131780399
Jacobi法のスペクトル半径は0.09999950750566644

いずれもスペクトル半径は1を下回っているので、収束条件を満たしている。

そのため、いずれの手法でも、解に近づいているようである。

ただ、やはりc=2のGS法とJacobi法では スペクトル半径が1に極めて近く、そのために収束が極めて遅かった。

しかし、スペクトル半径が1より小さいということは、答えに収束するということかもしれないので、(少なくとも誤差がなければ)50000回以上に反復させてみる。

ただし、上のアルゴリズムでは、あとでグラフを書くために、配列に適宜新しい解を追加していたが、それだと、反復回数が大きくなるにつれて処理速度がどんどん遅くなるので、少し改める。

```
In [15]: 1 def gs_ver2(A, b = constant_vector()):
2         LD_inv = np.linalg.inv(np.tril(A))
3         U = np.triu(A, 1)
4
5         H = -1 * np.dot(LD_inv, U)
6         c = np.dot(LD_inv, b)
7
8         k = 0
9         error = tol + 1
10        x = b
11
12        while error > tol:
13            x = np.dot(H, x) + c
14            error = np.linalg.norm(np.dot(A, x) - b)
15            k += 1
16
17        print("Gauss-Seidel法はk={}で、誤差ノルムが{}で収束しました.".format(k, error))
18        return x
```

```
In [16]: 1 def jacobi_ver2(A, b = constant_vector()):
2         D = np.diag(np.diag(A))
3         LU = A - D
4         D_inv = np.linalg.inv(D)
5
6         H = -1 * np.dot(D_inv, LU)
7         c = np.dot(D_inv, b)
8
9         k = 0
10        error = tol + 1
11        x = b
12
13        while error > tol:
14            x = np.dot(H, x) + c
15            error = np.linalg.norm(np.dot(A, x) - b)
16            k += 1
17
18        print("Jacobi法はk={}で、誤差ノルムが{}で収束しました.".format(k, error))
19        return x
```

```
In [17]: 1 gs_ans = gs_ver2(tridiagonal_matrix(2))
2         jacobi_ans = jacobi_ver2(tridiagonal_matrix(2))
```

Gauss-Seidel法はk=2133409で、誤差ノルムが9.99954566620938e-07で収束しました。
Jacobi法はk=4267675で、誤差ノルムが9.999394932987578e-07で収束しました。

やはり、GS法もJacobi法もスペクトル半径が1よりも小さいので、反復回数を増やすと答えに収束した。

(以上全ての計算で停止条件は $\|Ax - b\| < tol$ で取っている)

また、定常反復法でのスペクトル半径は

c=20のSOR < c=20のGS < c=20のJacobi < c=2のSOR < c=2のGS < c=2のJacobi (< 1)

反復回数は、

c=20のSOR = c=20のGS < c=20のJacobi < c=2のSOR < c=2のGS < c=2のJacobi

となっており、同じ順番になっていた。