

Metody numeryczne

Mateusz Kwolek

2. Dane jest macierz $A \in \mathbb{R}^{128 \times 128}$ o następującej strukturze

$$A = \begin{bmatrix} 4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 1 & 4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \dots \\ 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 \\ \dots & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ \dots & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 \\ \dots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 \end{bmatrix} \quad (2)$$

Rozwiązać równanie $Ax = e$, gdzie A jest macierzą (2), natomiast e jest wektorem, którego wszystkie składowe są równe 1, za pomocą

- (a) metody Gaussa-Seidela,
- (b) metody gradientów sprzężonych.

Algorytmy **muszą** uwzględniać strukturę macierzy (2)!


Proszę porównać graficznie tempo zbieżności tych metod, to znaczy jak zmieniają się normy $\|x_k - x_{k-1}\|$, gdzie x_k oznacza k -ty iterat. Porównać efektywną złożoność obliczeniową ze złożonością obliczeniową rozkładu Cholesky'ego dla tej macierzy.

1) Kod źródłowy

Do stworzenia programu użyłem języka Python ze względu na jego prostotę, bogactwo bibliotek oraz częste zastosowanie do wykonywania obliczeń matematycznych.

Do wykonania wykresu tempa zbieżności metod użyłem biblioteki matplotlib.

Po uruchomieniu programu zostanie wyświetlone okno z wykresem gdzie poruszając się możemy dokładnie przyjrzeć się wynikom. Te zostaną również zapisane w pliku png.



```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # Metoda Gaussa-Seidela
5  def gauss_seidel(gauss_norms, A, b, x=None, tol=1e-8):
6      matrix_size = len(b)
7      if x == None:
8          x = np.zeros(matrix_size)
9      norm = tol + 1
10     while norm > tol:
11         x_prev = x.copy()
12         for i in range(matrix_size):
13             sum = 0
14             if i >= 4:
15                 sum += A[i, i - 4] * x[i - 4]
16             if i >= 1:
17                 sum += A[i][i - 1] * x[i - 1]
18             if i < matrix_size - 4:
19                 sum += A[i][i + 4] * x[i + 4]
20             if i < matrix_size - 1:
21                 sum += A[i][i + 1] * x[i + 1]
22
23             x[i] = (b[i] - sum) / A[i, i]
24
25             norm = np.linalg.norm(x - x_prev)
26             gauss_norms.append(norm)
27
28     return x
29
30 # Funkcja do mnożenia macierzy przez wektor
31 def arr_by_vector(A, p):
32     matrix_size = len(A)
33     vector = np.zeros(matrix_size)
34     for i in range(matrix_size):
35         if i >= 4:
36             vector[i] += A[i, i - 4] * p[i - 4]
37         if i >= 1:
38             vector[i] += A[i][i - 1] * p[i - 1]
39         if i >= 0:
40             vector[i] += A[i][i] * p[i]
41         if i < matrix_size - 4:
42             vector[i] += A[i][i + 4] * p[i + 4]
43         if i < matrix_size - 1:
44             vector[i] += A[i][i + 1] * p[i + 1]
45
46     return vector

```

```

1  # Metoda gradientów sprzężonych
2  def conjugate_gradient(gauss_norms, A, b, x=None, tol=1e-8):
3      if x is None:
4          x = np.zeros(len(b))
5          r = b - np.dot(A, x)
6          p = r
7          matrix_size = len(b)
8          i = 0
9          stop = False
10         while i < matrix_size and stop == False:
11             dot_ap = arr_by_vector(A, p)
12             dot_rr = np.dot(r, r)
13             alpha = dot_rr / np.dot(p, dot_ap)
14             r_new = r - alpha * dot_ap
15             beta = np.dot(r_new, r_new) / dot_rr
16             p_new = r_new + beta * p
17             x = x + alpha * p
18             p = p_new
19             r = r_new
20
21             norm = np.linalg.norm(r_new)
22             gauss_norms.append(norm)
23             if norm < tol:
24                 stop = True
25
26         return x
27
28 # Inicjalizacja macierzy A i wektora e
29 matrixSize = 128
30 e = np.ones(matrixSize)
31 A = np.zeros((matrixSize, matrixSize))
32 for i in range(matrixSize):
33     A[i, i] = 4
34     if i < matrixSize - 1:
35         A[i, i + 1] = 1
36     if i < matrixSize - 4:
37         A[i, i + 4] = 1
38     if i >= 1:
39         A[i, i - 1] = 1
40     if i >= 4:
41         A[i, i - 4] = 1
42
43 gauss_norms = []
44 gradient_norms = []
45
46 # Wydruk rozwiązań
47 print("Gauss-Seidel solution:\n", gauss_seidel(gauss_norms, A, e))
48 print("Conjugate gradients solution:\n", conjugate_gradient(gradient_norms, A, e))
49
50 # Porównanie norm w zależności od iteracji przy użyciu matplotlib
51 x_gauss_values = np.arange(len(gauss_norms))
52 x_gradient_values = np.arange(len(gradient_norms))
53
54 plt.plot(x_gauss_values, gauss_norms, linewidth=0.5, color="#C7234F")
55 plt.scatter(x_gauss_values, gauss_norms, label='Gauss-Seidel', s=1, color="#C7234F")
56
57 plt.plot(x_gradient_values, gradient_norms, linewidth=0.5, color="#4318BA")
58 plt.scatter(x_gradient_values, gradient_norms, label='Conjugate Gradient', s=1, color="#4318BA")
59
60 plt.xlabel('Iteration')
61 plt.ylabel('Norm')
62 plt.legend()
63 plt.grid()
64 plt.savefig('wykres.png')
65 plt.show()

```

2) Wynik

Gauss-Seidel solution:

```
[0.1942768 0.1309302 0.14679491 0.16231132 0.09196262 0.13520749
0.11957885 0.11199721 0.14035394 0.11669839 0.12768499 0.12976704
0.11792602 0.12996003 0.12321575 0.12332362 0.12821491 0.12231973
0.12616837 0.12550782 0.12357099 0.12637776 0.12428321 0.12490572
0.12561555 0.12431503 0.12541531 0.12497054 0.12474606 0.12533124
0.12476995 0.12505073 0.12509842 0.12484399 0.1251229 0.12495815
0.12496554 0.12507147 0.12493632 0.12502824 0.12500981 0.12496826
0.12503211 0.12498276 0.12499873 0.12501359 0.12498422 0.12500987
0.12499891 0.12499445 0.12500755 0.12499464 0.1250013 0.12500215
0.12499649 0.12500275 0.12499912 0.12499917 0.12500157 0.12499876
0.12500033 0.12500049 0.12499928 0.12500032 0.12500033 0.12499928
0.12500048 0.12500034 0.12499875 0.12500158 0.12499917 0.12499911
0.12500276 0.12499649 0.12500215 0.1250013 0.12499463 0.12500755
0.12499445 0.12499891 0.12500987 0.12498422 0.12501359 0.12499874
0.12498276 0.12503212 0.12496825 0.1250098 0.12502825 0.12493631
0.12507147 0.12496554 0.12495814 0.1251229 0.12484399 0.12509842
0.12505074 0.12476995 0.12533124 0.12474606 0.12497054 0.12541532
0.12431503 0.12561555 0.12490572 0.12428321 0.12637776 0.12357099
0.12550782 0.12616837 0.12231973 0.12821491 0.12332362 0.12321574
0.12996003 0.11792602 0.12976704 0.127685 0.11669839 0.14035394
0.11199722 0.11957885 0.13520749 0.09196262 0.16231132 0.14679491
0.1309302 0.1942768 ]
```

Conjugate gradients solution:

```
[0.1942768 0.1309302 0.14679491 0.16231132 0.09196262 0.13520749
0.11957885 0.11199721 0.14035394 0.11669839 0.127685 0.12976704
0.11792602 0.12996003 0.12321575 0.12332362 0.12821491 0.12231973
0.12616837 0.12550782 0.12357099 0.12637776 0.12428321 0.12490572
0.12561555 0.12431504 0.12541532 0.12497054 0.12474606 0.12533124
0.12476995 0.12505074 0.12509842 0.12484399 0.1251229 0.12495815
0.12496554 0.12507147 0.12493632 0.12502824 0.12500981 0.12496826
0.12503211 0.12498276 0.12499873 0.12501358 0.12498422 0.12500987
0.12499891 0.12499445 0.12500755 0.12499463 0.1250013 0.12500215
0.1249965 0.12500275 0.12499911 0.12499917 0.12500157 0.12499875
0.12500033 0.12500048 0.12499928 0.12500033 0.12500033 0.12499928
0.12500048 0.12500033 0.12499875 0.12500157 0.12499917 0.12499911
0.12500275 0.1249965 0.12500215 0.1250013 0.12499463 0.12500755
0.12500275 0.1249965 0.12500215 0.1250013 0.12499463 0.12500755
0.12499445 0.12499891 0.12500987 0.12498422 0.12501358 0.12499873
0.12498276 0.12503211 0.12496826 0.12500981 0.12502824 0.12493632
0.12507147 0.12496554 0.12495815 0.1251229 0.12484399 0.12509842
0.12500275 0.1249965 0.12500215 0.1250013 0.12499463 0.12500755
0.12499445 0.12499891 0.12500987 0.12498422 0.12501358 0.12499873
0.12498276 0.12503211 0.12496826 0.12500981 0.12502824 0.12493632
0.12507147 0.12496554 0.12495815 0.1251229 0.12484399 0.12509842
0.12505074 0.12476995 0.12533124 0.12474606 0.12497054 0.12541532
0.12431504 0.12561555 0.12490572 0.12428321 0.12637776 0.12357099
0.12550782 0.12616837 0.12231973 0.12821491 0.12332362 0.12321575
0.12996003 0.11792602 0.12976704 0.127685 0.11669839 0.14035394
0.11199721 0.11957885 0.13520749 0.09196262 0.16231132 0.14679491
0.1309302 0.1942768 ]
```

3) Wykres

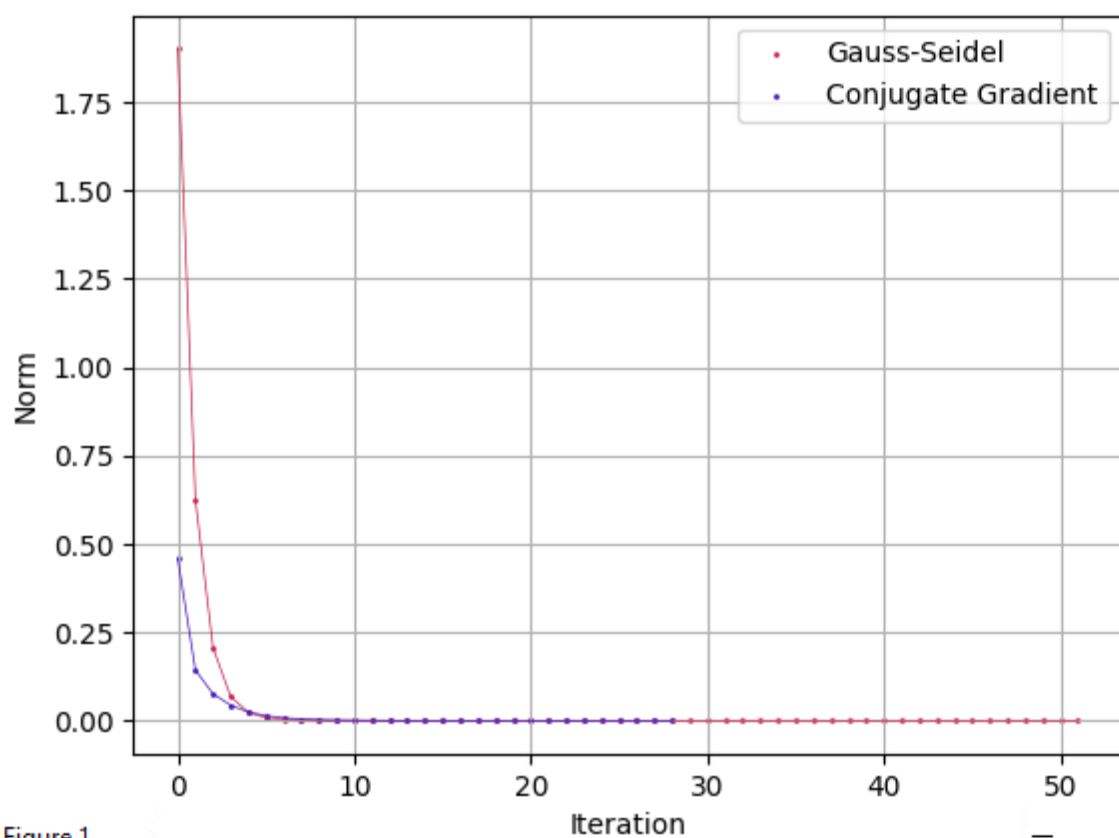


Figure 1

