Rozwiązywanie układów równań NxN

Instrukcja: Na zajęciach należy wykonać poniższe zadania, a następnie sporządzić sprawozdanie zawierające odpowiedzi (w postaci kodu) z komentarzami w środowisku Jupyter Notebook i umieścić je na platformie e-learningowej.

Cel zajęć: Celem zajęć jest zapoznanie się z numerycznymi metodami rozwiązywania układów równań liniowych. To podstawowe zadanie algebry liniowej które macierzowo możemy zapisać jako:

$$\mathbf{Ax} = \mathbf{b}$$

gdzie $\mathbf{A}$ - macierz współczynników, $\mathbf{x}$ - wektor zmiennych a $\mathbf{b}$ - wektor wyników prawej strony równania.

Do oceny jakości rozwiązania będziemy wykorzystywać residuum (ang. *residual*) $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$

Do wykonania zadań niezbędne będą ponisze funkcje:

In [1]:

```python
import numpy as np
import scipy
import matplotlib
import matplotlib.pyplot as plt
import pickle

from typing import Union, List, Tuple

def random_matrix_Ab(m:int):
    if not isinstance(m, int):
        print("Wartość m nie jest liczbą")
        return None

    try:
        A = np.random.randint(1, 10, size=(m, m))
        B = np.random.randint(1, 10, size=m)
    except Exception as e:
        print(f'Błąd {e}')
        return None

    return A, B


def residual_norm(A:np.ndarray,x:np.ndarray, b:np.ndarray):
    if A.shape[0] != A.shape[1] or A.shape[0] != x.shape[0] or A.shape[0] != b.shape[
        print("Niepoprawne wymiary")
        return None

    # Obliczenie residuum: r = b - Ax
    residuum = b - np.dot(A, x)
    # Obliczenie normy residuum
    norma_residuum = np.linalg.norm(residuum)
    return norma_residuum

def log_sing_value(n: int, min_order: Union[int, float], max_order: Union[int, float]
    if n <= 0:
        print("Błąd: Rozmiar wektora wartości singularnych musi być większy od 0.")
        return None

    # Generowanie równomiernego rozkładu logarytmicznego w zakresie od 10^min_order do
    log_values = np.logspace(min_order, max_order, num=n, base=10.0)

    return log_values


def order_sing_value(n: int, order: Union[int, float] = 2, site: str = 'gre') -> np.n
    if n <= 0:
        print("Błąd: Rozmiar wektora wartości singularnych musi być większy od 0.")
        return None

    singular_values = np.random.rand(n) * 10

    if site == 'low':
        singular_values[-1] *= 10**(-order)
    elif site == 'gre':
        singular_values[0] *= 10**order
    else:
        print("Błąd: Nieprawidłowa wartość dla parametru 'site'. Wybierz 'low' lub 'gr
        return None

    return singular_values


def create_matrix_from_A(A:np.ndarray, sing_value:np.ndarray):
    """Funkcja generująca rozkład SVD dla macierzy A i zwracająca otworzenie macierzy
```

```
        Parameters:
        A(np.ndarray): rozmiarz macierzy A (m,m)
        sing_value(np.ndarray): wektor wartości singularnych (m,)


        Results:
        np.ndarray: macierz (m,m) utworzoną na podstawie rozkładu SVD zadanej maci
    return None
```

Zadanie 1

1. Zaimplementuj funkcje *random_matrix_Ab* według opisu powyzej generującą macierz kwadratową $\mathbf{A}$ i wektor $\mathbf{b}$ o zadanych wymiarach odpowiednio $m \times m$, $m \times 1$ i o wartościach losowych. W tym celu skorzystaj z funkcji *randint (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.random.randint.html)*. W razie podania nieprawidłowej wartości m funkcja ma zwrócić wartość None.
2. Wygeneruj takie macierze dla $m = 10,\ 20,\ 50,\ 100,\ 1000$.
3. Zaimplementuj normę residual_norm zgodnie z opisem z main.py (używając *norm (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.norm.html?highlight=norm#numpy.linalg.norm)*)

In [2]:

```python
def random_matrix_Ab(m:int):
    if not isinstance(m, int):
        print("Wartość m nie jest liczbą")
        return None

    try:
        A = np.random.randint(1, 10, size=(m, m))
        B = np.random.randint(1, 10, size=m)
    except Exception as e:
        print(f'Błąd {e}')
        return None

    return A, B

def residual_norm(A:np.ndarray,x:np.ndarray, b:np.ndarray):
    if A.shape[0] != A.shape[1] or A.shape[0] != x.shape[0] or A.shape[0] != b.shape[
        print("Niepoprawne wymiary")
        return None

    # Obliczenie residuum: r = b - Ax
    residuum = b - np.dot(A, x)
    # Obliczenie normy residuum
    norma_residuum = np.linalg.norm(residuum)
    return norma_residuum


wartości_m = [10, 20, 50, 100, 1000]
zbior_macierzy = []
zbior_wektorow = []
for m in wartości_m:
    macierz_A, wektor_B = random_matrix_Ab(m)
    print(f'Macierz o wymiarze {m}x{m}')
    print(macierz_A)
    zbior_macierzy.append(macierz_A)
    zbior_wektorow.append(wektor_B)
```

```
Macierz o wymiarze 10x10
[[4 7 2 1 3 3 9 5 2 6]
 [1 1 4 7 5 7 2 7 8 7]
 [7 9 8 4 4 7 1 1 4 9]
 [8 5 8 2 5 6 1 8 9 8]
 [3 4 4 7 9 6 6 1 1 5]
 [5 6 7 3 5 5 6 8 9 2]
 [9 8 4 9 4 9 8 9 2 4]
 [5 7 6 3 9 9 9 2 6 8]
 [9 5 3 5 4 8 2 2 2 5]
 [7 5 6 4 7 1 2 8 8 3]]
Macierz o wymiarze 20x20
[[1 9 9 8 8 4 5 8 4 9 5 3 1 6 9 9 7 4 1 5]
 [7 6 4 8 8 9 2 6 5 7 3 7 5 7 3 9 7 4 1 5]
 [1 4 8 9 6 2 1 4 6 5 3 1 7 9 7 7 7 4 2 3]
 [8 9 8 4 4 4 5 1 5 3 1 6 3 8 9 6 4 6 1 8]
 [7 9 1 7 9 4 5 3 8 6 7 4 1 4 2 5 5 6 4 6]
 [5 3 6 5 5 2 3 5 4 1 4 4 3 2 5 7 1 1 5 6]
 [2 9 8 9 3 7 1 6 7 6 5 1 9 2 5 1 7 4 9 5]
 [7 7 6 5 2 3 4 3 4 3 9 4 3 4 2 1 8 7 7 6]
 [8 7 6 4 7 5 4 5 5 2 4 9 1 1 3 7 1 2 7 9]
 [4 5 9 3 6 9 3 3 7 4 4 3 2 5 6 8 6 9 7 9]
 [9 9 1 9 6 5 1 3 8 9 3 2 1 9 1 2 3 4 8 6]
 [4 1 1 4 8 3 2 6 5 6 6 7 2 1 8 9 4 5 9 5]
 [2 3 4 6 8 2 5 6 7 6 4 1 1 2 3 3 1 3 4 4]
 [6 4 8 3 4 5 5 3 7 7 2 4 5 8 5 9 4 5 5 6]
 [1 5 1 2 4 8 1 9 3 8 9 8 6 4 1 7 2 4 7 3]
 [6 4 8 2 7 6 1 6 4 3 1 1 2 2 2 4 6 1 8 5]
 [3 1 1 9 8 2 2 3 9 9 2 6 3 6 8 1 4 7 6 7]
 [9 4 2 2 5 1 1 3 2 7 3 7 9 8 5 6 8 2 9 3]
 [3 1 4 4 5 6 3 8 3 2 9 8 6 8 9 8 3 7 1 1]
 [7 2 2 7 6 7 2 7 4 8 1 5 7 4 5 9 7 7 7 4]]
Macierz o wymiarze 50x50
[[5 1 8 ... 3 2 1]
 [6 8 3 ... 9 5 1]
 [9 3 3 ... 6 2 3]
 ...
 [2 2 4 ... 1 5 7]
 [6 2 9 ... 8 7 2]
 [7 5 5 ... 2 2 3]]
Macierz o wymiarze 100x100
[[4 7 1 ... 8 9 2]
 [6 2 9 ... 2 5 5]
 [2 2 7 ... 4 1 4]
 ...
 [2 5 4 ... 7 4 2]
 [5 7 1 ... 4 2 4]
 [4 2 7 ... 5 1 9]]
Macierz o wymiarze 1000x1000
[[2 8 2 ... 2 2 6]
 [5 5 4 ... 6 8 5]
 [3 1 9 ... 3 7 3]
 ...
 [6 6 7 ... 2 1 6]
 [4 4 5 ... 3 1 1]
 [6 4 5 ... 6 2 2]]
```

Zadanie 2

1. Dla macierzy i wektorów wygenerowanych w poprzednim zadaniu znajdź rozwiązanie układu równań $\mathbf{Ax} = \mathbf{b}$ używając funkcji *solve (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.solve.html?highlight=solve#numpy.linalg.solve)*.
2. Sprawdź dokładność otrzymanego rozwiązania (oblicz normę residuum).

3. Określ uwarunkowanie macierzy $\mathbf{A}$ przy pomocy funkcji *cond (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.cond.html?highlight=cond#numpy.linalg.cond)*.
4. Odpowiedź na pytanie czy zakres wartości oraz wymiary macierzy mają wpływ na jakość otrzymanych wyników?
5. Zbadać czas wykonania obliczeń przy pomocy funkcji *timeint*

In [3]:

```
#macierz 10x10
A_10 = zbior_macierzy[0]
x_10 = zbior_wektorow[0]
b_10 = np.dot(A_10, x_10) # Aby uzyskać b, mnożymy macierz A przez wektor x

# Znalezienie rozwiązania układu równań Ax = b
rozwiazanie_10 = np.linalg.solve(A_10, b_10)

# Obliczenie normy residuum
norma_residuum_10 = residual_norm(A_10, rozwiazanie_10, b_10)

# Obliczenie uwarunkowania macierzy A
uwarunkowanie_10 = np.linalg.cond(A_10)

print(f'Rozwiązanie dla macierzy 10x10: {rozwiazanie_10} \nNorma residuum dla macierzy

#macierz 20x20
A_20 = zbior_macierzy[1]
x_20 = zbior_wektorow[1]
b_20 = np.dot(A_20, x_20) # Aby uzyskać b, mnożymy macierz A przez wektor x

# Znalezienie rozwiązania układu równań Ax = b
rozwiazanie_20 = np.linalg.solve(A_20, b_20)

# Obliczenie normy residuum
norma_residuum_20 = residual_norm(A_20, rozwiazanie_20, b_20)

# Obliczenie uwarunkowania macierzy A
uwarunkowanie_20 = np.linalg.cond(A_20)

print(f'Rozwiązanie dla macierzy 20x20: {rozwiazanie_20} \nNorma residuum dla macierzy

#macierz 50x50
A_50 = zbior_macierzy[2]
x_50 = zbior_wektorow[2]
b_50 = np.dot(A_50, x_50) # Aby uzyskać b, mnożymy macierz A przez wektor x

# Znalezienie rozwiązania układu równań Ax = b
rozwiazanie_50 = np.linalg.solve(A_50, b_50)

# Obliczenie normy residuum
norma_residuum_50 = residual_norm(A_50, rozwiazanie_50, b_50)

# Obliczenie uwarunkowania macierzy A
uwarunkowanie_50 = np.linalg.cond(A_50)

print(f'Rozwiązanie dla macierzy 50x50: {rozwiazanie_50} \nNorma residuum dla macierzy

#macierz 100x100
A_100 = zbior_macierzy[3]
x_100 = zbior_wektorow[3]
b_100 = np.dot(A_100, x_100) # Aby uzyskać b, mnożymy macierz A przez wektor x

# Znalezienie rozwiązania układu równań Ax = b
rozwiazanie_100 = np.linalg.solve(A_100, b_100)

# Obliczenie normy residuum
norma_residuum_100 = residual_norm(A_100, rozwiazanie_100, b_100)

# Obliczenie uwarunkowania macierzy A
uwarunkowanie_100 = np.linalg.cond(A_100)

print(f'Rozwiązanie dla macierzy 100x100: {rozwiazanie_100} \nNorma residuum dla macie
```

```python
#Macierz 1000x1000
A_1000 = zbior_macierzy[4]
x_1000 = zbior_wektorow[4]
b_1000 = np.dot(A_1000, x_1000) # Aby uzyskać b, mnożymy macierz A przez wektor x

# Znalezienie rozwiązania układu równań Ax = b
rozwiazanie_1000 = np.linalg.solve(A_1000, b_1000)

# Obliczenie normy residuum
norma_residuum_1000 = residual_norm(A_1000, rozwiazanie_1000, b_1000)

# Obliczenie uwarunkowania macierzy A
uwarunkowanie_1000 = np.linalg.cond(A_1000)

print(f'Rozwiązanie dla macierzy 1000x1000: {rozwiazanie_1000} \nNorma residuum dla ma
```

```
Rozwiązanie dla macierzy 10x10: [7. 4. 7. 8. 8. 3. 9. 4. 6. 9.]
Norma residuum dla macierzy 10x10 1.6077746776921858e-13
Uwarunkowanie Macierzy A: 369.84066603047904

Rozwiązanie dla macierzy 20x20: [3. 6. 8. 5. 8. 2. 3. 4. 3. 2. 1. 1. 8. 2. 5. 3. 4.
3. 3. 3.]
Norma residuum dla macierzy 20x20 4.0194366942304643e-13
Uwarunkowanie Macierzy A: 411.0581250131867

Rozwiązanie dla macierzy 50x50: [1. 3. 2. 2. 2. 1. 6. 3. 9. 1. 6. 7. 7. 2. 6. 8. 6.
7. 8. 8. 6. 2. 8. 5.
 6. 2. 7. 7. 8. 4. 4. 2. 6. 8. 8. 2. 2. 2. 8. 8. 4. 6. 1. 8. 7. 8. 3. 3.
 2. 7.]
Norma residuum dla macierzy 50x50 1.7648943294557312e-12
Uwarunkowanie Macierzy A: 783.076543620589

Rozwiązanie dla macierzy 100x100: [4. 9. 6. 9. 8. 8. 4. 6. 2. 7. 1. 7. 8. 8. 6. 6.
7. 1. 5. 2. 7. 6. 2. 4.
 8. 3. 7. 2. 2. 2. 4. 7. 6. 8. 1. 6. 7. 6. 7. 3. 5. 4. 7. 3. 5. 5. 2. 6.
 5. 7. 9. 7. 3. 9. 2. 7. 8. 4. 3. 1. 6. 5. 7. 2. 8. 1. 6. 9. 9. 5. 5. 1.
 7. 9. 7. 7. 4. 1. 1. 2. 9. 8. 5. 3. 7. 7. 7. 9. 1. 4. 1. 9. 4. 5. 4. 9.
 5. 3. 8. 7.]
Norma residuum dla macierzy 100x100 9.900550620421764e-12
Uwarunkowanie Macierzy A: 5301.144221104383

Rozwiązanie dla macierzy 1000x1000: [8. 2. 9. 4. 7. 5. 6. 6. 4. 8. 7. 4. 4. 2. 4. 3.
2. 6. 9. 4. 9. 2. 7. 1.
 8. 7. 6. 4. 7. 2. 6. 5. 4. 6. 5. 6. 7. 9. 2. 6. 5. 1. 5. 3. 9. 7. 6. 2.
 1. 8. 1. 5. 5. 2. 7. 7. 9. 7. 8. 1. 2. 1. 4. 2. 8. 3. 5. 8. 6. 1. 8. 5.
 2. 3. 9. 7. 2. 8. 9. 5. 7. 4. 4. 6. 3. 5. 6. 9. 8. 6. 3. 2. 9. 6. 1. 7.
 7. 9. 8. 6. 3. 6. 4. 2. 4. 4. 4. 3. 1. 4. 2. 6. 2. 2. 2. 6. 8. 7. 7. 6.
 3. 5. 2. 2. 7. 7. 9. 2. 7. 8. 8. 7. 8. 7. 1. 4. 2. 2. 2. 4. 4. 1. 7. 9.
 8. 3. 6. 8. 3. 8. 9. 8. 3. 4. 2. 3. 7. 4. 6. 9. 2. 8. 1. 6. 5. 7. 5. 1.
 4. 5. 2. 2. 3. 4. 1. 4. 2. 7. 9. 8. 6. 2. 7. 2. 9. 9. 4. 8. 4. 7. 6. 2.
 7. 9. 9. 5. 1. 8. 6. 4. 1. 1. 6. 7. 9. 4. 2. 5. 9. 1. 2. 1. 5. 9. 5. 5.
 5. 3. 5. 9. 1. 7. 2. 2. 4. 2. 7. 1. 8. 4. 3. 4. 8. 8. 7. 1. 5. 7. 5. 3.
 1. 3. 7. 2. 2. 1. 1. 2. 8. 9. 5. 7. 5. 4. 7. 4. 3. 2. 9. 1. 8. 5. 5. 1.
 6. 3. 9. 6. 1. 6. 7. 5. 2. 3. 2. 6. 8. 3. 6. 9. 3. 8. 6. 1. 6. 6. 4. 5.
 8. 5. 9. 5. 2. 5. 6. 3. 7. 8. 4. 9. 2. 4. 5. 3. 6. 1. 8. 9. 6. 5. 2. 8.
 1. 9. 1. 4. 3. 8. 1. 4. 4. 9. 6. 8. 1. 1. 6. 4. 6. 2. 5. 7. 7. 1. 1. 9.
 1. 2. 8. 3. 7. 2. 8. 3. 8. 5. 5. 1. 1. 8. 6. 8. 3. 5. 2. 5. 9. 1. 7. 1.
 1. 1. 1. 4. 2. 6. 8. 5. 4. 6. 5. 5. 7. 9. 8. 7. 5. 1. 3. 4. 6. 3. 8. 9.
 1. 7. 4. 9. 9. 1. 2. 2. 6. 7. 3. 2. 2. 8. 2. 8. 1. 9. 5. 3. 2. 4. 5. 8.
 3. 8. 6. 8. 5. 6. 6. 5. 3. 5. 5. 2. 4. 6. 5. 5. 7. 9. 4. 5. 2. 4. 5. 5.
 1. 6. 2. 6. 2. 9. 8. 1. 8. 1. 3. 4. 7. 2. 4. 8. 6. 4. 5. 5. 4. 3. 5. 6.
 2. 2. 9. 8. 8. 4. 4. 6. 4. 9. 7. 1. 3. 6. 7. 2. 1. 7. 8. 3. 5. 1. 5. 7.
 7. 6. 8. 2. 4. 9. 7. 4. 9. 3. 6. 8. 4. 5. 2. 5. 2. 5. 5. 9. 7. 2. 9. 1.
 6. 2. 3. 5. 3. 7. 8. 4. 4. 5. 9. 1. 4. 4. 4. 2. 2. 5. 2. 2. 7. 1. 5. 7.
 4. 7. 2. 4. 6. 3. 6. 2. 1. 4. 5. 7. 1. 8. 9. 7. 3. 7. 6. 2. 8. 7. 2. 3.
 6. 6. 3. 4. 2. 8. 5. 2. 7. 8. 3. 2. 2. 2. 2. 7. 8. 5. 5. 7. 8. 4. 1. 5.
 2. 9. 6. 4. 9. 7. 6. 5. 9. 9. 6. 8. 9. 5. 9. 2. 4. 2. 2. 6. 9. 1. 3. 3.
 3. 3. 4. 8. 4. 7. 8. 8. 7. 5. 5. 4. 4. 4. 4. 7. 7. 9. 3. 7. 5. 9. 5. 6.
 4. 9. 4. 3. 3. 5. 7. 3. 7. 7. 4. 8. 7. 7. 1. 8. 5. 7. 8. 8. 9. 4. 5. 7.
 2. 8. 3. 9. 5. 3. 2. 8. 9. 5. 8. 9. 1. 6. 5. 2. 3. 7. 7. 6. 3. 2. 7. 2.
 1. 7. 6. 1. 3. 5. 3. 7. 4. 6. 1. 9. 6. 2. 8. 4. 1. 8. 1. 3. 1. 2. 5. 6.
 5. 7. 3. 2. 7. 7. 3. 6. 8. 8. 1. 2. 6. 4. 5. 4. 5. 9. 2. 1. 4. 1. 3. 3.
 5. 5. 5. 8. 1. 4. 5. 9. 9. 3. 1. 1. 2. 1. 6. 1. 2. 1. 5. 4. 6. 7. 9. 8.
 5. 1. 3. 7. 9. 2. 2. 9. 4. 1. 1. 3. 8. 4. 2. 5. 2. 7. 8. 7. 7. 3. 9. 2.
 2. 6. 4. 1. 4. 6. 1. 8. 5. 3. 4. 9. 1. 1. 5. 9. 9. 5. 8. 9. 9. 9. 4. 2.
 7. 9. 6. 4. 4. 8. 2. 2. 4. 7. 8. 5. 5. 8. 7. 2. 3. 6. 9. 3. 9. 4. 1. 4.
 7. 6. 9. 6. 6. 2. 6. 3. 4. 2. 2. 8. 3. 2. 8. 9. 3. 3. 5. 6. 2. 6. 9. 3.
 1. 8. 2. 2. 4. 3. 7. 4. 5. 6. 8. 5. 2. 9. 1. 5. 9. 1. 4. 9. 7. 6. 2. 3.
 2. 5. 2. 4. 6. 6. 6. 5. 3. 9. 2. 2. 3. 1. 7. 9. 1. 4. 9. 3. 7. 3. 2. 2.
 6. 3. 5. 2. 8. 5. 9. 9. 6. 1. 3. 8. 9. 7. 7. 7. 9. 5. 6. 4. 4. 7. 1. 5.
 9. 1. 9. 7. 8. 2. 4. 2. 4. 8. 9. 9. 5. 6. 6. 6. 4. 7. 1. 4. 6. 3. 8. 5.
```

```
 3. 1. 9. 5. 4. 4. 3. 6. 8. 1. 1. 8. 7. 8. 5. 9. 4. 8. 6. 8. 6. 1. 7. 1.
 1. 9. 4. 7. 3. 4. 6. 6. 4. 3. 2. 4. 2. 3. 7. 9. 5. 6. 4. 4. 4. 4. 6. 6.
 2. 6. 6. 3. 1. 6. 5. 9. 6. 9. 5. 1. 1. 2. 4. 1.]
Norma residuum dla macierzy 1000x1000 1.1136375459180173e-09
Uwarunkowanie Macierzy A: 257120.99638496275
```

Odpowiedź na pytanie 5. Macierze o wysokim uwarunkowaniu są bardziej podatne na błędy numeryczne, co może wpływać na precyzję otrzymanych wyników. Im wyższe uwarunkowanie, tym trudniej jest zachować precyzję numeryczną. W tym kodzie

Zadanie 3

Rozkład dowolnej macierzy metodą *dekompozycji na wartości singularne (https://pl.wikipedia.org/wiki/Rozk%C5%82ad_wed%C5%82ug_warto%C5%9Bci_osobliwych)* można w Pythonie przeprowadzić przy pomocy funkcji *svd (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.svd.html)*. Rozkład dla przykładowej macierzy obrazuje kod:

In [15]:
```
import numpy as np
import numpy.linalg as nplin

A = np.array([[1,2,3],[1,2,3],[1,2,3]])

# Użycie rozkładu SVD na macierzy A
U,S,V = nplin.svd(A)

print(S)
# Odtworzenie macierzy A przy pomocy metody SVD
A2 = np.dot(U * S, V)

print("Macierz A:\n {0}".format(A))
print("Macierz odtworzona z SVD:\n {0}".format(A2))
```

```
[6.4807407 0.        0.       ]
Macierz A:
 [[1 2 3]
 [1 2 3]
 [1 2 3]]
Macierz odtworzona z SVD:
 [[1. 2. 3.]
 [1. 2. 3.]
 [1. 2. 3.]]
```

Wykonaj następujące kroki:

1. Zdefiniuj funkcję inicjalizujące wektory *wartości singularnych* w następujący sposób:

- wektor nierosnących wartości singularnych w postaci wektora przestrzeni logarytmicznej, np:

In [16]:
```
S1 = np.logspace(100, 1, num=3)
print(S1)
```

```
[1.00000000e+100 3.16227766e+050 1.00000000e+001]
```

In [17]:
```
def log_sing_value(n: int, min_order: Union[int, float], max_order: Union[int, float
    if n <= 0:
        print("Błąd: Rozmiar wektora wartości singularnych musi być większy od 0.")
        return None

    # Generowanie równomiernego rozkładu logarytmicznego w zakresie od 10^min_order do

    log_values = np.logspace(min_order, max_order, num=n, base=10.0)

    return log_values

S1 = log_sing_value(3, 1, 100)
print("S1:", S1)
```

```
S1: [1.00000000e+001 3.16227766e+050 1.00000000e+100]
```

- wektor nierosnących wartości singularnych, gdzie jedna wartość jest znacznie większa od pozostałych, np.:

In [18]:
```
S2 = np.logspace(100, 1, num=3)
S2[0] = S2[0]+100
```

In [19]:
```
def order_sing_value(n: int, order: Union[int, float] = 2, site: str = 'gre') -> np.n
    if n <= 0:
        print("Błąd: Rozmiar wektora wartości singularnych musi być większy od 0.")
        return None

    singular_values = np.random.rand(n) * 10

    if site == 'low':
        singular_values[-1] *= 10**(-order)
    elif site == 'gre':
        singular_values[0] *= 10**order
    else:
        print("Błąd: Nieprawidłowa wartość dla parametru 'site'. Wybierz 'low' lub 'gr
        return None

    return singular_values

S2 = order_sing_value(3, 100, 'gre')
print("S2:", S2)
```

```
S2: [1.55004838e+100 9.68818772e+000 6.91015296e+000]
```

- wektor nierosnących wartości, gdzie jedna wartość jest znacznie mniejsza od pozostałych.

In [20]:
```
S3 = np.logspace(100, 1, num=3)
S3[-1] = S3[0]-100
```

In [21]:
```
S3 = order_sing_value(3,100, 'low')
print("S3:", S3)
```

```
S3: [1.34513633e+000 2.39637110e+000 2.60964340e-100]
```

W celu inicjalizacji takich wektorów zaimplementuje funkcje:
- *log_sing_value* zgodnie z opisem w main.py i użyciu funkcji [logspace (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.logspace.html?highlight=logspace#numpy.logspace)](https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.logspace.html?highlight=logspace#numpy.logspace) - *order_sing_value* zgodnie z opisem w main.py

2. Zdefiniuj funkcję *create_matrix_from_A* z pliku main, która dla zadanej macierzy A z *zadania 1* i wektorów warości singularnych z punktu 1. tego zadania będzie zwracać odtworzoną macierz z podmienionym wektorem warości singularnych przy pomocy metody SVD, jak w przykładzie:

In [22]:
```
A = np.array([[1,2,3],[1,2,3],[1,2,3]])
U,S,V = nplin.svd(A)

A1 = np.dot(U * S1, V)
A2 = np.dot(U * S2, V)
A3 = np.dot(U * S3, V)
print('Macierz pierwotna:')
print(A)

print('Macierz na podstawie wartości S1:')
print(A1)

print('Macierz na podstawie wartości S2:')
print(A2)

print('Macierz na podstawie wartości S3:')
print(A3)
```

```
Macierz pierwotna:
[[1 2 3]
 [1 2 3]
 [1 2 3]]
Macierz na podstawie wartości S1:
[[-5.56348640e+99  8.55920985e+98  1.28388148e+99]
 [ 7.59986376e+99 -1.16920981e+99 -1.75381471e+99]
 [-2.03637736e+99  3.13288824e+98  4.69933236e+98]]
Macierz na podstawie wartości S2:
[[2.39177658e+99 4.78355315e+99 7.17532973e+99]
 [2.39177658e+99 4.78355315e+99 7.17532973e+99]
 [2.39177658e+99 4.78355315e+99 7.17532973e+99]]
Macierz na podstawie wartości S3:
[[ 0.20755904  1.56629752 -0.14477583]
 [ 0.20755904  0.836479    0.34176985]
 [ 0.20755904 -1.15742227  1.67103737]]
```

In [23]:
```python
import numpy.linalg as nplin
def create_matrix_from_A(A: np.ndarray, sing_value: np.ndarray) -> np.ndarray:

    if not isinstance(A, np.ndarray) or not isinstance(sing_value, np.ndarray):
        print("Błąd: A i sing_value muszą być macierzami numpy.")
        return None

    try:
        U, S, V = nplin.svd(A)
        A_reconstructed = np.dot(U * sing_value, V)
    except Exception as e:
        print(f'Błąd: {e}')
        return None

    return A_reconstructed

# Przykład użycia:
A = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
U, S, V = nplin.svd(A)

A1 = create_matrix_from_A(A, S1)
A2 = create_matrix_from_A(A, S2)
A3 = create_matrix_from_A(A, S3)
print('Macierz pierwotna:')
print(A)

print('Macierz na podstawie wartości S1:')
print(A1)

print('Macierz na podstawie wartości S2:')
print(A2)

print('Macierz na podstawie wartości S3:')
print(A3)
```

```
Macierz pierwotna:
[[1 2 3]
 [1 2 3]
 [1 2 3]]
Macierz na podstawie wartości S1:
[[-5.56348640e+99  8.55920985e+98  1.28388148e+99]
 [ 7.59986376e+99 -1.16920981e+99 -1.75381471e+99]
 [-2.03637736e+99  3.13288824e+98  4.69933236e+98]]
Macierz na podstawie wartości S2:
[[2.39177658e+99 4.78355315e+99 7.17532973e+99]
 [2.39177658e+99 4.78355315e+99 7.17532973e+99]
 [2.39177658e+99 4.78355315e+99 7.17532973e+99]]
Macierz na podstawie wartości S3:
[[ 0.20755904  1.56629752 -0.14477583]
 [ 0.20755904  0.836479    0.34176985]
 [ 0.20755904 -1.15742227  1.67103737]]
```

3. Dla otrzymanych macierzy oblicz wartości współczynnika uwarunkowania.
4. Odpowiedz na pytanie: czy konieczne jest wyliczanie macierzy aby to zrobić?
5. Dla każdego m sporządź wykres normy residuów rozwiązań i funkcji uwarunkowania macierzy.

In [24]:
```
#Podpunkt 3
# Obliczanie współczynnika uwarunkowania dla macierzy A1
cond_A1 = nplin.cond(A1)

# Obliczanie współczynnika uwarunkowania dla macierzy A2
cond_A2 = nplin.cond(A2)

# Obliczanie współczynnika uwarunkowania dla macierzy A3
cond_A3 = nplin.cond(A3)

print('Współczynnik uwarunkowania dla macierzy A1:', cond_A1)
print('Współczynnik uwarunkowania dla macierzy A2:', cond_A2)
print('Współczynnik uwarunkowania dla macierzy A3:', cond_A3)
```

```
Współczynnik uwarunkowania dla macierzy A1: inf
Współczynnik uwarunkowania dla macierzy A2: inf
Współczynnik uwarunkowania dla macierzy A3: 3.4257890086309904e+16
```

Wartości "inf" wskazują na to, że współczynniki uwarunkowania dla macierzy A2 i A3 są nieskończone. To oznacza, że jedna lub więcej wartości singularnych tych macierzy jest równe zero.

Współczynnik uwarunkowania dla macierzy jest zdefiniowany jako stosunek największej wartości singularnej do najmniejszej wartości singularnej. Kiedy najmniejsza wartość singularna wynosi zero, współczynnik uwarunkowania staje się nieskończony, ponieważ nie możemy dzielić przez zero.

Podpunkt 4

Współczynnik uwarunkowania macierzy można obliczyć bez konieczności wyznaczania całej macierzy. Dla macierzy A, współczynnik uwarunkowania cond(A) można obliczyć jako iloraz największej wartości singularnej przez najmniejszą wartość singularną

In [25]:
```
#Podpunkt 5

#wartości_m mamy z zadania 1

# Listy przechowujące wyniki
residual_norms = []
condition_numbers = []

# Iteracja po różnych rozmiarach macierzy m
for m in wartości_m:
    # Generowanie macierzy A i wektora b
    A = np.random.rand(m, m)
    b = np.random.rand(m)

    # Rozwiązanie układu równań
    x = nplin.solve(A, b)

    # Obliczenie residuum
    residual_norm = nplin.norm(b - np.dot(A, x))
    residual_norms.append(residual_norm)

    # Obliczenie współczynnika uwarunkowania
    condition_number = nplin.cond(A)
    condition_numbers.append(condition_number)

# Sporządzenie wykresów
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.plot(wartości_m, residual_norms, marker='o')
plt.title('Normy Residuów')
plt.xlabel('Rozmiar macierzy (m)')
plt.ylabel('Norma residuum')

plt.subplot(1, 2, 2)
plt.plot(wartości_m, condition_numbers, marker='o')
plt.title('Funkcje Uwarunkowania')
plt.xlabel('Rozmiar macierzy (m)')
plt.ylabel('Współczynnik uwarunkowania')

plt.tight_layout()
plt.show()
```