MATEMATYKA

Metody Numeryczne

Projekt 1.

Wykonał:

Oleg Łyżwiński 305158

Warszawa 2024

Poniżej przedstawiono rozwiązanie zadań A, B pierwszego projektu z przedmiotu Metody Numeryczne. Do rozwiązań wykorzystano język Python. Zastosowano następujące biblioteki: numpy, matplotlib oraz math. W załączniku dołączono plik env_Metody_Numeryczne.yml pozwalający na stworzenie środowiska zawierającego wszystkie wyżej wymienione pakiety przy pomocy Anakondy.

Zadanie A 1.

```python
# Zamiana liczb z systemu dziesiętnego na szestnastkowy
def dec_to_hex(dec_number):
    while dec_number > 0:
        quotient = dec_number % 16
        if quotient == 10:
            hex_number.append("A")
        elif quotient == 11:
            hex_number.append("B")
        elif quotient == 12:
            hex_number.append("C")
        elif quotient == 13:
            hex_number.append("D")
        elif quotient == 14:
            hex_number.append("E")
        elif quotient == 15:
            hex_number.append("F")
        else:
            hex_number.append(quotient)
        dec_number = dec_number // 16
    hex_number.reverse()
    return hex_number
```

```python
# Zamiana liczb z systemu dziesiętnego na ósemkowy
def dec_to_oct(dec_number):
    while dec_number > 0:
        quotient = dec_number % 8
        oct_number.append(quotient)
        dec_number //= 8
    oct_number.reverse()
    return oct_number
```

Zamiana liczby 16 zapisanej w systemie dziesiętnym na szesnastkowy i ósemkowy:

```
Wprowadzona liczba w systemie dziesietnym:                  16
Na podstaei algorytmu (liczba w systemie szesnastkowym):    10
Na podstaei funkcji Python hex:                             0x10
Na podstaei algorytmu (liczba w systemie osemkowym):        20
Na podstaei funkcji Python oct:                             0o20
```

Zamiana liczby 157 zapisanej w systemie dziesiętnym na szesnastkowy i ósemkowy:

```
Wprowadzona liczba w systemie dziesietnym:                  157
Na podstaei algorytmu (liczba w systemie szesnastkowym):    9D
Na podstaei funkcji Python hex:                             0x9d
Na podstaei algorytmu (liczba w systemie osemkowym):        235
Na podstaei funkcji Python oct:                             0o235
```

Zamiana liczby 2044 zapisanej w systemie dziesiętnym na szesnastkowy i ósemkowy:

```
Wprowadzona liczba w systemie dziesietnym:                  2044
Na podstaei algorytmu (liczba w systemie szesnastkowym):    7FC
Na podstaei funkcji Python hex:                             0x7fc
Na podstaei algorytmu (liczba w systemie osemkowym):        3774
Na podstaei funkcji Python oct:                             0o3774
```

Zadanie A 2

```python
eps=1
# Dzielimy do czasu gdy suma eps i 1 daje 1 (eps jest poniżej precyzji)
while eps+1 != 1:
    eps /= 2
    i=eps+1
    if(i == 1):
        # Cofnięcie jednego kroku pętli ponieważ wartość jest poniżej dokładności
        eps = eps * 2
        break
```

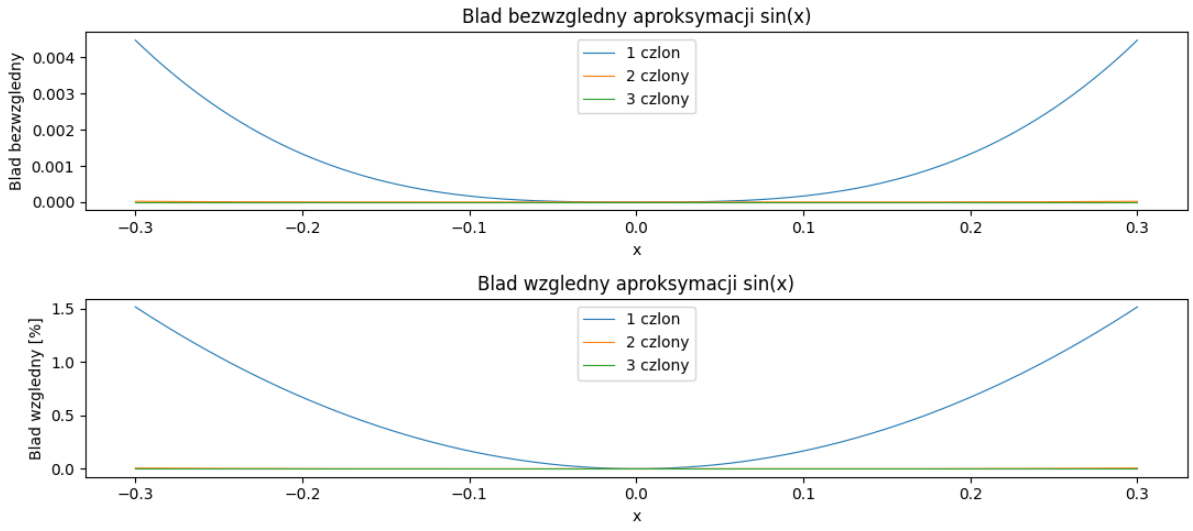
Poniżej przedstawiono wartość precyzji mojego urządzenia:

```
Wartosc epsilon na podstawie funkcji (sys.float_info.epsilon):    2.220446049250313e-16
Wartosc epsilon na algorytmu:                                     2.220446049250313e-16
```

W języku Python nie występuje pętla do while, konieczne więc jest cofnięcie jednego kroku pętli (pomnożenie wyniku przez 2).

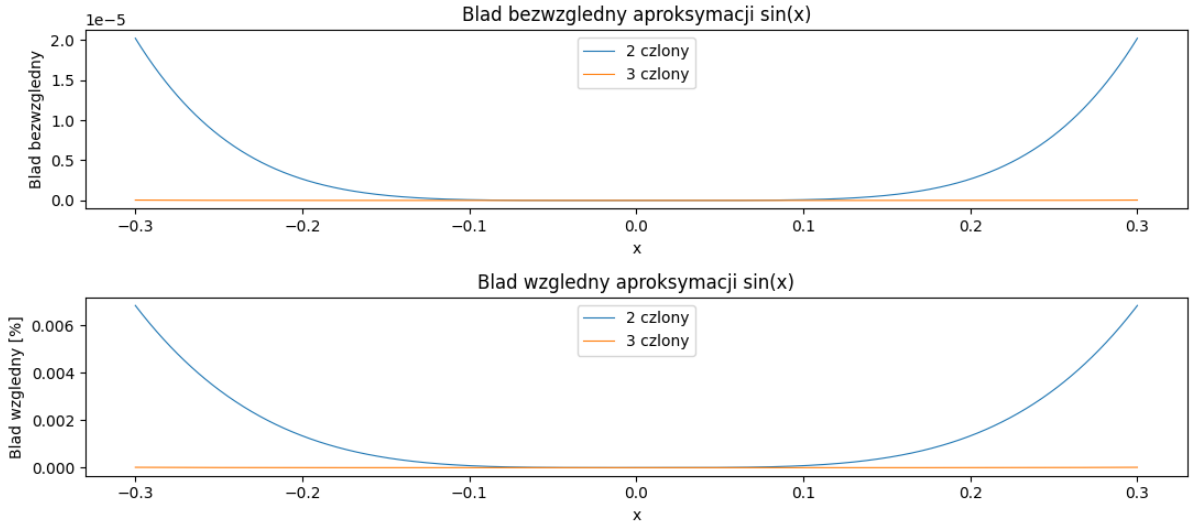
Zadanie A 3

Wykreślona charakterystyka dla jednego, dwóch i trzech członów szeregu Maclaurina:



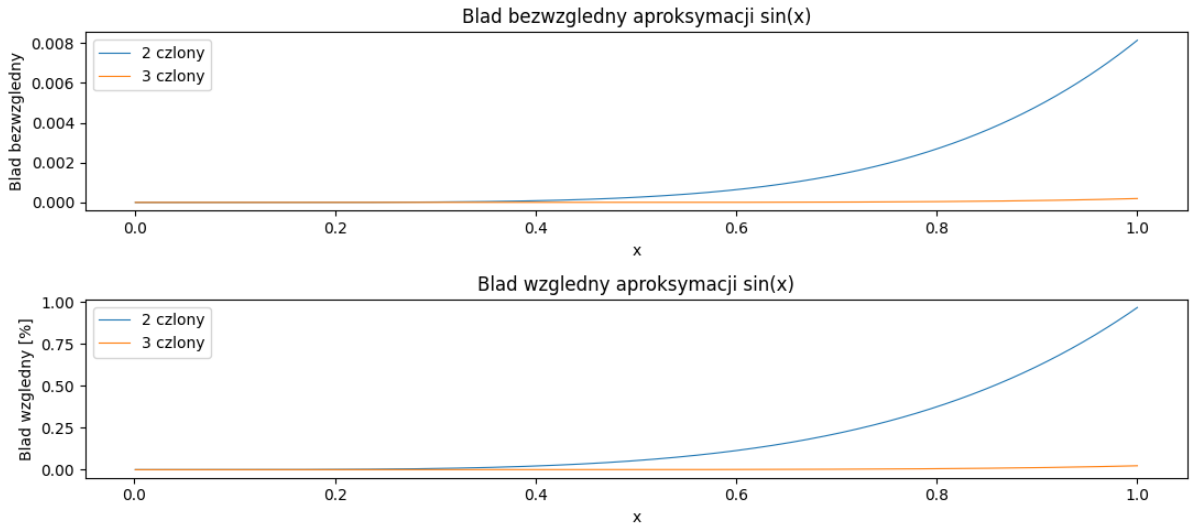
Na podstawie powyższej charakterystyki możemy zaobserwować, że dla x == 0.3 wartość błędu względnego wynosi 1,5%. Może być to nieakceptowalne przy wymaganej większej precyzji obliczeń.

Poniżej przedstawiono wyniki jedynie dla dwóch i trzech członów szeregu Maclaurina:



W tym przypadku dla x=0.3 błąd względny dla dwóch członów nie przekracza 0.007 %.

Poniżej przedstawiono wyniki jedynie dla dwóch i trzech członów szeregu Maclaurina w zakresie od 0 do 1:



Dla przybliżenia 3 wyrazami szeregu Maclaurina błąd względny dla x=1 wynosi jedynie 1.5 e-5, w przypadku przybliżenia 2 wyrazami wynosi już 1%.

Zad B 4

```python
# Sprawdza czy wszystkie minory macierzy są niezerowe
# Zwraca True jeśli wszystkie minory są niezerowe, False w przeciwnym razie.
def are_all_minors_non_zero(matrix):
    rows, cols = matrix.shape
    for i in range(rows):
        for j in range(cols):
            minor_matrix = np.delete(np.delete(matrix, i, axis=0), j, axis=1)
            minor_det = np.linalg.det(minor_matrix)
            if minor_det == 0:
                return False
    return True
```

```python
# Zamiana wiersza pierwszego na wiersz bez zerowych elementów
def zamien_wiersze(macierz):
    cnt = 0
    # Sprawdzenie, czy istnieje zero w pierwszym wierszu
    if 0 in macierz[0]:
        # Szukanie wierszy z zerem
        for i in range(1, len(macierz)):
            if 0 not in macierz[i]:
                # Znaleziono wiersz bez zera, zamiana miejscami
                pom = macierz[0].copy()
                macierz[0], macierz[i] = macierz[i], pom
            else:
                break

    return macierz
```

```python
# Metoda elimincaji Gaussa
def gauss(A, b):
    n = len(b)
    Ab = np.column_stack((A, b))
    x = np.zeros(n)

    # Sprawdzenie warunku podstawowego
    is_posible = are_all_minors_non_zero(A)

    if is_posible == True:
        # Ustawienie w pierwszym wierszu, wiersza bez zer
        Ab = zamien_wiersze(Ab)
        print("Macierz wejściowa:       \n", end="")
        print(Ab)
        for i in range(n):
            # Odejmowanie wierszy
            for j in range(i+1, n):
                ratio = Ab[j, i] / Ab[i, i]
                Ab[j] =  Ab[j] - ratio * Ab[i]
        print("Macierz po wyzerowaniu:  \n", end="")
        print(Ab)
        # rozwiązanie
        for i in range(n-1, -1, -1):
            x[i] = (Ab[i, -1] - np.dot(Ab[i, i+1:n], x[i+1:n])) / Ab[i, i]

        return x, True
    else:
        return x, False
```

Przed wykonaniem obliczeń sprawdzano czy wszystkie minory macierzy są nie zerowe oraz ustawiono w pierwszym wierszu, wiersz nie zawierający zer.

Uzyskane wyniki:

a)

```
Wskaźniki uwarunkowania:
Norma kolumnowa:              61.99999999999998
Norma spektralna:             51.03040383918669
Norma nieskończoności:        61.99999999999998

Macierz wejściowa:
[[ 1.   2.  10. ]
 [ 1.1  2.  10.4]]
Macierz po wyzerowaniu:
[[ 1.                  2.                   10.                ]
 [ 0.                 -0.20000000000000018 -0.5999999999999996 ]]
Rozwiązanie układu równań:           [4.000000000000009  2.9999999999999956]
Rozwiązanie funkcją np.linalg.solve: [4.000000000000003  2.9999999999999987]
```

Dla przykładu a wyniki różniły się dopiero na 15 miejscu po przecinku:

```
Roznica wynosi                 :  [-6.2172489379008766e-15  3.1086244689504383e-15]
```

b)

```
Wskaźniki uwarunkowania:
Norma kolumnowa:              47999995.993290655
Norma spektralna:             39999993.97257294
Norma nieskończoności:        47999995.993290655

Macierz wejściowa:
[[2.       5.999999 8.000001]
 [2.       6.       8.      ]]
Macierz po wyzerowaniu:
[[ 2.000000000000000e+00  5.999999000000000e+00  8.000000999999999e+00]
 [ 0.000000000000000e+00  1.000000000139778e-06 -9.999999992515995e-07]]
Rozwiązanie układu równań:           [ 6.999999997335465  -0.9999999991118216]
Rozwiązanie funkcją np.linalg.solve: [ 6.999999997335465  -0.9999999991118216]
```

Dla przykładu b metodą Gaussa uzyskano identyczny wynik jak w przypadku wyniku uzyskanego funkcją np.linalg.solve

```
Roznica wynosi                 :  [0. 0.]
```

c)

Pierwotny wynik wynikający z zaokrąglenia wartości w macierzy do liczb całkowitych:

```
Wskaźniki uwarunkowania:
Norma kolumnowa:              22.399999999999995
Norma spektralna:             15.012396717902485
Norma nieskończoności:        24.999999999999993

Macierz wejściowa:
[[ 5  3  4 18]
 [ 3  0  1  7]
 [ 6  3  6 27]]
Macierz po wyzerowaniu:
[[ 5  3  4 18]
 [ 0 -1 -1 -3]
 [ 0  0  1  5]]
Rozwiązanie układu równań:            [ 0.8 -2.   5. ]
Rozwiązanie funkcją np.linalg.solve:  [ 1.0000000000000007 -0.9999999999999977  3.9999999999999982]
```

Po przekształceniu zmiennych wejściowych na typ zmiennoprzecinkowy uzyskane wyniki był zbliżone do tych wyznaczonych metodą np.linalg.solve

```
Wskaźniki uwarunkowania:
Norma kolumnowa:              22.399999999999995
Norma spektralna:             15.012396717902485
Norma nieskończoności:        24.999999999999993

Macierz wejściowa:
[[ 5.  3.  4. 18.]
 [ 3.  0.  1.  7.]
 [ 6.  3.  6. 27.]]
Macierz po wyzerowaniu:
[[ 5.                3.                 4.
  18.               ]
 [ 0.               -1.7999999999999998 -1.4
  -3.799999999999999 ]
 [ 0.                0.                 1.6666666666666665
   6.666666666666668 ]]
Rozwiązanie układu równań:            [ 1.                -1.0000000000000009  4.000000000000001 ]
Rozwiązanie funkcją np.linalg.solve:  [ 1.0000000000000007 -0.9999999999999977  3.9999999999999982]
```

Dla przykładu c wyniki różniły się dopiero na 19 miejscu po przecinku:

```
Roznica wynosi                :  [ 6.6613381477509392e-16  3.2196467714129540e-15 -2.6645352591003757e-15]
```

Różnice rzędu 1 e-15 oraz 1 e-16 wynikają z zaokrągleń zastosowanych przez algorytm w takcie obliczeń. Są to wartości bardzo bliskie wartości epsilon wyznaczonej w zadaniu A 2.

Zadanie B 6

```python
#Sprawdzenie czy macierz jest dodatnio określona
def is_positive_definite(matrix):
    n = matrix.shape[0]
    for i in range(1, n + 1):
        minor_matrix = matrix[:i, :i]
        if np.linalg.det(minor_matrix) <= 0:
            return False
    return True
```

```python
def cholesky_banachiewicz(A, b):
    n = len(A)
    L = np.zeros((n, n))

    # Wyznaczenie macierzy L
    for i in range(n):
        for j in range(i+1):
            tmp_sum = sum(L[i][k] * L[j][k] for k in range(j))
            if i == j:
                L[i][j] = np.sqrt(A[i][i] - tmp_sum)
            else:
                L[i][j] = (1.0 / L[j][j] * (A[i][j] - tmp_sum))

    # Rozwiązywanie L*y = b
    y = np.zeros(n)
    for i in range(n):
        y[i] = (b[i] - sum(L[i][j] * y[j] for j in range(i))) / L[i][i]

    # Rozwiązywanie L^T*x = y
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (y[i] - sum(L[j][i] * x[j] for j in range(i+1, n))) / L[i][i]

    return x
```

```
Macierz A jest dodatnio okreslona.
Rozwiazanie ukladu rownan:              [-2.  0.  1.]
Rozwiązanie funkcją np.linalg.solve:  [-2.  0.  1.]
```

Zadanie B 7

```python
def jacobi(A, b, x, epsilon, max_iter):
    n = len(A)
    error_tab = []

    for k in range(max_iter):
        x_act = np.zeros(n)
        for i in range(n):
            x_act[i] = (b[i] - np.dot(A[i,:], x) + A[i,i]*x[i]) / A[i,i]

        error = np.linalg.norm(x_act - x)
        error_tab.append(error)

        if error < epsilon:
            break

        x = x_act.copy()

    return x, error_tab
```

```python
def gauss_seidel(A, b, x, epsilon, max_iter):
    n = len(A)
    error_tab = []

    for k in range(max_iter):
        for i in range(n):
            x[i] = (b[i] - np.dot(A[i,:i], x[:i]) - np.dot(A[i,i+1:], x[i+1:])) / A[i,i]

        error = np.linalg.norm(np.dot(A, x) - b)
        error_tab.append(error)

        if error < epsilon:
            break

    return x, error_tab
```

```python
# Warunek dominanty przekątnej
def diagonal(matrix):

    n = len(matrix)
    result = []

    for i in range(n):
        diagonal = abs(matrix[i, i])
        row_sum = np.sum(np.abs(matrix[i, :])) - diagonal

        result.append(diagonal > row_sum)

    return result
```

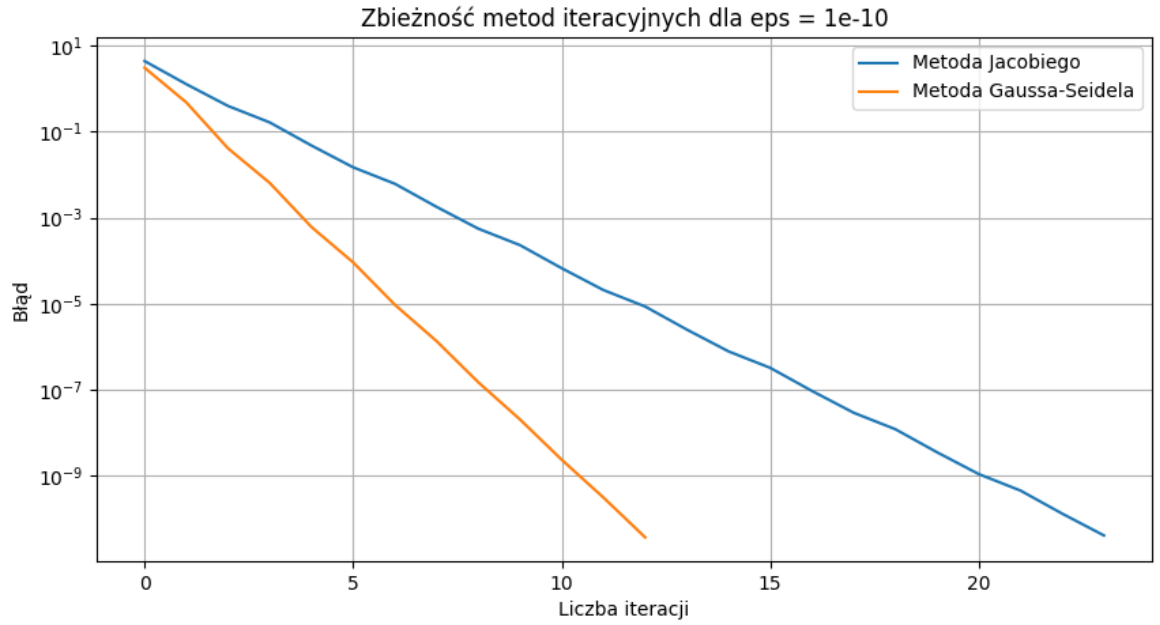
W pierwszej kolejności badano czy macierz spełnia warunek konieczny do zastosowania metody Jacobiego oraz Gauss'a Seidel'a, a więc warunku dominanty przekątnej lub wierszowej. Automatycznie implikuje spełnienie warunku zbieżności Jacobiego, a co za tym idzie pozwala na zastosowanie metody Jacobiego oraz Gauss'a Seidel'a

Poniżej przedstawiono charakterystyki błędu w skali logarytmicznej od liczby iteracji dla rozwiązań równań wyżej wymienionymi metodami.

a)

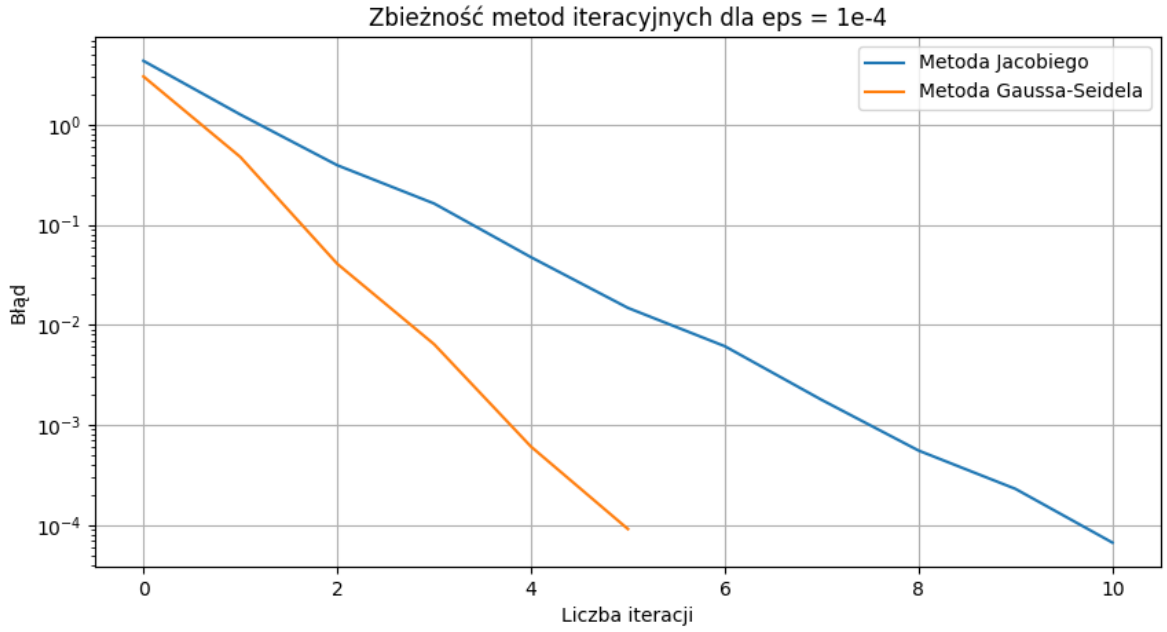
Eps=1e-10

```
Warunek dostateczny zbieznosci jest spelniony.
Rozwiązanie równań metodą Jacobiego:       [1.9999999999641522 3.9999999999869646 3.0000000000182494]
Rozwiązanie równań metodą Gaussa-Seidela:  [1.9999999999907154 3.999999999992724  2.9999999999977414]
Rozwiązanie funkcją np.linalg.solve:       [2. 4. 3.]
```



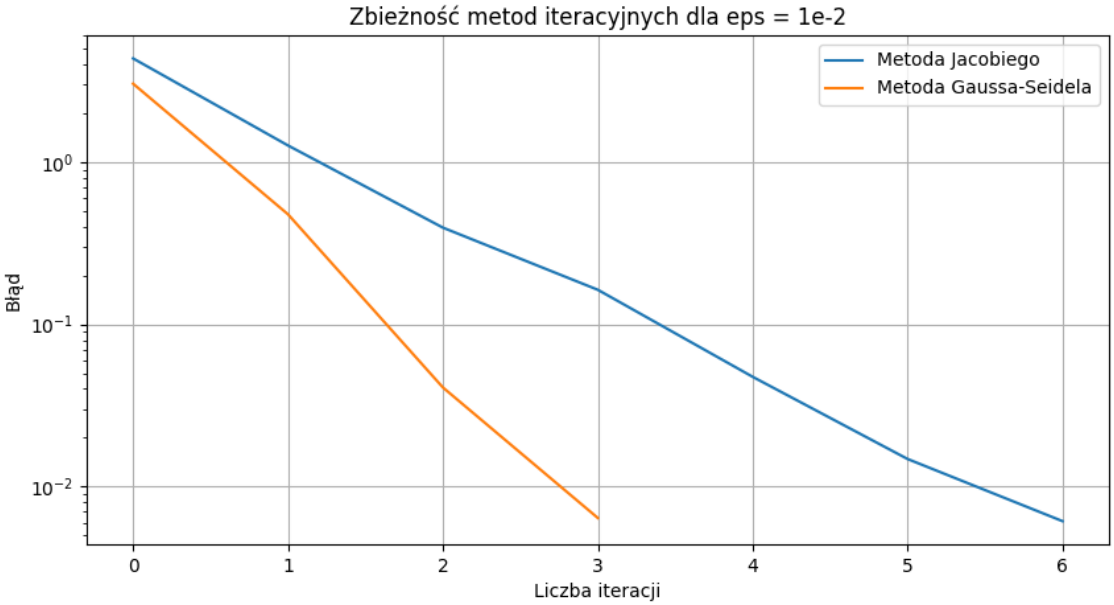
Eps=1e-4

```
Warunek dostateczny zbieznosci jest spelniony.
Rozwiązanie równań metodą Jacobiego:       [1.99998681640625  3.999927490234375 3.                ]
Rozwiązanie równań metodą Gaussa-Seidela:  [1.99997548828125   3.9999847412109375 2.9999932470703126]
Rozwiązanie funkcją np.linalg.solve:       [2. 4. 3.]
```



Eps=1e-2

```
Warunek dostateczny zbieznosci jest spelniony.
Rozwiązanie równań metodą Jacobiego:       [1.9971875          3.9943750000000002 2.9957812500000003]
Rozwiązanie równań metodą Gaussa-Seidela:  [1.99828125   3.9990234375 2.9995078125]
Rozwiązanie funkcją np.linalg.solve:       [2. 4. 3.]
```

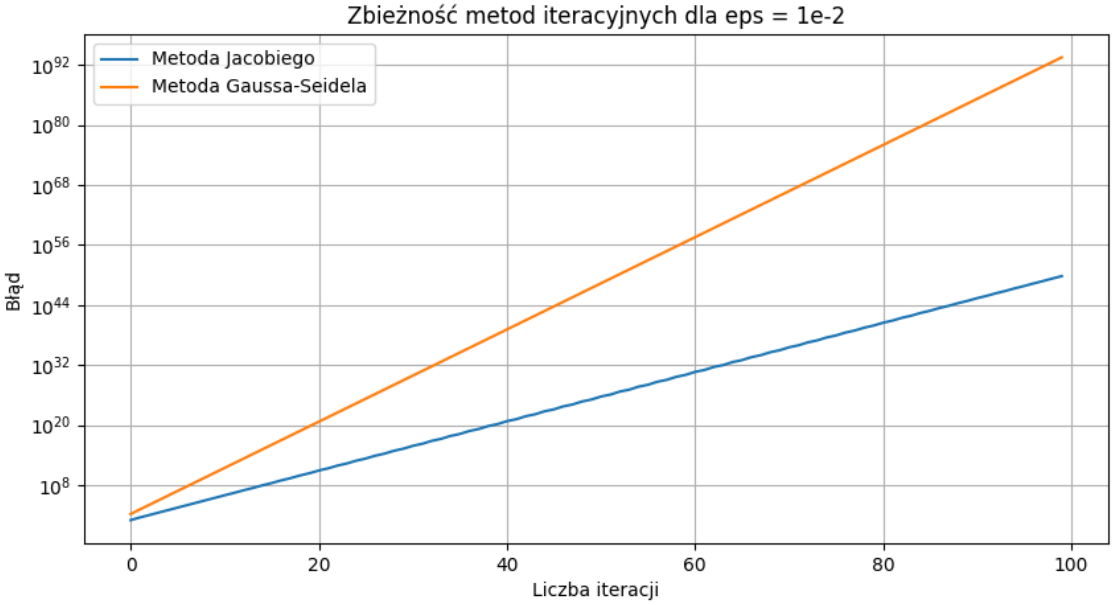


b)

W przypadku macierzy z przykładu b podstawowy warunek nie został spełniony co powoduje rozbieganie błędu w nieskończoność.

```
Warunek dostateczny zbieznosci nie jest spelniony.
```

Na wykresie widzimy, że błąd rozbiega w nieskończoność (w tym przypadku wykonano 100 iteracji)



Na podstawie powyższych przykładów możemy wnioskować, że algorytm Gauss'a Seidel'a potrzebuje mniej iteracji do osiągnięcia pożądanej dokładności rozwiązania niż metoda Jacobiego. Ponadto na podstawie wyżej zamieszczonych charakterystyk możemy zauważyć, że program pozwala na osiągnięcie pożądanej dokładności rozwiązania.