НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра штучного інтелекту

ЛАБОРАТОРНА РОБОТА №3

з дисципліни «Проектування та аналіз обчислювальних алгоритмів»

Варіант № 17

Виконав:
Студент II курсу
Групи КІ-32
Присяжнюк Владислав
Прийняв:
Тимошенко Ю. О.

Київ 2024

Мета роботи:

Метою цієї лабораторної роботи є вивчення, реалізація та застосування динамічного методу першого та другого порядків для розв’язання жорстких систем звичайних диференційних рівнянь (ЗДР), що відповідають системам лінійних алгебраїчних рівнянь (СЛАР). Основний акцент роботи полягає на реалізації алгоритму Булірша-Штера для інтегрування ЗДР першого порядку, що дозволяє отримати розв’язок із заданою точністю.

Теоретична частина:

СЛАР можна розв’язувати як динамічну систему, використовуючи підходи динамічного методу першого порядку або методу другого порядку:

Динамічний метод першого порядку:

$$\frac{dx(t)}{dt} + A^T Ax\,(t) = A^T b,\ x(0) = 0,$$

Динамічний метод другого порядку:

$$\varepsilon_1 x''(t) + \varepsilon_2 x'(t) + A^T Ax\,(t) = A^T b,\ x(0) = 0,\ x'(0) = 0$$

Метод Булірша-Штера— це чисельний алгоритм для розв’язування систем ЗДР із високою точністю. Він заснований на:

- **Модифікованому методі середньої точки**: Використовується для побудови наближень на малих інтервалах часу. Цей метод розраховує проміжні значення за допомогою послідовного уточнення розв’язку.
- **Екстраполяції Річардсона**: Покращує точність шляхом поєднання кількох наближень, отриманих для різних кроків інтегрування. Це дозволяє отримати розв’язок із меншою похибкою, ніж при використанні фіксованого кроку.
- **Адаптивного регулювання кроку** h: Алгоритм автоматично змінює крок інтегрування залежно від поточної похибки. Це дозволяє забезпечити високу точність і стійкість навіть для жорстких задач.

Жорсткими називаються такі системи, де існують значні розбіжності в масштабах часу (наприклад, одні змінні змінюються дуже швидко, а інші — повільно).

Практична частина:

Використаємо матрицю з попередньої лабороторної роботи:

$$A = \begin{matrix} 1 & 0.99 & 0.98 & 0.97 & 0.96 \\ 0.99 & 1 & 0.99 & 0.98 & 0.97 \\ 0.98 & 0.99 & 1 & 0.99 & 0.98 \\ 0.97 & 0.98 & 0.99 & 1 & 0.99 \\ 0.97 & 0.97 & 0.98 & 0.99 & 1 \end{matrix}$$

Знайдемо праву частину за допомогою підстановки одиниць до вектора x:

$$b = \begin{matrix} 4.9 \\ 4.93 \\ 4.94 \\ 4.93 \\ 4.9 \end{matrix}$$

Давайте застосуємо до системи трансформацію Гауса:

$$A^T Ax = A^T b$$

Спочатку обрахуємо транспоновану матрицю A:

$$A^T = \begin{matrix} 1 & 0.99 & 0.98 & 0.97 & 0.97 \\ 0.99 & 1 & 0.99 & 0.98 & 0.97 \\ 0.98 & 0.99 & 1 & 0.99 & 0.98 \\ 0.97 & 0.98 & 0.99 & 1 & 0.99 \\ 0.96 & 0.97 & 0.98 & 0.99 & 1 \end{matrix}$$

Тепер обрахуємо $A^T A$, $A^T b$:

$$A^T A = \begin{matrix} 4.803 & 4.832 & 4.841 & 4.830 & 4.801 \\ 4.832 & 4.862 & 4.871 & 4.861 & 4.830 \\ 4.841 & 4.871 & 4.881 & 4.871 & 4.841 \\ 4.830 & 4.860 & 4.871 & 4.862 & 4.832 \\ 4.801 & 4.830 & 4.841 & 4.832 & 4.803 \end{matrix}$$

$$A^T b = \begin{matrix} 24.108 \\ 24.256 \\ 24.305 \\ 24.256 \\ 24.108 \end{matrix}$$

Оскільки ми отримали значення: $A^T A$, $A^T b$, може перейти до рішення ЗДВ виду динамічного методу першого порядку:

$$\frac{dx(t)}{dt} + \mathbf{A}^{\mathrm{T}}\mathbf{A}x(t) = \mathbf{A}^{\mathrm{T}}b, \;\; x(0) = 0$$

$$\frac{dx(t)}{dt} + \begin{pmatrix} 4.803 & 4.832 & 4.841 & 4.830 & 4.801 \\ 4.832 & 4.862 & 4.871 & 4.861 & 4.830 \\ 4.841 & 4.871 & 4.881 & 4.871 & 4.841 \\ 4.830 & 4.860 & 4.871 & 4.862 & 4.832 \\ 4.801 & 4.830 & 4.841 & 4.832 & 4.803 \end{pmatrix} x = \begin{pmatrix} 24.108 \\ 24.256 \\ 24.305 \\ 24.256 \\ 24.108 \end{pmatrix}$$

Для вирішиння цієї СЛАР використаємо метод Булірша-Штера. Реалізуємо алогоритм мовою програмування Python.

Код на мові програмування Python:

```
import numpy as np

ATA = np.array([
    [4.803, 4.832, 4.841, 4.830, 4.801],
    [4.832, 4.862, 4.871, 4.861, 4.830],
    [4.841, 4.871, 4.881, 4.871, 4.841],
    [4.830, 4.861, 4.871, 4.862, 4.830],
    [4.801, 4.830, 4.841, 4.830, 4.803]
])

ATb = np.array([24.108, 24.256, 24.305, 24.256, 24.108])

def system(t, x):
    return np.dot(-ATA, x) + ATb

def bulirsch_stoer(f, x0, t0, t_end, h_init, tol, max_iter=10000):
    def modified_midpoint(f, t, x, h, m):
        x_mid = x + 0.5 * h * f(t, x)
        x_last = x
        for i in range(1, m):
            x_next = x_last + h * f(t + i * h, x_mid)
            x_last = x_mid
            x_mid = x_next
```

```
    return x_mid

def richardson_extrapolation(results, m):
    for i in range(1, len(results)):
        factor = (m / (m - 1)) ** (2 * i)
        results[i:] = results[i:] + (results[i:] - results[:-i]) / (factor - 1)
    return results[-1]

t = t0
x = np.array(x0)
h = h_init
iterations = 0

while t < t_end and iterations < max_iter:
    if t + h > t_end:
        h = t_end - t

    m = 2
    x_m = [modified_midpoint(f, t, x, h, m)]
    error = float('inf')

    max_inner_steps = 10
    inner_steps = 0

    while error > tol and len(x_m) < max_inner_steps:
        m += 1
        x_m.append(modified_midpoint(f, t, x, h, m))
        error = np.linalg.norm(x_m[-1] - x_m[-2], ord=2)
        inner_steps += 1

    if inner_steps >= max_inner_steps:
        print(f"Warning: Inner loop did not converge at t = {t:.2f}")
```

```
        x_next = richardson_extrapolation(np.array(x_m), m)

        x = x_next
        t += h
        iterations += 1
        if error > 0:
            h = h * min(1.5, max(0.5, (tol / error) ** 0.5))
        else:
            h = h * 1.5

    return x, iterations

x0 = np.zeros(ATA.shape[0])  # x(0) = 0
t0 = 0
t_end = 10
h_init = 0.1
tol = 1e-6

final_x, iterations = bulirsch_stoer(system, x0, t0, t_end, h_init, tol)

print("Final solution x(t):", final_x)
print("Number of iterations for convergence:", iterations)
```

Принцип алгоритму:

Спочатку задається:

- Початковий час t_0 і кінцевий час t_{end}.
- Початковий вектор x_0, наприклад, x(0)=[0,0,0,0,0].
- Початковий крок інтегрування h_{init} (наприклад, h=0.1h=0.1).
- Допустима похибка tol.

Алгоритм використовує метод середньої точки для обчислення проміжних значень x із заданим кроком h. Цей метод виконує декілька "малих кроків" mmm (поділок), щоб уточнити розв’язок. Кількість цих малих кроків mmm поступово збільшується, поки розрахунки не стануть досить точними (tol).

Після того, як для різних m обчислено наближення, використовується екстраполяція для уточнення. Алгоритм оцінює похибку між розрахунками на різних m. Якщо похибка занадто велика, алгоритм зменшує крок h і повторює розрахунки. Якщо навпаки похибка дуже мала то алгоритм збільшуєш h.

Алгоритм повторює кроки інтегрування, доки час t не досягне t_{end}.

Результат роботи програми:

```
Final solution x(t): [2.33084842e+12 2.34525570e+12 2.34999313e+12 2.34506311e+12
 2.33065463e+12]
Number of iterations for convergence: 10000
```

Рис 1.1 – Результат роботи програми.

Як бачимо результати не збіглись з раніше обраним вектором x.

Тепер давайте обрахуємо за допомогою динамічного методу першого порядку, розробимо метод за допомогою мови програмування Python.

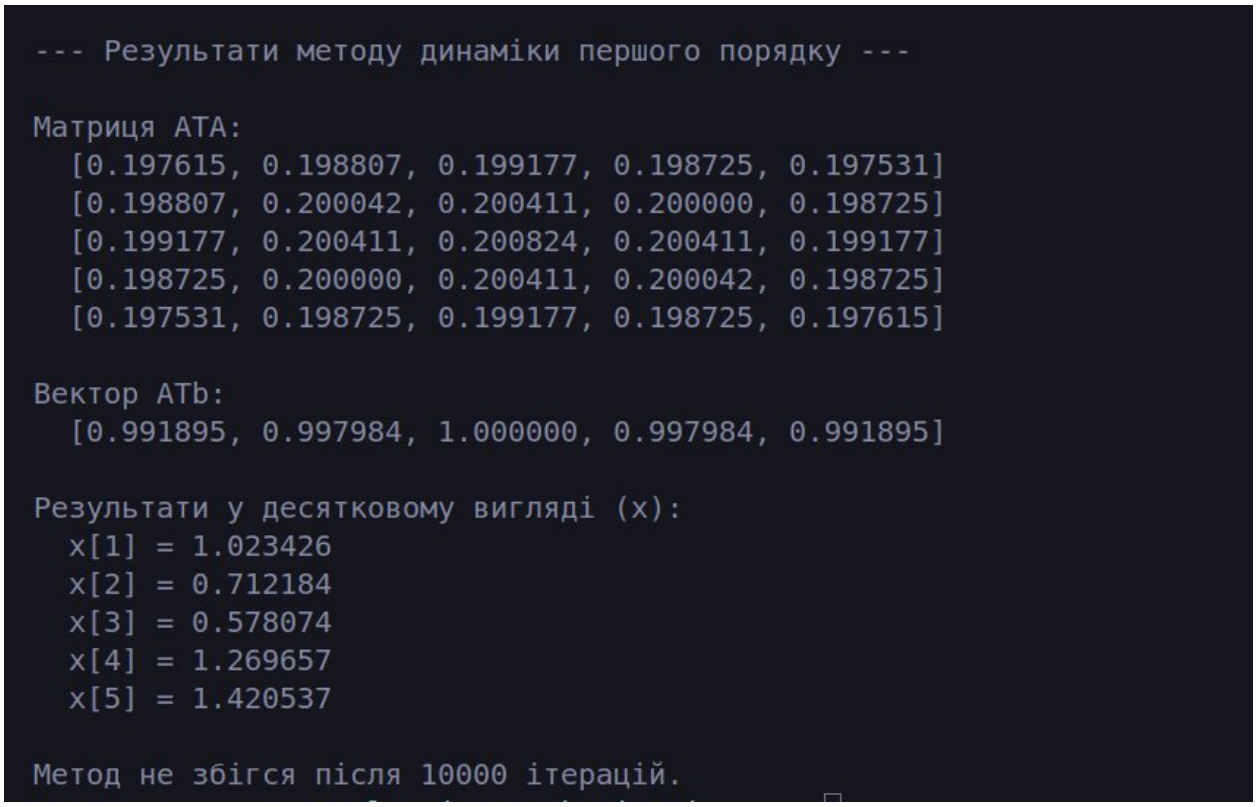
Розроблена програма на мові програмування Python:

```
import numpy as np

def compute_residual(ATA, ATb, x):
    return np.linalg.norm(np.dot(ATA, x) - ATb)

def first_order_dynamic_method(ATA, ATb, x0, dt, tolerance, max_iter=10000,
regularize=True, normalize=True):
    if normalize:
        ATA /= np.linalg.norm(ATA, ord=np.inf)
        ATb /= np.linalg.norm(ATb, ord=np.inf)

    if regularize:
        lambda_reg = 1e-6
        ATA += lambda_reg * np.eye(ATA.shape[0])

    x_current = x0
    iteration = 0
    residual = compute_residual(ATA, ATb, x_current)

    print(f"Початковий резидуал: {residual:.6e}")

    while residual > tolerance and iteration < max_iter:
        Ax = np.dot(ATA, x_current)
        dx_dt = ATb - Ax
        x_current = x_current + dx_dt * dt

        residual = compute_residual(ATA, ATb, x_current)
        iteration += 1

        if iteration % 50 == 0:
            print(f"Ітерація {iteration}: Резидуал = {residual:.6e}, x = {x_current}")
```

```
        if np.isnan(residual) or np.isnan(x_current).any() or residual > 1e20:
            print(f"Виявлено числову нестабільність на ітерації {iteration}")
            break

    converged = residual <= tolerance
    return x_current, residual, iteration, converged

def convert_matrix_to_str(matrix):
    return [[f"{val:.6f}" for val in row] for row in matrix]

def convert_vector_to_str(vector):
    return [f"{val:.6f}" for val in vector]

# Оновлені матриці
ATA = np.array([
    [4.803, 4.832, 4.841, 4.830, 4.801],
    [4.832, 4.862, 4.871, 4.861, 4.830],
    [4.841, 4.871, 4.881, 4.871, 4.841],
    [4.830, 4.861, 4.871, 4.862, 4.830],
    [4.801, 4.830, 4.841, 4.830, 4.803]
])

ATb = np.array([24.108, 24.256, 24.305, 24.256, 24.108])

x0 = np.zeros(ATA.shape[0])
dt = 0.8
tolerance = 1e-6

x, residual, iterations, converged = first_order_dynamic_method(ATA, ATb, x0,
dt, tolerance)
```

```
print("\n--- Результати методу динаміки першого порядку ---\n")

print("Матриця ATA:")
ATA_str = convert_matrix_to_str(ATA)
for row in ATA_str:
    print("  [" + ', '.join(row) + "]")

print("\nВектор ATb:")
ATb_str = convert_vector_to_str(ATb)
print("  [" + ', '.join(ATb_str) + "]")

print("\nРезультати у десятковому вигляді (x):")
for i, val in enumerate(x):
    print(f"  x[{i+1}] = {val:.6f}")

if converged:
    print(f"\nМетод збігся після: {iterations} ітерацій.")
else:
    print(f"\nМетод не збігся після {iterations} ітерацій.")
```

Результат роботи програми:



Як видно на зображення результат не збігся, але був доволі близько.

Висновок:

У ході виконання лабораторної роботи було вивчено та реалізовано чисельні методи для розв’язання систем звичайних диференційних рівнянь (ЗДР), що моделюють системи лінійних алгебраїчних рівнянь (СЛАР). Основна увага була приділена динамічному методу першого порядку та алгоритму Булірша-Штера, що забезпечують високу точність при розв’язанні жорстких задач.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- Atkinson, K.E. An Introduction to Numerical Analysis. – New York: Wiley, 1989. – 712 p.
- Hairer, E., Nørsett, S.P., Wanner, G. Solving Ordinary Differential Equations I: Nonstiff Problems. – Berlin: Springer, 1993. – 528 p.
- Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P. Numerical Recipes: The Art of Scientific Computing. – Cambridge: Cambridge University Press, 2007. – 1235 p.