Sprawozdanie nr. 3 - Metody numeryczne i optymailzacja

Jakub Andryszczak 259519,
Jakub Żak 244255,
Maciej Cierpisz 249163

Spis treści

1 Zadanie nr. 1

Znajdź „ręcznie” przybliżone rozwiązania w sensie kryterium najmniejszych kwadratów dla poniższych układów równań sprzecznych:

$$(a)\begin{cases}3x_1 - x_2 = 4\\ x_1 + 2x_2 = 0\\ 2x_1 + x_2 = 1\end{cases} \quad (b)\begin{cases}3x_1 + x_2 + x_3 = 6\\ 2x_1 + 3x_2 - x_3 = 1\\ 2x_1 - x_2 + x_3 = 0\\ 3x_1 - 3x_2 + 3x_3 = 8\end{cases} . \tag{1}$$

Przykład a:

$$AX = B \tag{2}$$

$$A = \begin{bmatrix}3 & -1\\ 1 & 2\\ 2 & 1\end{bmatrix} \quad B = \begin{bmatrix}4\\ 0\\ 1\end{bmatrix} \tag{3}$$

Obliczamy przybliżone wartości wektora X metodą najmniejszych kwadratów:

$$X = (A^T A)^{-1} * (A^T B) \tag{4}$$

$$(A^T A)^{-1} = \begin{bmatrix}\frac{6}{83} & -\frac{1}{83}\\ -\frac{1}{83} & \frac{14}{83}\end{bmatrix} \tag{5}$$

$$(A^T B) = \begin{bmatrix}14\\ -3\end{bmatrix} \tag{6}$$

$$X = \begin{bmatrix}\frac{6}{83} & -\frac{1}{83}\\ -\frac{1}{83} & \frac{14}{83}\end{bmatrix} * \begin{bmatrix}14\\ -3\end{bmatrix} = \begin{bmatrix}\frac{87}{83}\\ -\frac{56}{83}\end{bmatrix} \tag{7}$$

Przykład b:

$$AX = B \tag{8}$$

$$A = \begin{bmatrix}3 & 1 & 1\\ 2 & 3 & -1\\ 2 & -1 & 1\\ 3 & -3 & 3\end{bmatrix} \quad B = \begin{bmatrix}6\\ 1\\ 0\\ 8\end{bmatrix} \tag{9}$$

Obliczamy przybliżone wartości wkektora X metodą najmniejszych kwadratów:

$$X = (A^T A)^{-1} * (A^T B) \tag{10}$$

$$(A^T A)^{-1} = \begin{bmatrix} \frac{2}{3} & -\frac{5}{6} & -\frac{3}{2} \\ -\frac{5}{6} & \frac{7}{6} & 2 \\ -\frac{3}{2} & 2 & \frac{43}{12} \end{bmatrix} \tag{11}$$

$$(A^T B) = \begin{bmatrix} 44 \\ -15 \\ 29 \end{bmatrix} \tag{12}$$

$$X = \begin{bmatrix} \frac{2}{3} & -\frac{5}{6} & -\frac{3}{2} \\ -\frac{5}{6} & \frac{7}{6} & 2 \\ -\frac{3}{2} & 2 & \frac{43}{12} \end{bmatrix} * \begin{bmatrix} 44 \\ -15 \\ 29 \end{bmatrix} = \begin{bmatrix} -\frac{196}{71} \\ \frac{293}{71} \\ \frac{685}{71} \end{bmatrix} \tag{13}$$

2 Zadanie nr. 2

Znajdź parametry funkcji $a_0 + a_1 x^2 + a_2 sin\frac{\pi * x}{2}$, która aproksymuje poniższy zbiór punktów na płaszczyźnie w sensie kryterium najmniejszych kwadratów: (0,3), (1,0), (1,-1), (-1,2)

Poniżejimplementacja zadania w języku Python:

```python
import numpy as np

# Definicja wektorów y i x
y = np.array([3, 0, -1, 2])
x = np.array([0, 1, 1, -1])

# Transpozycja dwóch wektorów
y = y.reshape(-1, 1)
x = x.reshape(-1, 1)

# Tworzenie macierzy A
A = np.hstack((np.ones_like(x), x**2, np.sin(np.pi * x / 2)))

# Obliczenie z1
z1 = np.linalg.inv(A.T @ A) @ A.T @ y

print(z1)
```

Otrzymane wartości współczynników a:

$$\begin{cases} a_0 = 3 \\ a_1 = -2,25 \\ a_2 = -1,25 \end{cases} \tag{14}$$

3 Zadanie nr. 3

Pocisk jest wystrzeliwany z terytorium wroga, a jego położenie w locie jest obserwowane przez radarowe urządzenia śledzące w następujących pozycjach:

x_i [km]	0	250	500	750	1000
y_i [km]	0	8	15	19	20

Załóżmy, że nasze źródła wywiadowcze wskazują, że wrogie pociski są zaprogramowane do poruszania się po parabolicznym torze lotu. Oblicz jak daleko od punktu wystrzału spadnie pocisk.

Z założeń wcześniej postawionych możemy wywnioskować że wzór dla danego pocisku będzie prezentować się następująco:

$$y_i = ax_i^2 + bx_i + c \tag{15}$$

Dla podanych wartości wyliczono macierz A:

$$\begin{bmatrix} 0 & 0 & 1 \\ 62500 & 250 & 1 \\ 250000 & 500 & 1 \\ 562500 & 750 & 1 \\ 1000000 & 1000 & 1 \end{bmatrix} \tag{16}$$

Oraz wektor wynikowy:

$$\begin{bmatrix} 0 \\ 8 \\ 15 \\ 19 \\ 20 \end{bmatrix} \tag{17}$$

Następnie wykonano kod w Python, który wyliczał współczynniki aproksymacji z podanych wartości:

```
import numpy as np

# Dane z radarów
xi = np.array([0, 250, 500, 750, 1000])
yi = np.array([0, 8, 15, 19, 20])
```

```
# Tworzenie macierzy A
A = np.vstack([xi**2, xi, np.ones(len(xi))]).T
print(A)
# Rozwiązanie równań normalnych
params = np.linalg.lstsq(A, yi, rcond=None)[0]

# Współczynniki aproksymacji
a, b, c = params
```

Wyniki obliczeń prezentują się następująco

$$\begin{aligned} a &= -1.94286 * 10^{-5} \\ b &= 0.03982 \\ c &= 0.228571 \end{aligned} \tag{18}$$

Kolejno wyżej obliczone wartości podstawiono do wzoru na zasięg:

$$Z = \frac{-b - (b^2 - 4ac)^{\frac{1}{2}}}{2a} = 2044.245[km] \tag{19}$$

4 Zadanie nr. 4

Używając kryterium najmniejszych kwadratów dopasuj modele (a) i (b) do danych:

x	-5	-4	-3	-2	-1	0	1	2	3	4	5
y	2	7	9	12	13	14	14	13	10	8	4

(a)

$$y = a_0 + a_1 x, \tag{20}$$

(b)

$$y = a_0 + a_1 x + a_2 x^2. \tag{21}$$

Określ, który model lepiej dopasowuje się do danych na podstawie normy 2l błędu dopasowania:

Dla obu podpunktów wektor wynikowy wygląda tak samo

$$b = \begin{bmatrix} 2 \\ 7 \\ 9 \\ 12 \\ 13 \\ 14 \\ 14 \\ 13 \\ 10 \\ 8 \\ 4 \end{bmatrix} \tag{22}$$

Dla podpunktu a) macierz A prezentuje się następująco:

$$A = \begin{bmatrix} -5 & 1 \\ -4 & 1 \\ -3 & 1 \\ -2 & 1 \\ -1 & 1 \\ 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \end{bmatrix} \tag{23}$$

Wykonano kod w Python który oblicza współczynniki oraz wyznacza normę 2l błędu dopasowania:

```
  import numpy as np

# Dane
x = np.array([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
y = np.array([2, 7, 9, 12, 13, 14, 14, 13, 10, 8, 4])

# Model (a): y = ax + b
A_a = np.vstack([x, np.ones(len(x))]).T
print(A_a)
params_a = np.linalg.lstsq(A_a, y, rcond=None)[0]
a, b = params_a
y_pred_a = np.dot(A_a, params_a)
l2_error_a = np.linalg.norm(y - y_pred_a)
```

Współczynniki aproksymacji dla modelu (a):
a: 0.1818181818181818
b: 9.636363636363638

Norma L2 błędu dopasowania dla modelu (a): 12.763584563479451

W przypadku podpunktu b) macierz A wynosi:

$$A = \begin{bmatrix} 25 & -5 & 1 \\ 16 & -4 & 1 \\ 9 & -3 & 1 \\ 4 & -2 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \\ 16 & 4 & 1 \\ 25 & 5 & 1 \end{bmatrix} \tag{24}$$

Kod do wyliczania współczynników i normy 2l:

```
  A_b = np.vstack([x**2, x, np.ones(len(x))]).T
print(A_b)
params_b = np.linalg.lstsq(A_b, y, rcond=None)[0]
y_pred_b = np.dot(A_b, params_b)
l2_error_b = np.linalg.norm(y - y_pred_b)
print("Współczynniki aproksymacji dla modelu (b):")
print("a:", params_b[0])
print("b:", params_b[1])
print("c:", params_b[2])
print("Norma L2 łębdu dopasowania dla modelu (b):", l2_error_b)
```

Współczynniki aproksymacji dla modelu (b):
a: -0.4335664335664337
b: 0.18181818181818166
c: 13.972027972027968
Norma L2 błędu dopasowania dla modelu (b): 1.2737258819611161

Model (b) lepiej dopasowuje się do danych.

5 Zadanie nr. 5

Dopasuj szereg funkcji kosinusowych $g(x) = \sum_{j=0}^{n} c_j cosjx$ do funkcji kwadratowej $f(x) = \pi^2 - x^2$ w taki sposób, aby minimalizować błąd: $||f(x) - g(x)||_2$ wprzedziale $[0, \pi]$ Oceń błąd dla n = 1, 2, 3, ..., 10.

Poniżej implementacja rozwiązania w języku Python:

```
import numpy as np
import matplotlib.pyplot as plt

# Funkcja f(x)
def f(x):
  return np.pi**2 - x**2

# Funkcja g(x)
def g(x, n, c):
  sum = 0
  for j in range(n+1):
    sum += c[j] * np.cos(j*x)
  return sum

# Minimalizacja bledu
def minimize_error(n):
  # Wspolczynniki c
  c = np.zeros(n+1)

  # Macierz A
  A = np.zeros((n+1, n+1))
  for i in range(n+1):
    for j in range(n+1):
      A[i, j] = np.cos(i*j)

  # Wektor b
  b = np.zeros(n+1)
  for i in range(n+1):
    b[i] = f(i*np.pi/n)

  # Rozwiązanie ukladu równań
  c = np.linalg.solve(A, b)

  # Obliczenie bledu
  error = 0
  for i in range(n+1):
    error += (f(i*np.pi/n) - g(i*np.pi/n, n, c))**2

  return error

for n in range(1, 11):
```

```
    # Wspolczynniki c
    c = np.zeros(n+1)

    # Macierz A
    A = np.zeros((n+1, n+1))
    for i in range(n+1):
      for j in range(n+1):
        A[i, j] = np.cos(i*j)

    # Wektor b
    b = np.zeros(n+1)
    for i in range(n+1):
      b[i] = f(i*np.pi/n)

    # Rozwiązanie łukadu równań
    c = np.linalg.solve(A, b)


# Bledy
for n in range(1, 11):
  error = minimize_error(n)
  print(f"n={n}, łbąd={error}")
```

Wyniki obliczeń błędów dla zadanych wartości n:

```
n=1, błąd=1093.6208242606874
n=2, błąd=50.775736296527654
n=3, błąd=0.3461194363741739
n=4, błąd=168.28383001616405
n=5, błąd=269.63899806641734
n=6, błąd=726.4190934879125
n=7, błąd=422.91932245222927
n=8, błąd=310.09312416292926
n=9, błąd=248.57234583963657
n=10, błąd=215.05265655243966
```

6 Zadanie nr. 6

Dla macierzy

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{pmatrix}$$

Oraz dokładnego rozwiązania

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

wyznacz wektor danych dla modelu: Ax=b, a następnie:

(a) dla N = 10, wyznacz błąd rozwiązania $||x - x_*||$ oraz błąd residualny $||b - Ax||_2$ w funkcji parametru regularyzacji stosując: (i) standardową regularyzację Tichnowa, (ii) TSVD.

(b) oszacuj rank(A) i cond(A) oraz A^+,

(c) wyznacz krzywą L i oszacuj optymalną wartość parametru regularyzacji dla obu metod.

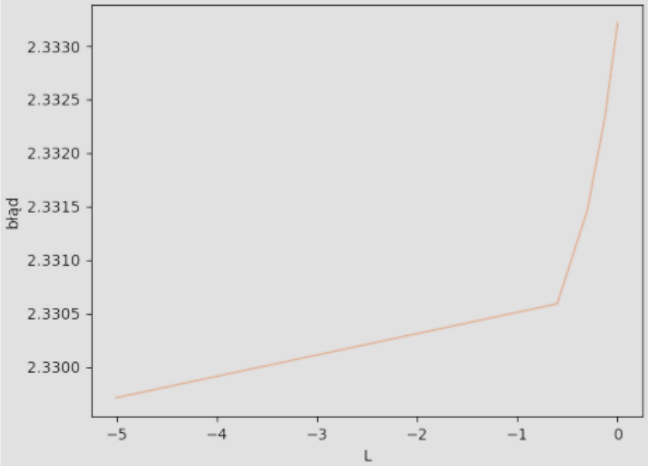
Napisano skrypt:

```
import numpy as np

Import matplotlib.pyplot


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

                        [4, 5, 6],

                        [7, 8, 9],

                        [10, 11, 12]

                        [13, 14, 15])

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


#Obliczenie wartosci wektora B

B = np.dot(A, x)

Pseudo_inv_A = np.linalg.pinv(A)
```

```
#Obliczenie wskaźnika uwarunkowania macierzy A

cond_A = np.linalg.cond(A)


#Wyznaczenie rzedu macierzy A

Rank_A = round(np.linalg.matrix_rank(A), 0)

x0 = np.dot(pseudo_inv_A, B)


#Obliczenie wartosci bledów

U, S, Vt = np.linalg.svd(A, full_matrices=False)

y1 = np.matmul(U.T, B)

S_inv = np.diag(1/S)

y2 = np. Matmul(S_inv, y1)

x1 = np.matmul(Vt.T, y2)


#Obliczenie wartosci bledu rozwiązania

blad_rozw = np.linalg.norm(x0 {x1, ord=2)


#Obliczenie wartosci bledu residualnego

blad_resi = np.linalg.norm(B {x1.dot(pseudo_inv_A), ord=2)


#Regularyzacja Tichnowa

L = np.linsapce(0.00001, 1 , 5)

blad_tich = np. Zeros(5)

X_tich = np.zeros(5)
```

```
For i in range(len(L)):
   x_tich[i] = np.linalg.norm(x1.dot(np.linalg.pinv(A)) - B, ord=2) +
       L[i]
   blad=tich[i] = np.linalg.norm(x1 {x_tich[i])
```

Otrzymano następujące wartości:
Błąd rozwiązania: 1.368774871883577e-15
Błąd residualny: 123.349910
Błąd regularyzacji Tichnowa: 215.385005
Cond(A): 3.5681136352506475e+16
Rank(A): 2
Krzywa L:



7 Zadanie nr. 7

Niech c = [0 1 ... N-1]^T należy do R^N będzie pierwszą kolumną, a r=[0 -1 ... -N+1] należy do R^N pierwszym wierszem macierzy Toeplitza.
Niech:
(A) x*=[1 2 ... N]^T należy do R^N,
(B) $x * N(0,1)$ należy do R^N (rozkład normalny).
Wykonaj projekcję "w przód" obu rozwiązań: $Ax* = b$, a następnie:
(a) dla N = 10, wyznacz błąd rozwiązania $||x - x_*||$ oraz błąd residualny $||b - Ax||_2$ w funkcji parametru regularyzacji stosując: (i) standardową regularyzację Tichnowa, (ii) TSVD.
(b) dla N = 10 przedstaw krzywe L, oddzielnie dla danych (A) i (B).
(c) oszacuj rank(A) i cond(A) dla N = 5, 10, 50, 100.
(d) przedstaw wykres zależności optymalnego parametru regularyzacji w funkcji wymiaru N dla danych (A) i (B),

(e) Uzasadnij dlaczego błąd rozwiązania jest dużo mniejszy dla danych typu (A) niż dla danych (B).

Ponizej kod realizujacy zadanie:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import toeplitz

def multiply_matrix_vector(A, x):
    return A.dot(x)

def calculate_errors(x_approx, x_exact, A, b):
    solution_error = np.linalg.norm(x_approx - x_exact)
    residual_error = np.linalg.norm(b - A.dot(x_approx))
    return solution_error, residual_error

def perform_regularizations(A, b, x_exact, lambda_tikhonov, k_tsvd):
    # Tikhonov Regularization
    A_tilde = np.vstack((A, lambda_tikhonov * np.eye(A.shape[1])))
    b_tilde = np.concatenate((b, np.zeros(A.shape[1])))
    x_tikhonov = np.linalg.lstsq(A_tilde, b_tilde, rcond=None)[0]

    # TSVD Regularization
    U, sigma, VT = np.linalg.svd(A, full_matrices=False)
    sigma_truncated = np.zeros_like(sigma)
    sigma_truncated[:k_tsvd] = sigma[:k_tsvd]
    Sigma_truncated = np.diag(sigma_truncated)
    Sigma_truncated_inv = np.diag([1 / s if s > 0 else 0 for s in
        sigma_truncated])
    x_tsvd = VT.T @ Sigma_truncated_inv @ U.T @ b

    # Calculate and store errors
    solution_error, residual_error = calculate_errors(x_tikhonov,
        x_exact, A, b)

    return x_tikhonov, solution_error, residual_error

def estimate_rank_and_condition(A):
    rank_A = np.linalg.matrix_rank(A)
    cond_A = np.linalg.cond(A)
    return rank_A, cond_A

def plot_l_curve(A, b, x_exact, lambda_values, subplot_title):
    norms_solution = []
    norms_residual = []

    for lam in lambda_values:
        _, solution_error, residual_error = perform_regularizations(A,
```

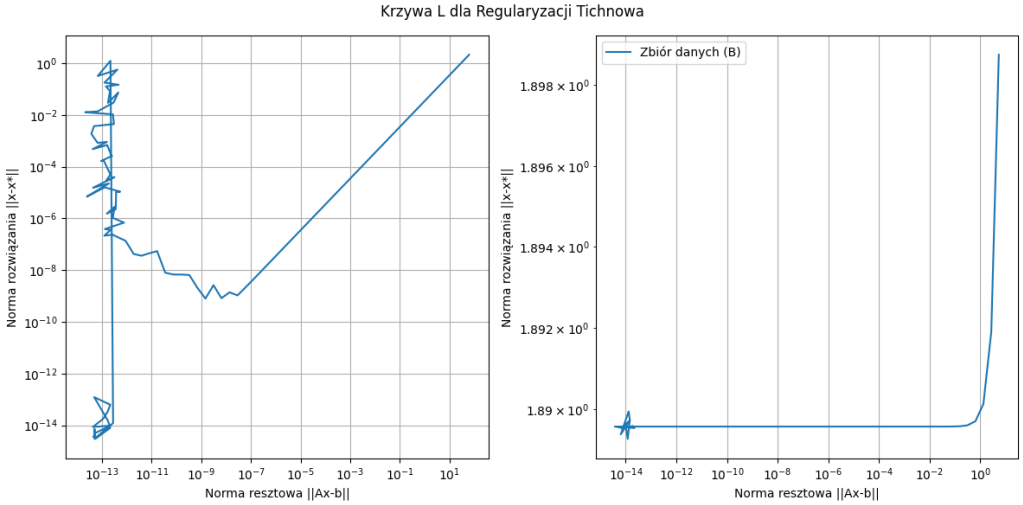
```
            b, x_exact, lam, N)
        norms_solution.append(solution_error)
        norms_residual.append(residual_error)

    plt.loglog(norms_residual, norms_solution, label=subplot_title)
    plt.xlabel('Norma resztowa ||Ax-b||')
    plt.ylabel('Norma rozwiązania ||x-x*||')
    plt.grid(True)

def plot_l_curve_tsvd(A, b, x_exact, k_values, dataset_name):
    norms_solution = []
    norms_residual = []

    U, S, Vt = np.linalg.svd(A, full_matrices=False)

    for k in k_values:
        S_truncated = np.zeros_like(S)
        S_truncated[:k] = S[:k]
        Sigma_truncated_inv = np.diag(1 / S_truncated[:k])
        x_tsvd = Vt[:k, :].T @ Sigma_truncated_inv @ U[:, :k].T @ b

        solution_error, residual_error = calculate_errors(x_tsvd,
            x_exact, A, b)
        norms_solution.append(solution_error)
        norms_residual.append(residual_error)

    plt.loglog(norms_residual, norms_solution, label=f'TSVD for
        {dataset_name}')
    plt.xlabel('Norma resztowa (||Ax - b||)')
    plt.ylabel('Norma rozwiązania (||x-x*||)')
    plt.grid(True)

def plot_optimal_regularization_parameter(N_values):
    optimal_lambda_A = []
    optimal_lambda_B = []

    for N in N_values:
        # Generate Toeplitz matrix A
        c = np.arange(N)
        r = np.concatenate(([c[0]], -c[1:]))
        A = toeplitz(c, r)

        # Generate solutions x* for datasets (A) and (B)
        x_star_a = np.arange(1, N + 1)
        x_star_b = np.random.normal(0, 1, N)

        # Generate forward projections b for datasets (A) and (B)
        b_a = multiply_matrix_vector(A, x_star_a)
        b_b = multiply_matrix_vector(A, x_star_b)
```

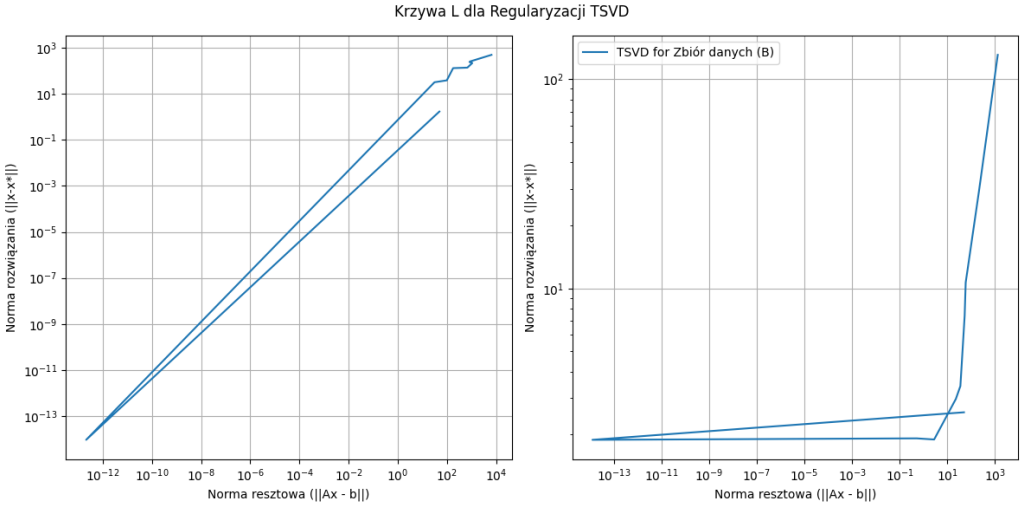
```
        # Determine optimal lambda for dataset (A) - Placeholder
        optimal_lambda_A.append(np.argmin(
            [calculate_errors(perform_regularizations(A, b_a, x_star_a,
                lam, N)[0], x_star_a, A, b_a)[1] for lam in
             lambda_values]))

        # Determine optimal lambda for dataset (B) - Placeholder
        optimal_lambda_B.append(np.argmin(
            [calculate_errors(perform_regularizations(A, b_b, x_star_b,
                lam, N)[0], x_star_b, A, b_b)[1] for lam in
             lambda_values]))

    # Plotting the optimal lambda values
    plt.figure(figsize=(10, 6))
    plt.plot(N_values, optimal_lambda_A, 'o-', label='Optymalna śćwarto
        lambda dla zbioru danych (A)')
    plt.plot(N_values, optimal_lambda_B, 's-', label='Optymalna śćwarto
        lambda dla zbioru danych(B)')
    plt.xlabel('śćWielko macierzy N')
    plt.ylabel('Optymalna śćwarto parametru lambda')
    plt.title('Optymalna śćwarto parametru lambda w zależśnoci od
        świelkoci macierzy N')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# Main code section
lambda_values = np.logspace(-15, 1, 100)
N_values = [5, 10, 50, 100]

# Estimate and print rank and condition number for Toeplitz matrices of
    different sizes
for N in N_values:
    c = np.arange(N)
    r = np.concatenate(([c[0]], -c[1:]))
    A = toeplitz(c, r)
    rank_A, cond_A = estimate_rank_and_condition(A)
    print(f"N={N}: rank(A)={rank_A}, cond(A)={cond_A:.2e}")

# Create plots for the L-curve for data set (A) and (B) for N=10
N = 10
c = np.arange(N)
r = np.concatenate(([c[0]], -c[1:]))
A = toeplitz(c, r)

# Generate solutions x* for datasets (A) and (B)
x_star_a = np.arange(1, N + 1)
x_star_b = np.random.normal(0, 1, N)
```

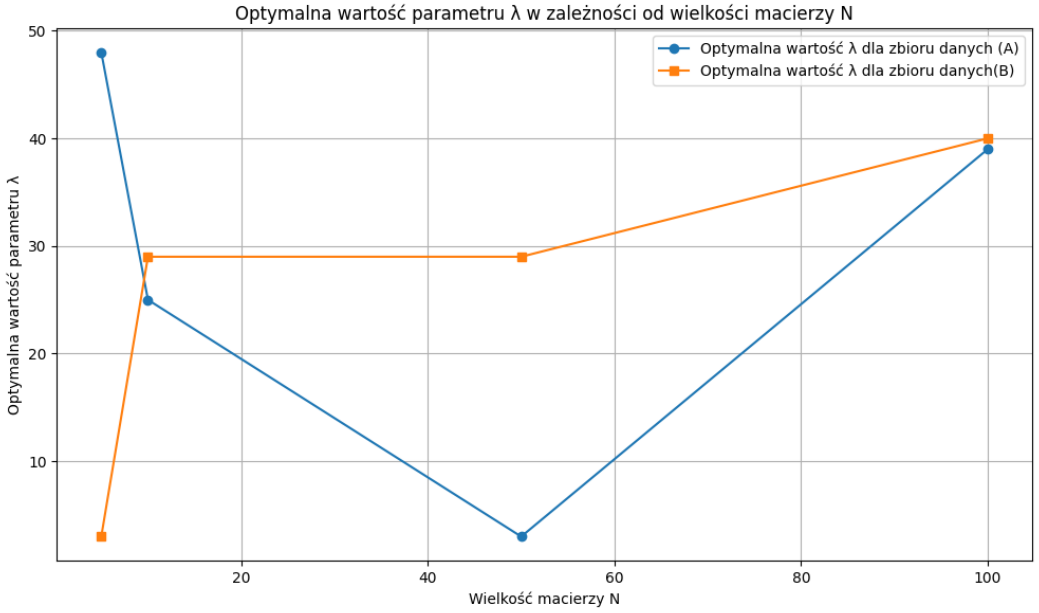
```
# Forward projections b
b_a = multiply_matrix_vector(A, x_star_a)
b_b = multiply_matrix_vector(A, x_star_b)

k_values = range(1, min(A.shape) + 1)

plt.figure(figsize=(12, 6))

# Plot the L-curve for data set (A)
plt.subplot(1, 2, 1)
plot_l_curve(A, b_a, x_star_a, lambda_values, 'Zbiór danych (A)')

# Plot the L-curve for data set (B)
plt.subplot(1, 2, 2)
plot_l_curve(A, b_b, x_star_b, lambda_values, 'Zbiór danych (B)')

plt.suptitle('Krzywa L dla Regularyzacji Tichnowa')
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))

# Plot the L-curve for data set (A)
plt.subplot(1, 2, 1)
plot_l_curve_tsvd(A, b_a, x_star_a, k_values, 'Zbiór danych (A)')

# Plot the L-curve for data set (B)
plt.subplot(1, 2, 2)
plot_l_curve_tsvd(A, b_b, x_star_b, k_values, 'Zbiór danych (B)')

plt.suptitle('Krzywa L dla Regularyzacji TSVD')
plt.legend()
plt.tight_layout()
plt.show()

# Plot optimal regularization parameter as a function of N for datasets
    (A) and (B)
plot_optimal_regularization_parameter(N_values)
```

Wyniki obliczeń szacunkowych:

$$\begin{cases} N = 5 : rank(A) = 2, cond(A) = 3.40e17 \\ N = 10 : rank(A) = 2, cond(A) = 2.03e18 \\ N = 50 : rank(A) = 2, cond(A) = 6.36e19 \\ N = 100 : rank(A) = 2, cond(A) = 6.49e20 \end{cases} \tag{25}$$


Krzywa L dla Regularyzacji Tichnowa
Norma rozwiązania ||x-x*||
Norma resztowa ||Ax-b||
Zbiór danych (B)
Norma rozwiązania ||x-x*||
Norma resztowa ||Ax-b||


Krzywa L dla Regularyzacji TSVD
Norma rozwiązania (||x-x*||)
Norma resztowa (||Ax - b||)
TSVD for Zbiór danych (B)
Norma rozwiązania (||x-x*||)
Norma resztowa (||Ax - b||)


Optymalna wartość parametru λ w zależności od wielkości macierzy N
Optymalna wartość λ dla zbioru danych (A)
Optymalna wartość λ dla zbioru danych(B)
Optymalna wartość parametru λ
Wielkość macierzy N
50
40
30
20
10
20
40
60
80
100

8 Wnioski

Błąd rozwiązania dla danych typu A jest duzo mniejszy ze wzgledu na wykorzystanie w zbiorze B liczb z zakresu rozkładu normalnego.