МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота №2

з дисципліни
«Чисельні методи»

Виконав:
Студент групи ШІ-22
Михальчук Антон
Перевірила:
доцент кафедри СШІ
Гентош Леся Ігорівна

Львів-2025

Лабораторна робота № 2

Тема: Прямі методи розв’язування СЛАР.

Мета – набути навиків практичного використання прямих методів розв’язування СЛАР: методу LU-розкладу, методу квадратних коренів, методу ортогоналізації та методу поворотів.

Варіант 15

Постановка завдання

- Скласти програму, яка реалізує знаходження розв’язку СЛАР за допомогою методу поворотів.
- Програма повинна обчислювати визначник.
- Додати модифікацію прогонки тридіагональної матриці. Првава, ліва, середня прогонки.
- Перевірити виконання наданних завдань.
- Перевірити виконання для матриць n>1000.
- Знайти обчислювальну, просторову та часову складність.
- Перевірити на достовірність, точність та стійкість.

Код програмної реалізації

```python
import numpy as np
from pathlib import Path

n = 10000
ZERO_PERCENTAGE = 0
INPUT_ID = 5
GENERATE_TRIDIAGONAL = True

if GENERATE_TRIDIAGONAL:
    sub_diag = np.random.uniform(-100, 100, n - 1)
    super_diag = np.random.uniform(-100, 100, n - 1)

    sub_diag_padded = np.concatenate(([0], sub_diag))
    super_diag_padded = np.concatenate((super_diag, [0]))

    required_c_magnitude = np.abs(sub_diag_padded) +
np.abs(super_diag_padded)
    stability_buffer = np.random.uniform(1.0, 100.0, n)
    main_diag_magnitude = required_c_magnitude + stability_buffer

    main_diag_sign = np.random.choice([-1, 1], n)
    main_diag = main_diag_magnitude * main_diag_sign

    a = np.zeros((n, n))
    np.fill_diagonal(a, main_diag)
    np.fill_diagonal(a[1:], sub_diag)
    np.fill_diagonal(a[:, 1:], super_diag)

    f = np.random.uniform(-100, 100, n)

    matrix = np.hstack((a, f.reshape(-1, 1)))

else:
    random_mask = np.random.rand(n, n + 1)
    zero_mask = random_mask < ZERO_PERCENTAGE
    matrix = np.random.uniform(-1000, 1000, size=(n, n + 1))
    matrix[zero_mask] = 0

output_dir = Path("..") / "inputs"
output_dir.mkdir(parents=True, exist_ok=True)

if n > 1000:
    file_path = output_dir / f"input{INPUT_ID}.npz"
    np.savez_compressed(file_path, a=matrix[:, :n], b=matrix[:, n])
    print(f"Matrices saved to file {file_path}")
else:
    file_path = output_dir / f"input{INPUT_ID}.txt"
    with open(file_path, 'w') as f:
        f.write(f"{n}\n")
        np.savetxt(f, matrix)
    print(f"Matrix saved to file {file_path}")
```

create_inputs.py

Генератор випадкових СЛАР з довільним параметром n та визначенням відсотку нульових елементів:

```python
import warnings
import time
import numpy as np
from rotation import solve_rotation
from tridiagonal import solve_tridiagonal
from data_handler import read_sole_data, method_evaluation, save_solution

np.seterr(divide='raise', invalid='raise')
warnings.simplefilter('error', RuntimeWarning)

INPUT_ID = 5
DECIMAL_PLACES = 60
IS_EVALUATE = True
METHOD = solve_tridiagonal

def main():
    a, b = read_sole_data(INPUT_ID)
    if a is None or b is None:
        return

    start_time = time.time()
    x, det = METHOD(a, b)
    end_time = time.time()
    execution_time = end_time - start_time

    if x is not None:
        if IS_EVALUATE:
            method_evaluation(a, x, b, det, execution_time, INPUT_ID, DECIMAL_PLACES,
METHOD) else:
            print(f"Execution time: {execution_time}\n")

    save_solution(x, INPUT_ID, DECIMAL_PLACES)

if __name__ == '__main__':
    main()
```

main.py

У цьому файлі відбувається налаштування чисельного експерименту. Існує можливість обрати: метод (постовпцевий, чи повний вибір головного елементу) та чи проводити оцінку, чи лише видати обрахунок.

```python
import numpy as np
from decimal import Decimal, getcontext

def solve_rotation(a, b, det_evaluation=True):
    print("Starting Rotation method...")
    n = len(a)
    a = a.copy()
    b = b.copy()
    det = None

    try:
        print("Forward elimination...")
        for k in range(n - 1):
            for i in range(k+1, n):
                r = np.sqrt(a[k,k] ** 2 + a[i,k] ** 2)
                c = a[k,k] / r
                s = a[i,k] / r

                row_k = a[k, k:].copy()
                row_i = a[i, k:].copy()
                a[k, k:] = c * row_k + s * row_i
                a[i, k:] = -s * row_k + c * row_i

                b_k_old = b[k]
                b_i_old = b[i]
                b[k] = c * b_k_old + s * b_i_old
                b[i] = -s * b_k_old + c * b_i_old
        print("Forward elimination finished.")

        if det_evaluation:
            print("Calculating determinant...")
            getcontext().prec = n * 5
            det = Decimal(1)
            for i in range(n):
                det *= Decimal(a[i][i])
            print("Determinant calculated.")

        print("Backward substitution...")
        x = np.zeros(n)
        x[n - 1] = b[n - 1] / a[n - 1, n - 1]

        for k in range(n - 2, -1, -1):
            x[k] = (b[k] - np.sum(a[k, k + 1:] * x[k + 1:])) / a[k,
k]      print("Solution found.")

    except (FloatingPointError, ZeroDivisionError):
        print("Error: Matrix is singular. No solution.")
        return None, 0

    print("Calculations finished.")
    return x, det
```

rotation.py

Файл з функцією розв’язання СЛАР методом Поворотів.

```python
import numpy as np
from decimal import Decimal, getcontext

def solve_tridiagonal(D_matrix, f_vector):
    n = D_matrix.shape[0]
    tol = 1e-12

    print("--- Starting Shuttle Method for Tridiagonal Systems ---")

    print("1. Checking: Tridiagonal matrix structure.")
    for i in range(n):
        for j in range(n):
            if abs(i - j) > 1 and not abs(D_matrix[i, j]) < tol:
                print("ERROR: Matrix is not tridiagonal!")
                return None, 0
    print("   -> OK. Matrix is tridiagonal.")

    if D_matrix.shape[0] != D_matrix.shape[1] or D_matrix.shape[0] != len(f_vector):
        print("Error: Matrix and vector dimensions do not match.")
        return None, 0

    N = D_matrix.shape[0] - 1
    if N < 2:
        print("Error: Matrix is too small for the shuttle method.")
        return None, 0

    print("2. Checking: Sufficient conditions for correctness and stability.")
    H1 = -D_matrix[0, 1]
    H2 = -D_matrix[N, N - 1]

    c_diag = np.array([D_matrix[i, i] for i in range(1, N)])
    a_diag = np.array([D_matrix[i, i - 1] for i in range(1, N)])
    b_diag = np.array([D_matrix[i, i + 1] for i in range(1, N)])

    is_diagonally_dominant = True
    strict_inequality_found = False

    for i in range(N - 1):
        if abs(c_diag[i]) < abs(a_diag[i]) + abs(b_diag[i]):
            is_diagonally_dominant = False
            break
        if abs(c_diag[i]) > abs(a_diag[i]) + abs(b_diag[i]):
            strict_inequality_found = True

    h1_cond_met = abs(H1) <= 1
    h2_cond_met = abs(H2) <= 1

    if abs(H1) < 1 or abs(H2) < 1:
        strict_inequality_found = True

    all_conditions_met = is_diagonally_dominant and h1_cond_met and h2_cond_met and strict_inequality_found

    if not all_conditions_met:
        print(
            "   -> WARNING! Sufficient conditions for stability are NOT met. Calculation proceeds, but may be 
unstable.")
    else:
        print("   -> OK. Sufficient conditions for stability are met.")

    print("3. Extracting coefficients from matrix and vector...")
    mu1 = f_vector[0]
    mu2 = f_vector[N]

    A_coeffs = np.array([D_matrix[i, i - 1] for i in range(1, N)])
    B_coeffs = np.array([D_matrix[i, i + 1] for i in range(1, N)])
    C_coeffs = np.array([-D_matrix[i, i] for i in range(1, N)])
    f_coeffs = np.array([-f_vector[i] for i in range(1, N)])
    print("   -> Coefficients extracted.")

    m = N // 2

    alpha = np.zeros(N + 2)
    beta = np.zeros(N + 2)
    gamma = np.zeros(N + 2)
    delta = np.zeros(N + 2)
    y = np.zeros(N + 1)

    getcontext().prec = 80
    determinant = Decimal(1)

    try:
        print(f"4. Rightward pass (from i=0 to i={m})...")
        alpha[1] = H1
        beta[1] = mu1

        for i in range(1, m + 1):
            idx = i - 1
            denominator = C_coeffs[idx] - A_coeffs[idx] * alpha[i]
            determinant *= Decimal(denominator)
            if abs(denominator) < tol:
                raise ZeroDivisionError(f"Denominator is near zero in the rightward pass at i={i}.")
            alpha[i + 1] = B_coeffs[idx] / denominator
            beta[i + 1] = (f_coeffs[idx] + A_coeffs[idx] * beta[i]) / denominator

        print(f"5. Leftward pass (from i={N} to i={m + 1})...")
        gamma[N] = H2
        delta[N] = mu2

        for i in range(N - 1, m, -1):
            idx = i - 1
            denominator = C_coeffs[idx] - B_coeffs[idx] * gamma[i + 1]
            determinant *= Decimal(denominator)
            if abs(denominator) < tol:
                raise ZeroDivisionError(f"Denominator is near zero in the leftward pass at i={i}.")
            gamma[i] = A_coeffs[idx] / denominator
            delta[i] = (f_coeffs[idx] + B_coeffs[idx] * delta[i + 1]) / denominator

        print(f"6. Meeting at point m={m} and solving for y[m]...")
        denominator_ym = 1 - alpha[m + 1] * gamma[m + 1]
        determinant *= Decimal(denominator_ym)

        if abs(denominator_ym) < tol:
            raise ZeroDivisionError("Denominator is near zero when calculating y_m.")

        y[m] = (alpha[m + 1] * delta[m + 1] + beta[m + 1]) / denominator_ym

        print("7. Backward substitution from center to boundaries...")
        for i in range(m - 1, -1, -1):
            y[i] = alpha[i + 1] * y[i + 1] + beta[i + 1]

        for i in range(m + 1, N + 1):
            y[i] = gamma[i] * y[i - 1] + delta[i]

    except ZeroDivisionError as e:
        print(f"ERROR: {e} Matrix is likely singular.")
        return None, 0

    print("--- Calculation finished ---")
    return y, determinant
```

tridiagonal.py

Метод прогонки тридіагональних матриць.

Львів-2025

```python
import numpy as np
from decimal import Decimal
from gaussian import solve_gaussian
import os

def read_sole_data(input_id):
    npz_path = f"inputs/input{input_id}.npz"
    txt_path = f"inputs/input{input_id}.txt"

    try:
        if os.path.exists(npz_path):
            print(f"Reading data from file {npz_path}...")
            data = np.load(npz_path)
            if 'a' in data and 'b' in data:
                a = data['a']
                b = data['b']
                print("Data loaded successfully.")
                return a, b
            else:
                print("Error: NPZ file does not contain keys 'a' and 'b'.")
                return None, None

        elif os.path.exists(txt_path):
            print(f"Reading data from file {txt_path}...")
            full_data = np.genfromtxt(txt_path, skip_header=1, delimiter=None,
filling_values=0)
            if full_data.ndim < 2:
                print("Error: Not enough data for matrix and vector.")
                return None, None
            a = full_data[:, :-1].copy()
            b = full_data[:, -1].copy()
            if a.shape[0] != a.shape[1] or a.shape[0] != b.shape[0]:
                print("Error: dimensions do not match.")
                return None, None
            print("Data loaded successfully.")
            return a, b

        else:
            print("Error: Data file not found.")
            return None, None

    except Exception as error:
        print(f"Error reading file: {error}")
        return None, None
```

data_handler.py

read_sole_data - функція для зчитування матриці A та вектора b з файлів формату .txt або .npz

```python
def method_evaluation(a, x, b, det, execution_time, input_id, pivoting_type, decimal_places):
    print("Starting method evaluation...")
    print("Step 1: Calculating benchmark determinant (np.linalg.det)...")
    try:
        det_benchmark = np.linalg.det(a)
        print("Benchmark determinant successfully calculated.")
    except RuntimeWarning as e:
        det_benchmark = None
        print(f"Error calculating benchmark determinant: {e}")

    print("Step 2: Calculating benchmark solution (np.linalg.solve)...")
    try:
        x_benchmark = np.linalg.solve(a, b)
        print("Benchmark solution successfully calculated.")
    except np.linalg.LinAlgError:
        x_benchmark = None
        print("Error: Matrix is singular. Cannot calculate benchmark solution.")

    print(f"Step 3: Opening file to write results: evaluations/evaluation{input_id}_{pivoting_type}.txt")
    with open(f"evaluations/evaluation{input_id}_{pivoting_type}.txt", 'w') as f:
        f.write(f"Execution time: {execution_time}\n")
        if det is not None:
            f.write(f"Determinant: {det:.{decimal_places}g}\n")
        if det_benchmark is not None:
            det_benchmark = Decimal(str(det_benchmark))
            f.write(f"Benchmark determinant: {det_benchmark:.{decimal_places}g}\n")
            abs_error_det = abs(det - det_benchmark)
            rel_error_det = abs_error_det / abs(det_benchmark)
            f.write(f"Absolute error of determinant: {abs_error_det}\n")
            f.write(f"Relative error of determinant: {rel_error_det}\n")

        if x is not None:
            print("Step 4: Calculating stability error...")
            epsilon = 1e-8
            b_perturbed = b + epsilon * np.random.randn(*b.shape)
            x_perturbed, det_perturbed = solve_gaussian(a, b_perturbed, type=pivoting_type,
det_evaluation=False)
            stability_error = np.linalg.norm(x_perturbed - x)
            f.write(f"Stability error: {stability_error}\n")
            print("Stability error calculated.")

        print("Step 5: Calculating condition number of matrix Cond(A)...")
        cond_a = np.linalg.cond(a.copy())
        f.write(f"Cond A: {cond_a}\n")
        print("Condition number calculated.")
        if x is not None and x_benchmark is not None:
            print("Step 6: Calculating solution errors (absolute and relative)...")
            abs_error_solution = np.linalg.norm(x - x_benchmark)
            rel_error_solution = abs_error_solution / np.linalg.norm(x_benchmark)
            f.write(f"Absolute error of solution: {abs_error_solution}\n")
            f.write(f"Relative error of solution: {rel_error_solution}\n")
            print("Solution errors calculated.")
```

data_handler.py
method_evaluation - функція, що проводить оцінку метода, завдяки порівнянню похибки розв'язку з "золотим стандартом" відповідними методами бібліотеки numpy.
У файл evaluation записується дискримінант, абсолютна та відносна похибка дискримінанту, стійкість методу, абсолютна та відносна похибка розв'язків.

```python
def save_solution(x, input_id, pivoting_type, decimal_places):
    n = len(x)
    output_dir = "outputs"
    os.makedirs(output_dir, exist_ok=True)
    if x is not None:
        if n > 1000:
            npz_path = f"{output_dir}/output{input_id}_{pivoting_type}.npz"
            np.savez_compressed(npz_path, x=x)
            print(f"Solution saved to {npz_path}")
        else:
            txt_path = f"{output_dir}/output{input_id}_{pivoting_type}.txt"
            np.savetxt(txt_path, x, fmt=f'%.{decimal_places}g')
            print(f"Solution saved to {txt_path}")
    else:
        txt_path = f"{output_dir}/output{input_id}_{pivoting_type}.txt"
        with open(txt_path, 'w') as f:
            f.write("Matrix is singular.")
        print(f"Matrix is singular. Saved to {txt_path}")
```

data_handler.py

save_solution – зберігає розв’язок у форматі .txt або .npz залежно від розміру СЛАР.

Аналіз чисельних експериментів

Перше завдання

input1.txt

```
4
20  -4  -3   8   2
-4 -26  -4   2 -12
-3  -4  20   2  -4
 8   2   2 -26   4

```

Розв’язок:

output1.txt

```
0.201667508842849951822273624202352948486804962158203125
0.4372662961091460243068240743014030158519744873046875
-0.07589691763516930034239038604937377385795116424560546875
-0.06399696816574029778479371088906191289424896240234375
```

Оцінка:

evaluation1.txt

```
Method name: solve_rotation
Matrix size: 4
Execution time: 0.00041937828063964844
Determinant: 316639.99999999989362
Benchmark determinant: 316640.00000000035
Absolute error of determinant: 4.5638E-10
Relative error of determinant: 1.4413213744315294831E-15
Stability error: 1.2072698833201454e-09
Residual norm of solution: 1.2560739669470201e-15
Relative residual norm: 5.238973632143384e-17
Cond A: 1.6025260709894267
Absolute error of solution: 5.551115123125783e-17
Relative error of solution: 1.1290603527181895e-16
Machine error for IEEE 754 standard ε ≈ 2.2e-16
```

Друге завдання

second_lab/.../input2.txt

```
3
13 -6 2 1
-6 22 4 3
2 4 -14 3
```

Розв’язок:

output2.txt

```
0.1942446043165466929014684183130157180130481719970703125
0.2122302158273381145203728692649747245013713836669921875
-0.12589928057553956275427253785892389714717864990234375
```

Оцінка:

evaluation2.txt

```
Method name: solve_rotation
Matrix size: 3
Execution time: 0.0003170967102050781
Determinant: -3892.00000000001
Benchmark determinant: -3892.0000000000005
Absolute error of determinant: 9.5E-12
Relative error of determinant: 2.44090441932169E-15
Stability error: 5.675334799188837e-10
Residual norm of solution: 9.930136612989092e-16
Relative residual norm: 1.0200086463287648e-16
Cond A: 2.396137707323496
Absolute error of solution: 1.2412670766236366e-16
Relative error of solution: 3.9525335045239625e-16
Machine error for IEEE 754 standard ε ≈ 2.2e-16
```

Третє завдання

input3.txt

```
3
13 -3 4 14
-3 -4 0 -7
4 0 -5 -1
```

Розв'язок:

output3.txt

```
0.9999999999999997779553950749686919152736663818359375
0.9999999999999997779553950749686919152736663818359375
1.0000000000000002220446049250313080847263336181640625
```

Оцінка:

evaluation3.txt

```
Method name: solve_rotation
Matrix size: 3
Execution time: 0.0003566741943359375
Determinant: 369.000000000000
Benchmark determinant: 368.99999999999994
Absolute error of determinant: 6E-14
Relative error of determinant: 1.62601626016260E-16
Stability error: 5.759953001054833e-10
Residual norm of solution: 4.351167857633658e-15
Relative residual norm: 1.5579680111468782e-16
Cond A: 3.3806002035964937
Absolute error of solution: 3.8459253727671276e-16
Relative error of solution: 2.220446049250313e-16
Machine error for IEEE 754 standard ε ≈ 2.2e-16
```

Обчислювальна складність методу Поворотів:

Ініціалізація копіювання матриці та вектора:

$$O_{init}(n) = n^2 + n$$

Прямий хід:

$$O_{forward}(n) = \sum_{k=0}^{n-2} \sum_{i=k+1}^{n-1} (4(n-k)+1) = \frac{4}{3}n^3 + n^2 - \frac{19}{3}n + 4$$

Обрахунок визначника:

$$O_{det}(n) = n$$

Зворотна підстановка:

$$O_{back}(n) = \sum_{k=0}^{n-2} 2(n-k-1) = n^2 - n$$

Разом:

$$O\left(\frac{4}{3}n^3\right) = \frac{4}{3}n^3 + 3n^2 - \frac{16}{3}n$$

Просторова складність методу поворотів:

Оригінальні матриця та вектор:

$$O(n^2) + O(n)$$

Копіювання матриці та вектора:

$$O(n^2) + O(n)$$

Допоміжні вектори: x:

$$O(n)$$

Загалом у байтах:

$$8n^2 + 8n + 8n^2 + 8n + 8n = 16n^2 + 24n$$

Часова складність методу поворотів:

□(□)=□O(□)

τ можна знайти експериментально, порахуємо час виконання алгоритмів при n=1000:

$$\tau = \frac{T(n)}{O(n)} = \frac{6.966634273529053}{\frac{4}{3}1000^3 + 3 \cdot 1000^2 - \frac{16}{3}1000} = 5.213266708120351e - 09$$

Обчислення для n=10000:

Наявними операційними потужностями є 16 гігабайтів оперативної пам’яті.

Прогнозоване обмеження на n:

$$16n^2 + 24n < 16000000000$$

З чого виводимо, що приблизним обмеженням доступної оперативної пам’яті повинна бути СЛАР з розміром 31600.

Завдячуючи, часовій складності можна спрогнозувати, що обчислення такого СЛАР займе близько 60 годин.

Проведемо експеримент з n=10000:

Прогнозоване використання пам’яті:

16·100002+24·10000=1600240000 *байт.* ≈ 1.6 *гб.*

Результати експерименту:

evaluation3.txt   evaluation6.txt

```
Method name: solve_rotation
Matrix size: 10000
Execution time: 1236.2821786403656
Determinant: -1.90234275378511636517130122588454606231939799414313156799014e+45212
Stability error: 6.549300934174995e-09
Residual norm of solution: 1.8469103904554806e-07
Relative residual norm: 6.469344624343806e-17
Cond A: 94557.24950780396
Absolute error of solution: 1.1022162848667295e-08
Relative error of solution: 2.1145506894026512e-11
Machine error for IEEE 754 standard ε ≈ 2.2e-16
```

Вдалося приблизно спрогнозувати використану оперативну пам’ять. Протягом всього виконання в середньому спостерігалася витрата 1.6 ГБ оперативної пам’яті:

Name	ID	CPU	Memory	Disk Rea
pycharm	6967	0.2 %	1.6 GB	
python	13848	6.3 %	1.6 GB	

Детермінант виявився більшим, аніж може дозволити утримувати тип float64, тому було прийнято рішення використати бібліотеку decimal для обрахунку чисел з довільною точністю. Однак, отриманий детермінант виявився астрономічної величини $\sim 10^{45212}$, через що зникає будь-яка можливість перевірити його на точність, адже бібліотека numpy не може обрахувати такого розміру числа. Проте варто вказати те, що детермінанти отримані від двох методів майже збігаються.

Для аналізу на достовірність та точність було використано бібліотеку мови програмування python - numpy. Вона є достовірним “золотим стандартом”, що має підтверджену подвійну точність згідно стандарту IEEE 754.

Аналіз на достовірність:

Абсолютні та відносні похибки були обчислені через другу норму різниці між нашим розв’язком та розв’язком, отриманим за допомогою NumPy. Такі похибки свідчать про достовірність чисельного розв’язку. Варто зауважити, що вдвічі швидший частковий метод дав менші помилки, аніж складніший метод поворотів.

Аналіз на точність:

Відносні похибки знаходяться на рівні очікуваної точності для precision і повністю узгоджуються з оцінкою через умову матриці.

$$\delta a < cond(A) \cdot \varepsilon_{mach} \approx 9.54 \cdot 10^4 \cdot 2.22 \cdot 10^{-16}$$

Відносна похибка методу поворотів:
$2.115 \cdot 10^{-11} \approx 2.08 \cdot 10^{-11}$
Відносна норма нев’язки є меншою за машинний епсилон:

$$\frac{||A\hat{x} - b||}{||A||\,||b||} < \varepsilon_{mach}$$

$$6.469 \cdot 10^{-17} < 2.22 \cdot 10^{-16}$$

Це підтверджує, що метод Поворотів дав високоточний розв’язок.

Аналіз на стійкість:

Стійкість оцінювалася як друга норма різниці між нашим розв’язком та розв’язком для модифікованих вхідних даних з доданим малим шумом.

Метод поворотів виявився менш стійким, аніж метод Гаусса.

Різниця між розв’язками у 9 цифр після коми демонструє, що метод зберігає числову стійкість щодо невеликих варіацій у вхідних даних, і його похибки відповідають передбачуваним оцінкам за умовою числової задачі.

Середня (зустрічна) прогонка тридіагональної матриці

Було проведено чисельний експеримент з n=10000 на двох тридіагональних матрицях.

Перша матриця була випадково створена, а тому не проходила перевірку на стійкість, немаючи діагональної переваги, що призвело до величезного cond. Це призвело до того, що результат роботи чисельного методу виявився недостовірним, абсолютна похибка 0.74 секстильйона є неприпустимою.

evaluation4.txt

```
Method name: solve_tridiagonal
Matrix size: 10000
Execution time: 43.4241578578949
Determinant: 3.03938477040932252351297192097846979546912028363646514425362e+25687
Stability error: 1.0526734863739258e+24
Residual norm of solution: 4.658648676429954e+19
Relative residual norm: 2.183686186008283e-21
Cond A: 1.5353716060968633e+21
Absolute error of solution: 7.363622528744953e+20
Relative error of solution: 2.80692825789633e-15
Machine error for IEEE 754 standard ε ≈ 2.2e-16
```

Друга матриця на противагу попередній була спеціально згенерована з діагональною перевагою, що дало можливіть отримати відносно низький cond, дозволивши пройти перевірку на стійкість.

Method name: solve_tridiagonal
Matrix size: 10000
Execution time: 20.947228908538882
Determinant: -5.22083987533952422552967691340024147612450039199648550931121e+21423
Stability error: 1.139884987228621e-08
Residual norm of solution: 1.1990693619808691e-12
Relative residual norm: 1.0164093631478534e-18
Cond A: 59.19950597606899
Absolute error of solution: 2.1741970433094974e-14
Relative error of solution: 3.2697504197096675e-16
Machine error for IEEE 754 standard ε ≈ 2.2e-16

Обчислювальна складність:

Перевірка на тридіагональність: (n+1)^2

Перевірка на стійкість: (n-1) + 2

Витягування коефіцієнтів: 2 * (n – 1)

Прямі ходи: 6 * (n-1)

Зустріч: 5

Зворотній хід: 2n

Виключно метод: 13n-1

Висновки

Метод Поворотів є стійким і високоточним для загальних систем, що підтверджено малою відносною похибкою ($\approx 10^{-11}$), яка відповідає оцінці через число обумовленості cond(A). Його обчислювальна складність $O(n^3)$ робить його повільним для надвеликих матриць.

Модифікація Прогонки є значно ефективнішою зі складністю O(n), що ідеально підходить для тридіагональних матриць.

Критична залежність: Експерименти показали, що ефективність і надійність лінійного методу Прогонки критично залежить від властивостей матриці (наявності діагональної переваги), тоді як метод Поворотів є більш універсальним.