|  |  |  |  |
| --- | --- | --- | --- |
| MO\_02 | Romaniak Hubert | Informatyka niestacjonarna II rok | Semestr letni 2023/24 |

# Zadanie 1

## Dekompozycja LU, eliminacja w przód i podstawianie wstecz w języku Python, używając biblioteki *numpy*

1. def lower\_upper\_decomposition(A):

2. n = A.shape[0]

3. a = A.copy()

4. for k in range(n - 1):

5. akk = a[k][k]

6. for i in range(k + 1, n):

7. aux = a[i][k] / akk if akk else 0

8. for j in range(k + 1, n):

9. a[i][j] -= a[k][j] \* aux

10. a[i][k] = aux

11. U = np.triu(a)

12. L = a - U

13. return L, U

14.

15. def eliminate\_forward(L, B):

16. n = L.shape[0]

17. b = B.copy()

18. for k in range(n - 1):

19. for i in range(k + 1, n):

20. b[i] -= b[k] \* L[i][k]

21. return b

22.

23. def substitute\_backward(U, Y):

24. n = U.shape[0]

25. y = Y.copy()

26. y[n-1] /= U[n-1][n-1]

27. for i in range(n-2, -1, -1):

28. s = 0

29. for j in range(i+1, n):

30. s += U[i][j] \* y[j]

31. y[i] -= s

32. y[i] /= U[i][i]

33. return y

## Układ równań do rozwiązania

Szukane:

## Rozwiązanie i jego dokładność dla każdego elementu

## Macierze L i U

U

Po pomnożeniu macierzy otrzymujemy w rozwiązaniu macierz .

# Zadanie 2

## Metody iteracyjne Jacobiego i Gaussa-Seidela w języku Python, używając biblioteki *numpy*

1. def jacobi(A, B):

2. D = np.diag(np.diag(A))

3. D\_inv = nla.inv(D)

4. M = (- D\_inv) @ (A - D)

5. print(f'\t{M = }')

6. print(f'\t||M|| = {norm\_inf(M):.6f}')

7. C = D\_inv @ B

8. x = np.zeros(B.size)

9. errors = []

10. while True:

11. new\_x = M @ x + C

12. errors.append(error := norm\_inf(x - new\_x))

13. if error <= 1e-7:

14. return new\_x, np.array(errors)

15. x = new\_x

16.

17. def gauss\_seidel(A, B):

18. L = np.tril(A) - np.diag(np.diag(A))

19. DU\_inv = nla.inv(A - L)

20. M = (- DU\_inv) @ L

21. print(f'\t{M = }')

22. print(f'\t||M|| = {norm\_inf(M):.6f}')

23. C = DU\_inv @ B

24. x = np.zeros(B.size)

25. errors = []

26. while True:

27. new\_x = M @ x + C

28. errors.append(error := norm\_inf(x - new\_x))

29. if error <= 1e-7:

30. return new\_x, np.array(errors)

31. x = new\_x

32.

33. def norm\_inf(a):

34. return np.abs(a).sum(axis=len(a.shape)-1).max()

## Układ równań do rozwiązania

Szukane:

Warunek przerwania:

## Rozwiązanie i jego dokładność dla każdego elementu

### Metoda Jacobiego

### Metoda Gaussa-Seidela

## Wykresy zbieżności metod

Obraz zawierający linia, Wykres, diagram, tekst

Opis wygenerowany automatycznie

## Rząd zbieżności metod

## Macierze M oraz ich normy zgodne

### Metoda Jacobiego

### Metoda Gaussa-Seidela

# Appendix

## ex\_1.py

1. import numpy as np

2.

3. np.set\_printoptions(precision=6, floatmode='fixed')

4.

5. def print\_array(name, array):

6. array\_string = str(array).replace('\n', '\n' + ' ' \* (len(name) + 3))

7. print(name, '=', array\_string)

8.

9. def lower\_upper\_decomposition(A):

10. n = A.shape[0]

11. a = A.copy()

12. for k in range(n - 1):

13. akk = a[k][k]

14. for i in range(k + 1, n):

15. aux = a[i][k] / akk if akk else 0

16. for j in range(k + 1, n):

17. a[i][j] -= a[k][j] \* aux

18. a[i][k] = aux

19. U = np.triu(a)

20. L = a - U

21. return L, U

22.

23. def eliminate\_forward(L, B):

24. n = L.shape[0]

25. b = B.copy()

26. for k in range(n - 1):

27. for i in range(k + 1, n):

28. b[i] -= b[k] \* L[i][k]

29. return b

30.

31. def substitute\_backward(U, Y):

32. n = U.shape[0]

33. y = Y.copy()

34. y[n-1] /= U[n-1][n-1]

35. for i in range(n-2, -1, -1):

36. s = 0

37. for j in range(i+1, n):

38. s += U[i][j] \* y[j]

39. y[i] -= s

40. y[i] /= U[i][i]

41. return y

42.

43. if \_\_name\_\_ == '\_\_main\_\_':

44. data = np.loadtxt('data.txt', dtype=np.float64)

45. A = data[:-1]

46. B = data[-1]

47.

48. L, U = lower\_upper\_decomposition(A)

49. L += np.eye(A.shape[0])

50.

51. print\_array('L', L)

52. print\_array('U', U)

53. print(f'{np.allclose(A, L @ U, atol=1e-6) = }')

54.

55. y = eliminate\_forward(L, B)

56. x = substitute\_backward(U, y)

57. print(f'solution = {x}')

58.

59. B\_check = A @ x

60. print\_array('B\_check', B\_check)

61. print\_array('B\_orgin', B)

62.

63. error = B - B\_check

64. print(f'error = {error}')

## ex\_2.py

1. import matplotlib.pyplot as plt

2. import numpy as np

3. import numpy.linalg as nla

4.

5. np.set\_printoptions(precision=6, floatmode='fixed')

6.

7. def print\_array(name, array):

8. array\_string = str(array).replace('\n', '\n' + ' ' \* (len(name) + 3))

9. print(name, '=', array\_string)

10.

11. def jacobi(A, B):

12. D = np.diag(np.diag(A))

13. D\_inv = nla.inv(D)

14. M = (- D\_inv) @ (A - D)

15. print\_array(' M', M)

16. print(f'\t||M|| = {norm\_inf(M):.6f}')

17. C = D\_inv @ B

18. x = np.zeros(B.size)

19. errors = []

20. while True:

21. new\_x = M @ x + C

22. errors.append(error := norm\_inf(x - new\_x))

23. if error <= 1e-7:

24. return new\_x, np.array(errors)

25. x = new\_x

26.

27. def gauss\_seidel(A, B):

28. L = np.tril(A) - np.diag(np.diag(A))

29. DU\_inv = nla.inv(A - L)

30. M = (- DU\_inv) @ L

31. print\_array(' M', M)

32. print(f'\t||M|| = {norm\_inf(M):.6f}')

33. C = DU\_inv @ B

34. x = np.zeros(B.size)

35. errors = []

36. while True:

37. new\_x = M @ x + C

38. errors.append(error := norm\_inf(x - new\_x))

39. if error <= 1e-7:

40. return new\_x, np.array(errors)

41. x = new\_x

42.

43. def norm\_inf(a):

44. return np.abs(a).sum(axis=len(a.shape)-1).max()

45.

46. if \_\_name\_\_ == '\_\_main\_\_':

47. data = np.loadtxt('data.txt', dtype=np.float64)

48. A = data[:-1]

49. B = data[-1]

50.

51. print('JACOBI')

52. x, jacobi\_errors = jacobi(A, B)

53. jacobi\_errors = np.log10(jacobi\_errors)

54. errors\_slope = (jacobi\_errors[-1] - jacobi\_errors[0]) / jacobi\_errors.size

55. print(f'\torder of convergence = {errors\_slope:.6f}')

56. B\_check = A @ x

57. B\_error = B - B\_check

58. print\_array('\tx', x)

59. print\_array('\tB check', B\_check)

60. print\_array('\tB orgin', B)

61. print\_array('\tB error', B\_error)

62. print()

63.

64. print('GAUSS-SEIDEL')

65. x, gauss\_seidel\_errors = gauss\_seidel(A, B)

66. gauss\_seidel\_errors = np.log10(gauss\_seidel\_errors)

67. errors\_slope = (gauss\_seidel\_errors[-1] - gauss\_seidel\_errors[0]) / gauss\_seidel\_errors.size

68. print(f'\torder of convergence = {errors\_slope:.6f}')

69. B\_check = A @ x

70. B\_error = B - B\_check

71. print\_array('\tx', x)

72. print\_array('\tB check', B\_check)

73. print\_array('\tB orgin', B)

74. print\_array('\tB error', B\_error)

75. print()

76.

77. fig = plt.figure()

78. ax = fig.add\_subplot()

79. ax.set\_xlabel('iterations')

80. ax.set\_ylabel('log\_10(||x\_(n+1) - x\_n||\_inf)')

81. ax.set\_xlim(0, 16)

82. ax.set\_ylim(-8, 0)

83. ax.plot(range(jacobi\_errors.size), jacobi\_errors, label = 'Jacobi')

84. ax.plot(range(gauss\_seidel\_errors.size), gauss\_seidel\_errors, label = 'Gauss-Seidel')

85. ax.grid()

86. ax.legend()

87. ax.set\_aspect('equal')

88. fig.show()