МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

**Б.П.ДОВГИЙ**
**Є.С.ВАКАЛ**
**І.В.ГАП'ЯК**
**О.В.ОБВІНЦЕВ**

# НАВЧАЛЬНИЙ ПОСІБНИК ІЗ ДИСЦИПЛІНИ "МЕТОДИ ОБЧИСЛЕНЬ"

**для студентів механіко-математичного факультету заочної форми навчання**

Київ – 2022

УДК 51-37:004.4(0.75.8)
Д 58

Рецензенти:
д-р тех. наук, проф. О. С. Лимарченко
канд. фіз.-мат. наук, доц. Г. М. Зражевський

*Рекомендовано до друку*
*вченою радою механіко-математичного факультету*
*(протокол № 3 від 28 жовтня 2022 року)*

**Довгий Б.П.**

Д58 Навчальний посібник із дисципліни “Методи обчислень” для студентів механіко-математичного факультету заочної форми навчання / Б. П. Довгий, Є. С. Вакал, І. В. Гап’як, О. В. Обвінцев. – К., 2022. – 182 с.

ISBN

Навчальне видання містить теоретичні відомості, індивідуальні завдання для самостійного розвя’зання та методичні рекомендації до їх виконання, опис модулів для реалізації числових методів із дисципліни “Методи обчислень”, що викладається студентам механіко-математичного факультету Київського національного університету імені Тараса Шевченка заочної форми навчання. За змістом видання поділено на теми згідно з навчальною програмою дисципліни. Головна увага приділяється питанням застосування числових методів до розв'язання задач, що виникають в галузі математики. Для їх глибшого і змістовнішого розуміння навчальне видання доповнено короткими поясненнями із відповідних розділів наближених обчислень та текстами програм. У посібнику як базовий інструмент написання програм обрано об'єктно-орієнтовану мову програмування високого рівня Python, в якій реалізовано низку модулів для забезпечення розв’язання студентами індивідуальних учбових завдань.

Для студентів і викладачів механіко-математичного факультету. Посібник буде корисним для фахівців з обчислювальної математики, які використовують у наукових дослідженнях систему програмування Python і модуль NumPy.

УДК 51-37:004.4(0.75.8)

# ПЕРЕДМОВА

Можна навести багато прикладів складних математичних задач, які моделюють реальні процеси в природничих, технічних, економічних, соціальних та інших областях, але для більшості таких задач проблематично знайти аналітичні (точні) розв’язки, оскільки можливості сучасного математичного апарату також мають обмеження. В таких випадках на допомогу приходять наближені або числові методи. Навіть для знайденого розв’язку в явному вигляді при практичних застосуваннях необхідно отримувати числові характеристики розв’язку, його графічну візуалізацію, досліджувати залежності розв’язку від певних параметрів і т.п., що приводить до застосування ЕОМ і, як наслідок, до використання числових методів. Таким чином, багатьом фахівцям, а особливо математикам, необхідно володіти як теоретичними знаннями, так і практичними навичками в цьому розділі математики.

На механіко-математичному факультеті числові методи викладаються в курсі "Методи обчислень", який є нормативним для студентів, що навчаються за спеціальністю "Математика". Основною метою курсу є ознайомлення студентів із класичними методами наближеного розв'язання різноманітних математичних задач та розвинення практичних навичок застосування цих методів до розв'язання конкретних учбових задач на ЕОМ. Згідно з навчальним планом курс "Методи обчислень" вивчається в першому семестрі IV курсу. Під час навчання передбачено лекційний матеріал та проведення практичних робіт, але при цьому потрібно враховувати специфіку навчання на заочному відділенні – малу кількість годин, досить великий інформаційний об’єм курсу і необхідність розв’язання задач з використанням ПЕОМ.

Враховуючи значний обсяг теоретичного матеріалу, а також короткий термін його засвоєння і було написано даний методичний посібник, який має слугувати для ефективного досягнення мети курсу і максимальної комплексної допомоги студентам в оволодінні необхідною інформацією.

Методичний посібник за смисловим змістом поділено на такі розділи :

- ✓ теоретична частина з поділом на теми зі стислим описом класичних числових методів для розв’язання СЛАР, нелінійних рівнянь і систем, інтерполювання, числового диференціювання та інтегрування, розв’язання задачі Коші і крайових задач для ЗДР, крайових задач для основних типів рівнянь математичної фізики;
- ✓ деталізація розрахункових формул і алгоритмів деяких тем курсу, які традиційно "складні" для студентів;
- ✓ набір простих учбових індивідуальних завдань для самостійної роботи студентів, який охоплює всі теми курсу;
- ✓ створене ПЗ для ПЕОМ – у вигляді модулів на мові Python з реалізацією числових методів за темами курсу. Це полегшує студентам розв'язання поставлених задач, оскільки реалізація методів вимагає значних зусиль, часу та навичок при їх програмуванні. Зауважимо, що ця частина посібника буде корисна і при опану–ванні теми "Наукові обчислення" з курсу "Прикладне програмування".

При складанні посібника автори використовували оригінальні матеріали, що пропонувались протягом багатьох років студентам механіко-математичного факультету в нормативних курсах "Методи обчислень", "Рівняння математичної фізики", "Методи математичної фізики", "Інформаційні системи та технології"; курсах на базі Python – "Інформатика та програмування", "ООП", спеціальних курсах [1–2, 4–6, 9–14, 19], а також відомі підручники та монографії [3, 7–8, 15–18].

# ТЕМА 1

# ТОЧНІ МЕТОДИ РОЗВ'ЯЗАНИЯ СИСТЕМ АЛГЕБРАЇЧНИХ РІВНЯНЬ (СЛАР): LU-РОЗКЛАД, МОНОТОННА ПРОГОНКА. ОБЧИСЛЕННЯ ВИЗНАЧНИКА МАТРИЦІ DET(A), ОБЕРНЕНОЇ МАТРИЦІ, ЧИСЛА ОБУМОВЛЕНОСТІ МАТРИЦІ COND(A)

## *LU*-розклад матриці A для розв'язання СЛАР *Ax=f*

**Теорема 1.1**. Якщо $\det(A)\neq 0$, то існує матриця *перестановок* $P$ така, що $P\cdot A$ має відмінні від 0 *кутові мінори* :

$$\Delta_1 = a_{1,1},\ \ \Delta_2 = \begin{vmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{vmatrix},\ \ ...,\ \ \Delta_n = \det(A),\ \ \Delta_i \neq 0,\ \ i=\overline{1,n}. \tag{1.1}$$

Елементарна матриця перестановок визначається як: $P_{k,j} = (E)_{j\leftrightarrow k \text{ рядки}}$.

**Теорема 1.2**. Нехай всі кутові мінори матриці $A$ відмінні від 0, тоді матрицю $A$ однозначно можна представити у вигляді добутку двох трикутних матриць [18, с. 55-60]:

$$A = L\cdot U,\quad L = \begin{pmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{pmatrix},\quad U = \begin{pmatrix} 1 & u_{1,2} & \cdots & u_{1,n} \\ 0 & 1 & \cdots & u_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}. \tag{1.2}$$

Таким чином, розв'язування СЛАР $A\cdot\vec{x}=\vec{f}$ можна реалізувати у вигляді послідовності наступних кроків:

$$A \ \Rightarrow\ [P,\ L,\ U], \tag{1.3}$$

$$U\cdot\vec{x}=\vec{b},\ \ \vec{b}=L^{-1}P\cdot\vec{f}\,. \tag{1.4}$$

Знаходження розкладу (1.3) називається *прямим ходом*, а (1.4) – *зворотним ходом методу Гаусса*. З точки зору ефективності чисельних обчислень у алгоритм (1.3), (1.4) потрібно додати процедуру зменшення впливу *похибок заокруглень*.

Для реалізації (1.3)–(1.4) використовуються:

**А) Алгоритм без явного знаходження матриць *P*, *L*** – *метод Гаусса з вибором провідного елемента по стовпцю*.

Для $k$ – стовпця матриці $A^{(k-1)}$, $(A^{(0)}=A)$, де $k=1,2,...,n-1$,

1) знаходимо $i_k$ – номер рядка, в якому знаходиться *провідний елемент* $\left|a_{i_k,k}^{(k-1)}\right| = \max\limits_{k\le i\le n}\left(\left|a_{i,k}^{(k-1)}\right|\right) \,\&\, a_{i_k,k}^{(k-1)}\neq 0$;
2) якщо $i_k\neq k$, то виконуємо перестановку рядків $i_k$, $k$ множенням на відповідну елементарну матрицю перестановок $P_k = P_{k,i_k},\ \ P_k\cdot A^{(k-1)} = P_k\cdot\vec{f}^{\,(k-1)}$;
3) виключаємо $x_k$ з рядків $i=\overline{k+1,n}$ множенням на відповідну елементарну трикутну матрицю $L_k\cdot P_k\cdot A^{(k-1)} = L_k\cdot P_k\cdot\vec{f}^{\,(k-1)}$;

де

$$L_k=\begin{pmatrix}1&0&\cdots&0&\cdots&\cdots&0\\0&\ddots&\ddots&\vdots&\cdots&\cdots&\vdots\\0&\cdots&l_{k,k}&0&\cdots&\cdots&0\\0&\cdots&l_{k+1,k}&1&\ddots&\cdots&0\\0&\cdots&l_{k+1,k}&0&1&\ddots&0\\0&\cdots&\vdots&\vdots&\cdots&\ddots&\vdots\\0&\ldots&l_{n,k}&0&\cdots&\cdots&1\end{pmatrix},\quad l_{i,k}=\begin{cases}\dfrac{1}{a_{k,k}^{(k-1)}},i=k,\\-\dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}},i=\overline{k+1,n}.\end{cases}$$

У результаті об'єднання проведених обчислень (прямий хід методу Гаусса), отримуємо (1.4), де

$$U=L_{n-1}\cdot P_{n-1}\left(\cdot\ldots\left(L_2\cdot P_2\left(L_1\cdot P_1\cdot A\right)\right)\ldots\right),$$
$$\vec{b}=\vec{f}^{\,(n-1)}\equiv P_{n-1}\cdot\ldots\cdot P_2\cdot P_1\cdot\vec{f}.$$

Знаходження компонент $x_k$ (зворотний хід методу Гаусса) здійснюється за формулою:

$$x_k=b_k-\sum_{i=k+1}^{n}u_{k,i}x_i,\quad k=n,n-1,\ldots,1.$$

**Б) Алгоритми з явним знаходження матриць *P*, *L*, *U*** – *методи факторизації.*

Схема *LU-методу* :

1) Знаходимо розклад матриці $A\ \Rightarrow\ [P,\ L,\ U]$;
2) Отримуємо дві СЛАР з трикутними матрицями

$$(L\cdot U)\cdot\vec{x}=\vec{b},\ \ \vec{b}=P\cdot\vec{f},\quad\begin{cases}L\cdot\vec{y}=\vec{b},\\U\cdot\vec{x}=\vec{y}.\end{cases}\tag{1.5}$$

Як методи факторизації, так і метод Гаусса, мають певні варіанти реалізації, пов'язані зі специфікою матриці СЛАР :

a) якщо в (1.5) $P=E$, то такий варіант назвемо *простим LU-методом*, у якому елементи матриць $L,U$ визначаються наступним чином: для $i=\overline{1,n}$ спочатку обчислюємо *i*-й стовпець матриці $L$, а потім *i*-й рядок матриці $U$ за формулами

$$l_{i,j}=a_{i,j}-\sum_{k=1}^{j-1}l_{i,k}u_{k,j},\ \ j\le i;\qquad u_{i,j}=\frac{1}{l_{i,i}}\left(a_{i,j}-\sum_{k=1}^{i-1}l_{i,k}u_{k,j}\right),\ \ j>i.$$

Елементи векторів $\vec{y},\ \vec{x}$ знаходимо за формулами

$$y_i=\frac{1}{l_{1,1}}\left(f_i-\sum_{k=1}^{i-1}l_{i,k}y_k\right),\ i=\overline{1,n};\quad x_i=y_i-\sum_{k=i+1}^{n}u_{i,k}x_k,\ i=\overline{n,1}.$$

При реалізації цього методу можна не вводити матриці $L,U$, а використовувати тільки матрицю $A$ (враховуючи, що $u_{i,i}=1$).

b) якщо матриця $A$ є ермітовою ($A=A^*$) або симетричною ($A=A^{'}$), то такий варіант *LU-методу* має назву *методу квадратного кореня (метод Холецького).* Для ермітової матриці маємо $A=L\cdot U\equiv S^*\cdot(D\cdot S)$, де *S* верхня трикутна матриця з додатними елементами по діагоналі, *D* – діагональна матриця з $d_{i,i}=\pm1$. Розрахункові формули: для $i=\overline{1,n}$ обчислюємо

$$z=a_{i,i}-\sum_{k=1}^{i-1}\left|s_{k,i}\right|^2d_{k,k},\ d_{i,i}=sign(z),\ s_{i,i}=\sqrt{|z|},\ z=1/\left(s_{i,i}d_{i,i}\right);$$
$$s_{i,j}=z\cdot\left(a_{i,j}-\sum_{k=1}^{i-1}\bar{s}_{k,i}s_{k,j}d_{k,k}\right),\ j=\overline{i+1,n}.$$

У випадку $A = A^{'}, A > 0$ всі діагональні елементи матриці *D* рівні 1, тому розрахунки спрощуються і можна користуватись тільки верхньою трикутною частиною матриці $A$, крім того, на її місці можна будувати і матрицю *S*.

с) якщо $A$ – *тридіагональна* матриця

$$\begin{cases} a_i x_{i-1} - c_i x_i + b_i x_{i+1} = -f_i, \\ i = \overline{1,n}, \quad a_1 = b_n = 0, \end{cases} \tag{1.6}$$

для якої виконуються умови

$$\begin{cases} |c_i| \geq |a_i| + |b_i|, \quad i = \overline{1,n}; \\ a_i \neq 0, \quad i = \overline{2,n}; \\ b_i \neq 0, \quad i = \overline{1,n-1}, \end{cases} \tag{1.7}$$

то такий варіант методу Гаусса має назву *методу монотонної правої прогонки* [16, с. 73-75], і реалізація прямого ходу відбувається за рекурентними формулами:

$$\alpha_1 = \beta_1 = 0; \;\; z = c_i - a_i \alpha_i, \;\; \alpha_{i+1} = b_i / z, \;\; \beta_{i+1} = (f_i + a_i \beta_i)/z, \;\; i = \overline{1,n}, \tag{1.8}$$

а зворотного ходу – за рекурентними формулами:

$$x_n = \beta_{n+1}; \;\; x_{i-1} = \alpha_i x_i + \beta_i, \;\; i = n, n-1, ..., 2. \tag{1.9}$$

При виконанні умов (1.7) прогонка (1.8),(1.9) є *стійкою* (виконується умова $|\alpha_i| \leq 1, \;\; i = \overline{2,n}$).

Для випадку (1.6) є варіант *немонотонної прогонки* [15, с. 93-96], в якому пошук оптимального елемента виконується по рядку.

Для обчислення $\det(A)$ достатньо тільки прямого ходу методу Гаусса – привести матрицю $A$ до трикутної форми $U$ (1.4) і знайти $d = \det(L) = \prod_{i=1}^{n} l_{i,i}$ (у випадку матриці перестановок $P \neq E$ знак числа *d* потрібно додатково коригувати $(-1)^p \times d$, де *p* – кількість виконаних перестановок).

Для обчислення $A^{-1}$, як правило, використовують методи факторизації з системою правих частин, де вектор-стовпець $\vec{b}$ послідовно приймає вектор-стовпці матриці $E$.

## Оцінка похибки розв'язування СЛАР. Число *cond*(A)

Припустимо, що похибки, які вносяться у вхідні дані, малі.

***Означення 1.1.*** Під *обумовленістю* обчислювальної задачі розуміють чуттєвість (залежність) її розв'язку до вхідних даних, а коефіцієнт, який зв'язує ці похибки, називають *мірою обумовленості задачі* [18, с. 74-81].

Визначимо його для СЛАР :

$$A\vec{x} = \vec{f}, \quad \det(A) \neq 0. \tag{1.12}$$

Для (1.12) можна отримати

$$\vec{x} = A^{-1} \cdot \vec{f}. \tag{1.13}$$

Вважаючи $A^{-1}$, $\vec{f}$ змінними, формально продиференціюємо (1.13) :

$$\begin{aligned} d\vec{x} = dA^{-1} \cdot \vec{f} + A^{-1} \cdot d\vec{f} = \left[ A \cdot A^{-1} = E, \; dA \cdot A^{-1} + A \cdot dA^{-1} = 0, \; dA^{-1} = -A^{-1} \cdot dA \cdot A^{-1} \right] = \\ = -A^{-1} \cdot \left( dA \cdot \left( A^{-1} \cdot \vec{f} \right) - d\vec{f} \right) = -A^{-1} \cdot \left( dA \cdot \vec{x} - d\vec{f} \right). \end{aligned} \tag{1.14}$$

В останній рівності перейдемо до норм, використаємо відповідні нерівності, в результаті чого отримаємо:

$$\|d\vec{x}\| \le \|A^{-1}\| \cdot \left( \|dA\| \cdot \|\vec{x}\| + \|d\vec{f}\| \right). \tag{1.15}$$

Враховуючи співвідношення

$$\vec{f} = A \cdot \vec{x}, \quad \|\vec{f}\| \le \|A\| \cdot \|\vec{x}\|, \quad \|\vec{x}\| \ge \frac{\|\vec{f}\|}{\|A\|},$$

і вводячи відносні похибки вхідних даних і розв'язку

$$\varepsilon_A = \frac{\|dA\|}{\|A\|}, \quad \varepsilon_f = \frac{\|d\vec{f}\|}{\|\vec{f}\|}, \quad \varepsilon_x = \frac{\|d\vec{x}\|}{\|\vec{x}\|},$$

отримаємо:

$$\frac{\|d\vec{x}\|}{\|\vec{x}\|} \le \|A^{-1}\| \cdot \left( \|dA\| + \frac{\|d\vec{f}\|}{\|\vec{x}\|} \right) \le \|A^{-1}\| \cdot \left( \|dA\| + \frac{\|d\vec{f}\|}{\|\vec{b}\|} \cdot \|A\| \right) \le \|A^{-1}\| \cdot \|A\| \cdot \left( \frac{\|dA\|}{\|A\|} + \frac{\|d\vec{f}\|}{\|\vec{b}\|} \right),$$

$$\varepsilon_x \le cond(A) \cdot \left( \varepsilon_A + \varepsilon_f \right), \tag{1.16}$$

де $\mu_A = cond(A) = \|A\| \cdot \|A^{-1}\|$ – *число обумовленості* матриці $A$.

*Зауваження* 1. Матриці з $\mu_A >> 1$ – *погано обумовлені.*

*Зауваження* 2. Детальніші дослідження дають наступний вираз для (1.16):

$$\varepsilon_x \le \frac{\mu_A}{1 - \mu_A \cdot \varepsilon_A} \cdot \left( \varepsilon_A + \varepsilon_b \right).$$

*Зауваження* 3. Для $\mu_A$ маємо наступні співвідношення:

$1^o \quad \mu_A \ge 1;$

$2^o \quad \mu_A \ge \dfrac{\max\limits_i |\lambda_i(A)|}{\min\limits_i |\lambda_i(A)|};$

$3^o \quad \mu(A \cdot B) \le \mu(A) \cdot \mu(B).$

*Зауваження* 4. Простою ознакою поганої обумовленості матриці є велика різниця норм вектор-стовпців / вектор-рядків. Загальне правило дій у цьому випадку таке: за допомогою множення стовпців / рядків на числа (степені 2) урівноважуємо ці норми, масштабуємо невідомі / перетворюємо праві частини. Після цього застосовуємо методи типу Гаусса. Якщо цього не достатньо, потрібно використовувати *регуляризацію* [7, с. 137; 13, с. 46-47].

# ТЕМА 2

# ІТЕРАЦІЙНІ МЕТОДИ РОЗВ'ЯЗАННЯ СЛАР: ЯКОБІ, ЗЕЙДЕЛЯ, МІНІМАЛЬНИХ НЕВ'ЯЗОК, НАЙШВИДШОГО СПУСКУ

При розв'язанні СЛАР великої розмірності прямі методи стають неефективними. У таких випадках доцільніше застосовувати наближені ітераційні методи.

## Ітераційні методи розв'язання СЛАР

Нехай маємо СЛАР

$$A\vec{x} = \vec{f}, \tag{2.1}$$

де $A$ має обернену матрицю $A^{-1}$. Представимо матрицю $A$ у вигляді $A = A_1 + D + A_2$, де $D = diag\,[a_{11}, \dots, a_{nn}]$, $A_1$ – нижня трикутна, $A_2$ – верхня трикутна матриці :

$$(A_1 + D + A_2)\vec{x} = \vec{f}, \quad D\vec{x} = -(A_1 + A_2)\vec{x} + \vec{f},$$

$$\vec{x} = -D^{-1}(A_1 + A_2)\vec{x} + D^{-1}\vec{f}\,. \tag{2.2}$$

**Метод Якобі.** Схема методу :

$$\vec{x}^{(k+1)} = -D^{-1}(A_1 + A_2)\vec{x}^{(k)} + D^{-1}\vec{f}. \tag{2.3}$$

Якщо $A = A'$, $A > 0$ і $|a_{ii}| > \sum_{j=u\neq i}^{n} |a_{ij}|$ $i = \overline{1, n}$, то ітераційний процес (2.3) збігається.

**Метод Зейделя**. Схема методу:

$$\vec{x}^{(k+1)} = -D^{-1}A_1\vec{x}^{(k+1)} - D^{-1}A_2\vec{x}^{(k)} + D^{-1}\vec{f}\,. \tag{2.4}$$

Умова закінчення ІП (2.3), (2.4) : $\left\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\right\|_c < \varepsilon$.

*Канонічна форма однокрокових ітераційних методів має вигляд* [18, с. 84-85]:

$$B_{(k+1)}\frac{\vec{x}^{(k+1)} - \vec{x}^{(k)}}{\tau_{k+1}} + A\vec{x}^{(k)} = \vec{f}\,. \tag{2.5}$$

Якщо $B_{(k+1)} = E$, то ітераційний процес (2.5) називають *явним*, інакше – *неявним*. Якщо $B_{(k+1)} = B$, то ітераційний процес називається с*таціонарним.*

**Метод простої ітерації**. Схема методу

$$\frac{\vec{x}^{(k+1)} - \vec{x}^{(k)}}{\tau} + A\vec{x}^{(k)} = \vec{f}\,. \tag{2.6}$$

**Метод Річардсона.** Схема методу

$$\frac{\vec{x}^{(k+1)} - \vec{x}^{(k)}}{\tau_{k+1}} + A\vec{x}^{(k)} = \vec{f}\,. \tag{2.7}$$

Якщо $A = A'$, $A > 0$, то відомі алгоритми побудови оптимальних ітераційних параметрів для (2.6), (2.7), які базуються на значеннях границь спектра власних значень матриці СЛАР. Для (2.6) – це *оптимальний лінійний ІП* ($\tau_{opt} = 2/(\lambda_{\min}(A) + \lambda_{\max}(A))$), а для (2.7) – *метод Річардсона з чебишовським набором параметрів,* які впорядковані певним чином [16, с. 269-283].

**Методи релаксації.**

**а)** Узагальнення методу Зейделя. Схема методу

$$(D + \omega A_1)\frac{\vec{x}^{(k+1)} - \vec{x}^{(k)}}{\omega} + A\vec{x}^{(k)} = \vec{f}\,. \tag{2.8}$$

При $0<\omega<1$ (2.8) називають *методом нижньої релаксації*, при $1<\omega<2$ – *методом верхньої релаксації*, при $\omega=1$ – *методом Зейделя*.

Якщо $A=A'$, $A>0$, то (2.8) збігається при $0<\omega<2$.

Запишемо (2.8) у покомпонентній формі:

$$\begin{cases} x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \omega\dfrac{f_i}{a_{ii}} - \omega\left(\sum\limits_{j=1}^{i-1}\dfrac{a_{ij}}{a_{ii}}x_j^{(k+1)} + \sum\limits_{j=i+1}^{n}\dfrac{a_{ij}}{a_{ii}}x_j^{(k)}\right), i=\overline{1,n}; \\ k=0,1,\ldots, \left\|\vec{x}^{(k+1)}-\vec{x}^{(k)}\right\|_c < \varepsilon. \end{cases} \tag{2.9}$$

**b)** Узагальнення методу Якобі. Схема методу

$$D\frac{\vec{x}^{(k+1)}-\vec{x}^{(k)}}{\omega} + A\vec{x}^{(k)} = \vec{f}.. \tag{2.10}$$

Загальної формули для обчислення $\omega=\omega_{opt}$ для довільної матриці $A$ не існує.

***Методи побудовані на основі варіаційних підходів.***

I варіант методів. Заміна знаходження розв'язку СЛАР розв'язком задачі на пошук мінімуму деякого функціоналу. Так, наприклад, розв'язання СЛАР (2.1) з матрицею $A=A^*$, $A>0$ можна замінити мінімізацією функціоналу $\Phi(\vec{x}) = (A\vec{x},\vec{x}) - 2(\vec{f},\vec{x})$, а для невиродженої матриці – мінімізацією функціоналу $\Phi(\vec{x}) = (A\vec{x}-\vec{f}, A\vec{x}-\vec{f})$.

II варіант методів. В ітераційних методах виду (2.5) параметри $\tau_{k+1}$ вибираються із умови мінімуму $D$-норми похибки $\left\|\vec{z}^{(k+1)}\right\|_D$ при відомому значенні $\left\|\vec{z}^{(k)}\right\|_D$. Тут введено позначення: $\|\vec{w}\|_D = \sqrt{(D\vec{w},\vec{w})}$, $\vec{z}^{(k)} = \vec{x}^{(k)} - \vec{x}^{точ}$, $D$ – матриця $D=D'$, $D>0$. Розглянемо найпростіші види таких ІП [18, с. 115-122]. Нехай в СЛАР (2.1) $A=A'$, $A>0$ і маємо стаціонарний ітераційний процес у вигляді (2.5) з $B=E$. Позначимо для кроку $k$ нев'язку через $\vec{\delta}^{(k)} = A\vec{x}^{(k)} - \vec{f}$, тоді (2.5) можна переписати у вигляді

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - \tau_{k+1}\vec{\delta}^{(k)}. \tag{2.11}$$

**Метод мінімальних нев'язок.** ММН називається ітераційний процес (2.11), в якому $\tau_{k+1}$ вибирається з умови $\min\left\|\vec{\delta}^{(k+1)}\right\|$ при заданому $\left\|\vec{\delta}^{(k)}\right\|$ (враховуємо, що $A\vec{z}^{(k)} = \vec{\delta}^{(k)}$). Помножимо (2.11) на $A$ і віднімемо з обох частинах рівності $\vec{f}$:

$$A\vec{x}^{(k+1)} - \vec{f} = A\vec{x}^{(k)} - \vec{f} - \tau_{k+1}A\vec{\delta}^{(k)}, \quad \vec{\delta}^{(k+1)} = \vec{\delta}^{(k)} - \tau_{k+1}A\vec{\delta}^{(k)},$$

$$\left\|\vec{\delta}^{(k+1)}\right\|^2 = (\vec{\delta}^{(k)} - \tau_{k+1}A\vec{\delta}^{(k)}, \vec{\delta}^{(k)} - \tau_{k+1}A\vec{\delta}^{(k)}) = \left\|\vec{\delta}^{(k)}\right\|^2 - 2\tau_{k+1}(\vec{\delta}^{(k)}, A\vec{\delta}^{(k)}) + \tau_{k+1}^2\left\|A\vec{\delta}^{(k)}\right\|^2$$

Звідси бачимо, що $\left\|\vec{\delta}^{(k+1)}\right\|$ досягає мінімуму, якщо $\tau_{k+1} = (A\vec{\delta}^{(k)}, \vec{\delta}^{(k)})\Big/\left\|A\vec{\delta}^{(k)}\right\|^2$.

Алгоритм переходу від наближення $\vec{x}^{(k)}$ до наступного $\vec{x}^{(k+1)}$:

$$\vec{\delta}^{(k)} = A\vec{x}^{(k)} - \vec{f},\ \vec{y} = A\vec{\delta}^{(k)},\ \tau_{k+1} = (\vec{y},\vec{\delta}^{(k)})/(\vec{y},\vec{y}),\ \vec{y} = \tau_{k+1}\vec{\delta}^{(k)},\ \vec{x}^{(k+1)} = \vec{x}^{(k)} - \vec{y}.$$

**Метод найшвидшого спуску.** МНС відрізняється від ММН тільки формулою обчислення $\tau_{k+1}$. Це значення знаходять з умови $\min \|\vec{z}^{(k+1)}\|_A$ при заданому $\|\vec{z}^{(k)}\|_A$, що еквівалентно $\vec{\delta}^{(k)} \perp \vec{\delta}^{(k+1)}$. Помножимо (2.11) скалярно на $\vec{\delta}^{(k)}$, отримаємо $0 = (\vec{\delta}^{(k)},\vec{\delta}^{(k)}) - \tau_{k+1}(A\vec{\delta}^{(k)},\vec{\delta}^{(k)})$, звідси $\tau_{k+1} = (\vec{\delta}^{(k)},\vec{\delta}^{(k)})/(A\vec{\delta}^{(k)},\vec{\delta}^{(k)})$.

За умову закінчення ітераційних процесів ММН, МНС зазвичай використовують наступну: $\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\|_c < \varepsilon$.

Швидкість збіжності ММН, МНС така, як і у процеса (2.6) з оптимальним параметром $\tau$.

# ТЕМА 3

# РОЗВ'ЯЗКИ НЕЛІНІЙНИХ РІВНЯНЬ І СИСТЕМ РІВНЯНЬ: МЕТОД НЬЮТОНА ТА ЙОГО ВАРІАЦІЇ

## Ітераційні методи розв'язання нелінійних рівнянь

Знаходження коренів нелінійного рівняння

$$f(x)=0 \tag{3.1}$$

здійснюється у два етапи [7, с. 138-147].

На першому етапі досліджується розташування коренів (у загальному випадку на комплексній площині) і визначаються області в яких існує тільки один корінь і його кратність. Це дає нам початкове наближення до вибраного кореня.

На другому етапі в залежності від характеристик функції $f(x)$ будується певний ІП уточнення кореня.

У загальному вигляді ІП можна записати так

$$x^{(k+1)}=\Phi_{(k)}\left(x^{(k)},x^{(k-1)},...,x^{(k-m+1)}\right),\quad k=0,1,... \tag{3.2}$$

Для характеристики ІП (3.2) вводять такі поняття:

- *p-й порядок методу* – якщо виконується рівність $r^{(k+1)}=C\cdot(r^{(k)})^p$, де $C=const$, $r^{(k)}=\left|x^{точ}-x^{(k)}\right|$ – похибка;
- *стаціонарний метод* – якщо $\Phi_{(k)}$ не залежить від $k$, інакше – *нестаціонарний*;
- *m-кроковий* $(m\geq 1)$ – якщо для отримання чергового елементу послідовності використовують $m$ попередніх.

**Метод простої ітерації.** Спочатку рівняння (3.1) приводиться до еквівалентного

$$x=s(x) \tag{3.3}$$

і ІП проводиться за правилом

$$x^{(k+1)}=s(x^{(k)}),\quad k=0,1,... \tag{3.4}$$

Функція $s(x)$ зазвичай вибирається у вигляді

$$s(x)=x+\tau(x)f(x), \tag{3.5}$$

причому функція $\tau(x)$ не повинна змінювати знак в області пошуку кореня $U_r\left(x^{ôî÷}\right)=\{x:|\,x-x^{ôî÷}\,|\leq r\}$, а для $s(x)$ має виконуватись умова $|\,s'(x)|\leq q<1$ в області $U_r\left(x^{точ}\right)$.

Ураховуючи, що для збіжного ІП мають місце нерівності :

$$|\,x^{ôî÷}-x^{(k)}\,|\leq\frac{q}{1-q}|\,x^{(k)}-x^{(k-1)}\,|,\qquad |\,x^{ôî÷}-x^{(k)}\,|\leq\frac{q^k}{1-q}|\,x^{(1)}-x^{(0)}\,|,$$

умову закінчення ІП вибирають у вигляді $|\,x^{(k+1)}-x^{(k)}\,|\leq\varepsilon,\quad 0<\varepsilon<1$.

Ітераційний процес (3.4) має перший порядок (збігається лінійно зі швидкістю геометричної прогресії зі знаменником $q$).

На основі лінійного ІП можна побудувати нестаціонарний ІП, який має квадратичну збіжність. Це так званий **процес Ейткена-Стеффенсена**. Його алгоритм: за відомим $x^{(k)}$ циклічно $k=0,3,6,...,|\,x^{(k+3)}-x^{(k)}\,|\leq\varepsilon$, виконуємо наступні розрахунки

$$\begin{cases}x^{(k+1)}=s(x^{(k)}),\quad x^{(k+2)}=s(x^{(k+1)}),\\ x^{(k+3)}=x^{(k)}-(x^{(k)}-x^{(k+1)})^2(x^{(k)}-2x^{(k+1)}+x^{(k+2)})^{-1}.\end{cases} \tag{3.6}$$

**Метод Ньютона (лінеаризації).** Нехай функція $f(x)$ рівняння (3.1) двічі неперервно диференційована, тоді ІП

$$x^{(k+1)} = x^{(k)} - f(x^{(k)})(f'(x^{(k)}))^{-1}, \tag{3.7}$$

збігається до кореня (3.1) при $\forall\ x^{(0)}$, якщо скрізь $|f(x)\cdot f''(x)| < (f'(x))^2$.

Якщо $f'(x)$ і $f''(x)$ зберігають знак на $\Omega = U_r(x^{точ})$, а також виконується умова $f(x^{(0)})\cdot f''(x^{(0)}) \ge 0$, то ІП (3.7) збігається до кореня $x^{точ}$ монотонно і для похибки маємо нерівність $|x^{\text{ôî÷}} - x^{(k)}| \le 0.5 M_2 m_1^{-1} |x^{(k)} - x^{(k-1)}|^2$, $m_1 = \min\limits_{\Omega} |f'(x)|$, $M_2 = \max\limits_{\Omega} |f''(x)|$. ІП (3.7) має 2-й порядок. Виходячи із геометричних побудов, ІП (3.7) має назву ***методу дотичних***.

Для кореня $x^{точ}$ кратності *n*, ІП (3.7) записують у формі

$$f'(x^{(k)})(x^{(k+1)} - x^{(k)})/n + f(x^{(k)}) = 0.$$

При заміні у (3.7) похідної на поділену різницю отримаємо двокроковий ІП за схемою **методу січних** :

$$x^{(k+1)} = x^{(k)} - \left(x^{(k)} - x^{(k-1)}\right) f(x^{(k)}) \left(f(x^{(k)}) - f(x^{(k-1)})\right)^{-1} \tag{3.8}$$

та **методу Курчатова** :

$$x^{(k+1)} = x^{(k)} - 2\left(x^{(k)} - x^{(k-1)}\right) f(x^{(k)}) \left(f(2x^{(k)} - x^{(k-1)}) - f(x^{(k-1)})\right)^{-1}. \tag{3.9}$$

ІП (3.8), (3.9) мають порядок $p \approx 1.62$ і не вимагають обчислення похідної.

## Ітераційні методи розв'язання систем нелінійних рівнянь

Знаходження коренів системи *N* нелінійних рівнянь [7, с. 150-153]

$$\vec{f}(\vec{x}) = 0 \tag{3.10}$$

за допомогою ітераційних методів проводиться як і для одного рівняння (3.1).

**Метод простої ітерації**. Спочатку система (3.10) приводиться до еквівалентної

$$\vec{x} = \vec{s}(\vec{x}) \tag{3.11}$$

і ІП проводиться за правилом

$$\vec{x}^{(k+1)} = \vec{s}(\vec{x}^{(k)}), \quad k = 0,1,\dots\ . \tag{3.12}$$

Вектор-функцію $\vec{s}(\vec{x})$ потрібно вибрати так, щоб для матриці $S = \left[\partial s_i(x)/\partial x_j\right]_{i,j}^{N}$ в околі $U_r(\vec{x}^{точ})$ виконувалась умова $\|S\|_* \le q < 1$. Умову закінчення ІП (3.12) потрібно перевіряти для кожної компоненти: $|x_i^{(k+1)} - x_i^{(k)}| \le \varepsilon$, i = 1,2,...,N; $0 < \varepsilon < 1$.

**Метод Ньютона.** ІП проводиться за правилом (у векторній формі)

$$\mathbf{F}(\vec{x}^{(k)})\cdot\vec{\delta} = -\vec{f}(\vec{x}^{(k)}), \quad \vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{\delta}, \quad \mathrm{k} = 0,1,\dots, \tag{3.13}$$

де $\mathbf{F}(\vec{x}) = \left[\partial f_i(\vec{x})/\partial x_j\right]_{i,j=1}^{N}$ – матриця Якобі, а умовою закінчення є $\left\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\right\|_* \le \varepsilon,\ 0 < \varepsilon < 1$.

**Метод градієнта (метод найшвидшого спуску).** ІП проводиться за правилом (у векторній формі)

$$\begin{aligned} &\vec{z} = \vec{f}(\vec{x}^{(k)}),\ \mathbf{J} = \mathbf{F}(\vec{x}^{(k)}),\ \vec{\delta} = \mathbf{J}'\cdot\vec{z},\ \vec{v} = \mathbf{J}\cdot\vec{d}, \\ &\mu = \frac{(\vec{z},\vec{v})}{(\vec{v},\vec{v})},\ \vec{\delta} = \mu\cdot\vec{\delta},\ \vec{x}^{(k+1)} = \vec{x}^{(k)} - \vec{\delta}, \quad \mathrm{k} = 0,1,\dots, \end{aligned} \tag{3.14}$$

а умовою закінчення є $\left\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\right\|_* \le \varepsilon, \quad 0 < \varepsilon < 1$.

# ТЕМА 4

# ІНТЕРПОЛЮВАННЯ ФУНКЦІЙ: ФОРМУЛА ЛАГРАНЖА, НЬЮТОНА. СПЛАЙНИ НУЛЬОВОГО І ПЕРШОГО ПОРЯДКУ

## Наближення функції поліномом на всьому відрізку

Нехай на відрізку [$a$, $b$] задано вузли (різні) – *вузли інтерполяції*

$$a = X_1 < X_2 < \ldots < X_n = b. \tag{4.1}$$

Крім того, в цих вузлах відомі значення функції $f(X_i) = Y_i$. Необхідно побудувати *інтерполяційний поліном* (ІП)

$$P_{n-1}(x) = \sum_{k=1}^{n} p_k x^{k-1} \tag{4.2}$$

такий, що

$$P_{n-1}(X_i) = Y_i, \quad i = \overline{1,n}. \tag{4.3}$$

Враховуючи (4.2), (4.3), отримуємо для невідомих коефіцієнтів $p_k$, $k = \overline{1,n}$ СЛАР із визначником Вандермонда, який не дорівнює нулю (з огляду на властивість вузлів (4.1)), а отже ІП існує та єдиний. Крім "*поліноміальної*" форми представлення (4.2) існують й інші форми запису ІП [19, с. 6-30].

*Форма Лагранжа* дозволяє представити поліном (4.2) через значення функції у вузлах інтерполяції $f(X_i)$ у вигляді (форма "*лагранжіана*") [5, 11]

$$L_{n-1}(x) = \sum_{k=1}^{n} f(X_k) l_k(x). \tag{4.4}$$

Для знаходження поліномів $l_k(x)$ маємо: $\sum_{k=1}^{n} f(X_k) l_k(X_i) = f(X_i)$, $i = \overline{1,n}$, а отже $l_k(X_i) = \begin{cases} 1, k = i \\ 0, k \neq i \end{cases}$, звідки отримуємо $l_k(x) = \dfrac{\omega(x)}{(x - X_k)\omega'(X_k)}$, де $\omega(x) = \prod_{i=1}^{n} (x - X_i)$.

Запишемо формулу (4.4) детальніше:

$$L_{n-1}(x) = f(X_1)\frac{(x-X_2)(x-X_3)\cdots(x-X_n)}{(X_1-X_2)(X_1-X_3)\cdots(X_1-X_n)} + f(X_2)\frac{(x-X_1)(x-X_3)\cdots(x-X_n)}{(X_2-X_1)(X_2-X_3)\cdots(X_2-X_n)} + \cdots +$$
$$+ f(X_n)\frac{(x-X_1)(x-X_2)\cdots(x-X_{n-1})}{(X_n-X_1)(X_n-X_2)\cdots(X_n-X_{n-1})}.$$

*Форма Ньютона* є представленням полінома (4.2) у вигляді різницевого аналога формули Тейлора [7, с. 29-33].

***Означення 4.1.*** *Поділені різниці* 1-*го порядку* для функції $f(x)$ визначаються наступним чином

$$f(X_i; X_j) \equiv f_{i;j} = \frac{f(X_i) - f(X_j)}{X_i - X_j} = f_{j;i}, \quad (i \neq j). \tag{4.5}$$

***Означення 4.2.*** *Поділені різниці вищих порядків* визначаються рекурентно:

$$f(X_i; X_j; X_k) \equiv f_{i;j;k} = \frac{f_{i;j} - f_{j;k}}{X_i - X_k}, \ \ldots, f_{i;j;\ldots;s;k} = \frac{f_{i;j;\ldots;s} - f_{j;\ldots;s;k}}{X_i - X_k}. \tag{4.6}$$

Інтерполяційний поліном (4.2) у формі Ньютона має вид

$$N_{n-1}(x) = f(X_1) + (x - X_1) f_{1;2} + (x - X_1)(x - X_2) f_{1;2;3} + \ldots + (x - X_1)(x - X_2)\ldots(x - X_{n-1}) f_{1;2;\ldots;n}. \tag{4.7}$$

Похибка інтерполяції визначається як $r_{n-1}(x) = f(x) - P_{n-1}(x)$. Якщо $f(x) \in C^{(n)}[a,b]$, то маємо $|r_{n-1}(x)| = |f(x) - P_{n-1}(x)| \le \dfrac{M_n \cdot |\omega(x)|}{n!}$, $M_n = \max\limits_{x\in[a,b]} |f^{(n)}(x)|$.

Збіжності $|r_{n-1}(x)| \underset{n\to\infty}{\longrightarrow} 0$ у загальному випадку немає, вона суттєво залежить від вузлів сітки і гладкості функції $f(x)$ [7, с. 39-40].

## Кусково-поліноміальне (сплайнове) наближення функції

Для подолання негараздів, пов'язаних зі стійкістю, збіжністю, зростанням похибок і складності в обчисленнях при $n \to \infty$ розглянутого вище інтерполяційного процесу, було запропоновано будувати не один ІП на всьому відрізку [*a, b*], а використовувати *кусково-поліноміальну інтерполяцію*. У цьому випадку на кожному інтервалі $\Delta_i = [X_i, X_{i+1}]$ використовують поліном $S_{m,q}(x; f), x \in \Delta_i$ низького степеня $m \in \{0,1,2,3\}$. Коефіцієнти кожного з поліномів цієї системи визначають з умов інтерполяції на сітці та деяких додаткових умов, які відомі про функцію $f(x)$. Побудований у такий спосіб інтерполянт називають *сплайн-функцією* або просто *сплайном* [19, с. 9].

***Означення 4.3.*** Функцію $S_{m,q}(x), x \in [a,b]$ називають *сплайном степеня* $m > 0$ *дефекту* $q$ $(0 \le q \le m)$, якщо

а) на кожному відрізку $\Delta_i, i = \overline{1, n-1}$ вона є поліномом степеня $m$ ("*сплайнове*" представлення), тобто $S_{m,q}(x) = \sum\limits_{k=0}^{m} s_{i,k}(x - X_i)^k$, $x \in \Delta_i$;

б) функція $S_{m,q}(x)$ неперервна на $[a,b]$ разом зі своїми похідними до степеня $m-q$ включно

$$S_{m,q}(x) \in C^{m-q}([a,b]) \tag{4.7}$$

***Означення 4.4.*** Сплайн $S_{m,q}(x), x \in [a,b]$ називають *інтерполяційним* для $f(x)$ і позначають $S_{m,q}(x; f)$, якщо він задовольняє умову інтерполяції $S_{m,q}(X_i) = f_i$, $i = \overline{1, n}$.

Ми обмежемося розглядом сплайнів: $S_{0,1}(x; f)$ – нульового степеня (*кусково-стала функція*) і $S_{1,1}(x; f)$ – першого степеня (*кусково-лінійна функція*). Розрахункові формули для цих сплайнів записуються досить просто, а саме:

а) кусково-стала функція (для неї не виконується умова (4.7))

$$S_{0,1}(x; f) = \begin{cases} f_i, & x \in [X_i, X_{i+1}),\ i = \overline{0, n-1}; \\ f_n, & x = X_n. \end{cases} \tag{4.8}$$

b) кусково-лінійна функція

$$S_{1,1}(x; f) = s_{i,1}(x - X_i) + s_{i,0},\ x \in [X_i, X_{i+1}],\ s_{i,1} = f_{i;i+1},\ s_{i,0} = f_i,\ i = \overline{0, n-1}. \tag{4.9}$$

Перевага сплайнів перед звичайною інтерполяцією полягає в простоті обчислень полінома, стійкості процесу обчислень, збіжності.

Крім сплайнів нульового і першого порядку на практиці використовують *квадратичні* і *кубічні* сплайни. Сплайни використовують не тільки для задачі наближення функції, а і для задач числового диференціювання та інтегрування, числового розв'язання диференціальних рівнянь, крайових задач математичної фізики, інтегральних рівнянь тощо [19, с. 76-112].

# ТЕМА 5

# ЧИСЛОВЕ ДИФЕРЕНЦІЮВАННЯ: БАЗОВІ ФОРМУЛИ, ОЦІНКА ПОХИБКИ

## Числове диференціювання

*Постановка задачі.* Нехай

$$L_m(u(x)) = \sum_{j=0}^{m} a_j u^{(j)}(x) \tag{5.1}$$

лінійний диференціальний оператор $m$-го порядку. Необхідно його наближено обчислити в деякій точці $x_i \in \omega_h$, де $\omega_h$ – рівномірна або нерівномірна сітка.

*Схема розв'язання.* Наближаємо функцію $u(x)$ поліномом (наприклад, у формі Лагранжа) $L_n(x) \in \mathrm{P}_n(x)$, тоді для похідних у формулі (5.1) маємо $u^{(j)}(x) \approx L_n^{(j)}(x)$.

*Найпростіші* формули числового диференціювання (ФЧД) для $u'$, $u''$ отримують, вибираючи певний *шаблон* $S$ із точок рівномірної сітки $\omega_h$ (тут обрано шаблон із мінімально можливою кількістю точок, окрім центральної різницевої похідної $u_{\tilde{x},i}$) :

| **Шаблон** | **Похідна** | **Позначення і обчислення** | **Назва похідної** | **Точність** |
|---|---|---|---|---|
| $S[x_{i-1}, x_i]$ | $u'(x_i)$ | $u_{\bar{x},i} = \dfrac{u_i - u_{i-1}}{h}$ | *ліва різницева похідна* | $O(h^1)$ |
| $S[x_i, x_{i+1}]$ | $u'(x_i)$ | $u_{x,i} = \dfrac{u_{i+1} - u_i}{h}$ | *права різницева похідна* | $O(h^1)$ |
| $S[x_{i-1}, x_i, x_{i+1}]$ | $u'(x_i)$ | $u_{\tilde{x},i} = \dfrac{u_{i+1} - u_{i-1}}{2h}$ | *центральна різницева похідна* | $O(h^2)$ |
| $S[x_{i-1}, x_i, x_{i+1}]$ | $u''(x_i)$ | $u_{\bar{x}x,i} = \dfrac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$ | *друга різницева похідна* | $O(h^2)$ |

ФЧД для похідної $u'''$, яка зустрічається, наприклад, в рівнянні *Кортевега-де Фриза*, можна отримати і не користуючись вказаною вище *схемою*, а застосувати вже отримані різницеві похідні для $u'$, $u''$ на мінімальних шаблонах :

ліва різниця від другої різницевої похідної

$$u'''(x_i) \approx (u_{\bar{x}x})_{\bar{x},i} = \frac{u_{i+1} - 3u_i + 3u_{i-1} - u_{i-2}}{h^3}; \tag{5.2}$$

центральна різниця від другої різницевої похідної

$$\begin{aligned} u'''(x_i) &\approx (u_{\bar{x}x})_{\tilde{x},i} = \frac{u_{\bar{x}x,i+1} - u_{\bar{x}x,i-1}}{2h} = \\ &= \frac{h^{-2}(u_i - 2u_{i+1} + u_{i+2}) - h^{-2}(u_{i-2} - 2u_{i-1} + u_i)}{2h} = \frac{u_{i+2} - 2u_{i+1} + 2u_{i-1} - u_{i-2}}{2h^3}. \end{aligned} \tag{5.3}$$

Використаємо цей прийом для отримання ФЧД для $u^{\mathrm{IV}}$, яка зустрічається, наприклад, в рівнянні *вільної рівноваги пружного бруска*, в *бігармонічному* рівнянні :

друга різницева похідна від другої різницевої похідної

$$u^{\mathrm{IV}}(x_i) \approx (u_{\bar{x}x})_{\bar{x}x,i} = \frac{u_{\bar{x}x,i-1} - 2u_{\bar{x}x,i} + u_{\bar{x}x,i+1}}{h^2} = \frac{u_{i+2} - 4u_{i+1} + 6u_i - 4u_{i-1} + u_{i-2}}{h^4}. \tag{5.4}$$

Похідні більш високого порядку в модельних задачах зустрічаються не часто, тому розглядати їх не будемо.

Розглянемо отримання оцінки точності на прикладі 2-ої різницевої похідної $u_{\bar{x}x,i}$

$$u(x_i \pm h) = u(x_i) \pm hu'(x_i) + \frac{h^2}{2}u''(x_i) \pm \frac{h^3}{6}u'''(x_i) + \frac{h^4}{24}u^{(4)}(x_i) \pm \frac{h^5}{120}u^{(5)}(x_i) + O(h^6);$$

$$\psi(x_i) = \frac{u(x_i + h) - 2u(x_i) + u(x_i - h)}{h^2} - u''(x_i) = \frac{h^2}{12}u^{(4)}(x_i) + O(h^4) = O(h^2).$$

*Завдання 1.* Для ФЧД (5.2)-(5.4) отримати оцінку точності.

Для отримання вищої точності можна збільшити кількість точок шаблону або скористатися формулою *Рунге-Ромберга* [7, с. 74-79].

Нехай для обчислення $\eta(x)$ використовується деяка наближена розрахункова формула $\xi(x,h)$ з порядком точності $p$ (це має сенс не тільки для числового диференціювання !)

$$\eta(x) = \xi(x,h) + O(h^p) = \xi(x,h) + \varphi(x)\cdot h^p + O(h^{p+1}). \qquad (5.5)$$

Використовуючи дві зв'язані сітки з кроками $h_1 = h, h_2 = kh$, можна отримати для (5.5) наступні формули :

$$\varphi(x)\cdot h^p = \frac{\xi(x, k\cdot h) - \xi(x,h)}{1 - k^p}, \qquad (5.6)$$

$$\eta(x) = \xi(x,h) + \frac{\xi(x, k\cdot h) - \xi(x,h)}{1 - k^p}. \qquad (5.7)$$

Формула (5.6) – *перша формула Рунге-Ромберга*. Вона використовується для перевірки, чи знайдено результат з необхідною точністю $\varepsilon$. Формула (5.7) – *друга формула Рунге-Ромберга*. Вона використовується як для підвищення точності обчислень, так і для побудови нових розрахункових формул з порядком не нижче ніж $p+1$.

*Багатоточкові* ФЧД отримують, вибираючи *шаблон S* із *num_pnt* точок рівномірної сітки $\omega_h$. Подамо деякі з цих формул (у матричній формі) для наближення похідної [10, с. 31-32]:

**1.** першого порядку

$$num\_pnt = 3 \quad \begin{pmatrix} u'_{i-1} \\ u'_i \\ u'_{i+1} \end{pmatrix} = \frac{1}{2h}\begin{pmatrix} -3 & 4 & 1 \\ -1 & 0 & 1 \\ 1 & -4 & 3 \end{pmatrix}\begin{pmatrix} u_{i-1} \\ u_i \\ u_{i+1} \end{pmatrix} + \frac{h^2}{6}\begin{pmatrix} 2u'''(\xi_{i-1}) \\ -u'''(\xi_i) \\ 2u'''(\xi_{i+1}) \end{pmatrix};$$

$$num\_pnt = 4 \quad \begin{pmatrix} u'_{i-2} \\ u'_{i-1} \\ u'_i \\ u'_{i+1} \end{pmatrix} = \frac{1}{6h}\begin{pmatrix} -11 & 18 & -9 & 2 \\ -2 & -3 & 6 & -1 \\ 1 & -6 & 3 & 2 \\ -2 & 9 & -18 & 11 \end{pmatrix}\begin{pmatrix} u_{i-2} \\ u_{i-1} \\ u_i \\ u_{i+1} \end{pmatrix} + \frac{h^3}{12}\begin{pmatrix} -3u^{(4)}(\xi_{i-2}) \\ u^{(4)}(\xi_{i-1}) \\ -u^{(4)}(\xi_i) \\ 3u^{(4)}(\xi_{i+1}) \end{pmatrix};$$

$$num\_pnt = 5 \quad \begin{pmatrix} u'_{i-2} \\ u'_{i-1} \\ u'_i \\ u'_{i+1} \\ u'_{i+2} \end{pmatrix} = \frac{1}{12h}\begin{pmatrix} 35 & -104 & 114 & -56 & 11 \\ 11 & -20 & 6 & 4 & -1 \\ -1 & 16 & -30 & 16 & -1 \\ -1 & 4 & 6 & -20 & 11 \\ 11 & -56 & 114 & -104 & 35 \end{pmatrix}\begin{pmatrix} u_{i-2} \\ u_{i-1} \\ u_i \\ u_{i+1} \\ u_{i+2} \end{pmatrix} + \frac{h^4}{60}\begin{pmatrix} 12u^{(5)}(\xi_{i-2}) \\ -3u^{(5)}(\xi_{i-1}) \\ 2u^{(5)}(\xi_i) \\ -3u^{(5)}(\xi_{i+1}) \\ 12u^{(5)}(\xi_{i+2}) \end{pmatrix}.$$

**2.** другого порядку

$$num\_pnt = 3 \quad \begin{pmatrix} u''_{i-1} \\ u''_i \\ u''_{i+1} \end{pmatrix} = \frac{1}{h^2}\begin{pmatrix} 1 & -2 & 1 \\ 1 & -2 & 1 \\ 1 & -2 & 1 \end{pmatrix}\begin{pmatrix} u_{i-1} \\ u_i \\ u_{i+1} \end{pmatrix} - h\begin{pmatrix} u^{(3)}(\xi_{i-1}) \\ \frac{h}{12} * u^{(4)}(\xi_i) \\ -u^{(3)}(\xi_{i+1}) \end{pmatrix};$$

$$num\_pnt = 4 \quad \begin{pmatrix} u''_{i-2} \\ u''_{i-1} \\ u''_i \\ u''_{i+1} \end{pmatrix} = \frac{1}{h^2}\begin{pmatrix} 2 & -5 & 4 & -1 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ -1 & 4 & -5 & 2 \end{pmatrix}\begin{pmatrix} u_{i-2} \\ u_{i-1} \\ u_i \\ u_{i+1} \end{pmatrix} + \frac{h^2}{12}\begin{pmatrix} 11u^{(4)}(\xi_{i-2}) \\ -u^{(4)}(\xi_{i-1}) \\ u^{(4)}(\xi_i) \\ -11u^{(4)}(\xi_{i+1}) \end{pmatrix};$$

$$num\_pnt = 5 \quad \begin{pmatrix} u''_{i-2} \\ u''_{i-1} \\ u''_i \\ u''_{i+1} \\ u''_{i+2} \end{pmatrix} = \frac{1}{12h^2}\begin{pmatrix} 35 & -104 & 114 & -56 & 11 \\ 11 & -20 & 6 & 4 & -1 \\ -1 & 16 & -30 & 16 & -1 \\ -1 & 4 & 6 & -20 & 11 \\ 11 & -56 & 114 & -104 & 35 \end{pmatrix}\begin{pmatrix} u_{i-2} \\ u_{i-1} \\ u_i \\ u_{i+1} \\ u_{i+2} \end{pmatrix} + \frac{h^3}{180}\begin{pmatrix} -150u^{(4)}(\xi_{i-2}) \\ 15u^{(4)}(\xi_{i-1}) \\ h * 2u^{(5)}(\xi_i) \\ -15u^{(4)}(\xi_{i+1}) \\ 150u^{(4)}(\xi_{i+2}) \end{pmatrix}.$$

**3.** третього порядку

$$num\_pnt = 7 \quad u'''(x_i) \approx \frac{u_{i+3} - 8u_{i+2} + 13u_{i+1} - 13u_{i-1} + 8u_{i-2} - u_{i-3}}{8h^3}.$$

**4.** четвертого порядку

$$num\_pnt = 7 \quad u^{\mathrm{IV}}(x_i) \approx \frac{-u_{i+3} + 12u_{i+2} - 39u_{i+1} + 56u_i - 39u_{i-1} + 12u_{i-2} - u_{i-3}}{6h^4}.$$

Багатоточкові формули використовують не тільки для числового диференціювання, а і для побудови алгоритмів розв'язання задачі Коші для звичайних диференціальних рівнянь, крайових задач математичної фізики тощо.

Як бачимо, розрахункові формули *p*-го порядку точності для апроксимації $u^{(j)}(x)$ на багатоточковому шаблоні $S[x_0, x_1, ..., x_k]$ можна представити в наступній формі [7, с. 81-82] :

$$u^{(j)}(x) \approx R_{j,k,p}(x) = \psi_1(h) + \psi_2(h), \ \psi_1(h) = h^{-j}\sum_{i=0}^{k} c_i u(x_i), \ \psi_2(h) = O(h^p). \qquad (5.8)$$

Якщо аналізувати (5.8) на предмет комплексної оцінки поведінки *похибки методу* $\psi_2(h)$ і *похибки обчислення* $u(x_i)$ при розрахунках $\psi_1(h)$ (це можуть бути заокруглення, неусувні похибки) в залежності від параметра *h*, то враховуючи поведінку $\psi_2(h) \approx \widetilde{C}_2 \cdot h^p$ і $\psi_1(h) \approx \widetilde{C}_1 \cdot h^{-j}$ можна знайти значення $h = h_{opt}$ для конкретних значень *j*, *k*, *p* і $e_d$ – *похибки вхідних даних* ($u(x)$) формули $R_{j,k,p}(x)$. Це означає, що ми можемо підвищувати точність обчислення $R_{j,k,p}(x)$ зменшуючи *h*, але не допускати $h < h_{opt}$, яке приведе до неправильних результатів обчислення $u^{(j)}(x)$.

Розглянута ситуація пов'язана з *некоректністю задачі числового диференціювання*. Поясним це за допомогою такого прикладу [7, с. 82] : нехай маємо похибку вхідних даних виду $\delta u(x) = m^{-1} \cdot \sin(m^2 x)$, очевидно для похибки першої похідної будемо мати вираз $\delta u'(x) = m \cdot \cos(m^2 x)$. При $m \to \infty$, в нормі $\|\circ\|_C$, похибка функції необмежено спадає, а похибка похідної – необмежено зростає ! Отже, відсутня неперервна залежність похідної від функції, а це і є *некоректність диференціювання*. Зокрема це досить сильно впливає на знаходження похідних високого порядку.

# ТЕМА 6

# ЧИСЛОВЕ ІНТЕГРУВАННЯ: БАЗОВІ ФОРМУЛИ (ПРЯМОКУТНИКІВ, СЕРЕДНІХ ПРЯМОКУТНИКІВ, ТРАПЕЦІЙ, СІМПСОНА), ОЦІНКА ПОХИБКИ

## Числове інтегрування

*Постановка задачі.* Необхідно наближено обчислити визначений інтеграл

$$I=\int_a^b f(x)dx \quad \text{або} \quad I=\int_a^b \rho(x)u(x)dx, \tag{6.1}$$

де $\rho(x)>0$ – вагова функція, неперервна на $(a,b)$, $f(x)|u(x)$ – підінтегральні функції достатньої гладкості на $[a,b]$.

Більшість методів числового інтегрування (ЧІ) ґрунтуються на заміні (6.1) скінченною сумою – *квадратурними формулами* (КФ) [8, с. 85-89]

$$I_n=\sum_{k=1}^{n} A_k f_k \quad \text{або} \quad I_n=\sum_{k=1}^{n} A_k u_k, \tag{6.2}$$

де $A_k$ – *квадратурні коефіцієнти*, $x_k$ – *вузли* КФ.

***Означення 6.1.*** Якщо граничні точки $x=a$, $x=b$ входять до системи вузлів КФ, то формули (6.2) називаються формулами *закритого* типу, інакше – *відкритого* типу.

***Означення 6.2.*** КФ називається *чисельно стійкою*, якщо всі $A_k$ невід'ємні.

***Означення 6.3.*** *Похибкою* КФ називається величина $|I-I_n|=R_n$. Вона залежить від квадратурних коефіцієнтів і вузлів, а також гладкості функції $f|u$.

Формули (6.2) в цілому містять $2n$ коефіцієнтів, тому є можливість будувати різні КФ, що ґрунтуються на певній стратегії.

*Найпростіші формули* ЧІ отримують, вибираючи на $[a,b]$ фіксовану рівномірну сітку $\omega_h=\{x_k=a+(k-1)h, k=\overline{1,n}, h=(b-a)/(n-1)\}$ і користуючись адитивністю (6.1)

$$I=\int_a^b f(x)dx \Rightarrow I_n=\sum_{k=2}^{n} I_n(k), \quad I_n(k)=\int_{x_{k-1}}^{x_k} f(x)dx. \tag{6.3}$$

| Назва | Обчислення | Застосування складеної КФ | Точність |
|---|---|---|---|
| **1.** *КФ прямокутників* *а) лівих* | $hf_{k-1}$ | $h\sum_{k=1}^{n-1} f_k$ | $O(h^1)$ |
| *б) правих* | $hf_k$ | $h\sum_{k=2}^{n} f_k$ | $O(h^1)$ |
| *в) центральних (КФ середніх)* | $hf_{k-0.5}$, $x_{k-0.5}=0.5(x_{k-1}+x_k)$ | $h\sum_{k=2}^{n} f_{k-0.5}$ | $O(h^2)$ |
| **2.** *КФ трапецій* | $0.5h(f_{k-1}+f_k)$ | $h(0.5(f_1+f_n)+\sum_{k=2}^{n-1} f_k)$ | $O(h^2)$ |
| **3.** *КФ Сімпсона* | $\frac{0.5h}{3}(f_{k-1}+4f_{k-0.5}+f_k)$ | $\frac{h}{3}(f_1+f_n+4\sum_{k=2,4,...}^{n-1} f_k+2\sum_{k=3,5,...}^{n-2} f_k)$ | $O(h^4)$ |

Для отримання значення (6.1) із заданою точністю $0<\varepsilon<1$ використовують адаптивні алгоритми (як рекурентні так і рекурсивні), які будуються на апостеріорній оцінці похибки $R_n$ за допомогою першої формули Рунге-Ромберга [7, с. 92-94].

Якщо КФ будуються на стратегії забезпечення точного ($R_n=0$) обчислення (6.1) формулою (6.2) для всіх функції $u(x)$ з деякого класу U – наприклад, поліноми, то про такі КФ говорять, що вони є *КФ найвищого алгебраїчного степеня точності*. Саме такими є група формул *Гаусса-Крістоффеля*, де конкретна формула залежать від діапазону $[a,b]$ і $\rho(x)$ [8, с. 89-99] :

| $\rho(x)$ | $[a,b]$ | **Ортогональний многочлен** | **Назва КФ** |
|---|---|---|---|
| 1 | $[-1,1]$ | Лежандра | *Гаусса* |
| $1/\sqrt{1-x^2}$ | $[-1,1]$ | Чебишова 1-го роду | *Гаусса-Чебишова* |
| $(1-x)^{\alpha}(1+x)^{\beta},\ \alpha,\beta>-1$ | $[-1,1]$ | Якобі | *Якобі* |
| $x^{\alpha}\exp(-x),\ \alpha>-1$ | $[0,\infty)$ | Чебишова-Лагерра | *Лагерра* |
| $\exp(-x^2)$ | $(-\infty,\infty)$ | Чебишова-Ерміта | *Ерміта* |

Для найчастіше уживаних вагових функцій $\xi_i$ – вузли (які є коренями відповідних ортогональних поліномів) і $\gamma_i$ – ваги формул Гаусса-Крістоффеля уже обчислені і можна скористатись відповідними довідниками.

Для застосування КФ Гаусса при обчисленні інтеграла (6.1) на $[a,b]\neq[-1,1]$ необхідно спочатку виконати лінійне перетворення аргумента до значення $[-1,1]$ :

$$x_k=\tfrac{1}{2}(a+b)+\tfrac{1}{2}(b-a)\xi_k,\quad A_k=\tfrac{1}{2}(b-a)\gamma_i,\ k=\overline{1,n}.$$

Після цього можна використати КФ (6.2), де значення $\{\xi_i,\gamma_i\}$ при $n=1,2,3$ наступні :

$$n=1:\quad \xi_1=0;\quad \gamma_1=2.$$
$$n=2:\quad -\xi_1=\xi_2=\sqrt{1/3};\quad \gamma_1=\gamma_2=1.$$
$$n=3:\quad -\xi_1=\xi_3=\sqrt{3/5},\xi_2=0;\quad \gamma_1=\gamma_3=5/9,\gamma_2=8/9.$$

Формула Гаусса розрахована на функції $u(x)$, які мають достатньо високі похідні, причому не дуже великі за абсолютною величиною.

**Збіжність КФ**. Узагальнені формули, наведені в таблиці для формули (6.3) і формули Гаусса-Крістоффеля при $n\to\infty$ для довільної неперервної функції є *збіжними*. Якщо $R_n=O(h^p)$, то маємо формулу збіжності з *p-порядком точності*.

**Короткий огляд складніших ситуацій ЧІ** [7, с. 100-124]**.**

*Наявність розривів у* $f(x)$**:** розбиваємо $[a,b]$ на відрізки $\Delta_C$, де $f(x), f^{(m)}(x)$ неперервні, використовуємо адитивність інтеграла, застосовуємо на кожному відрізку $\Delta_C$ КФ, наприклад, середніх (при невеликих *m*) або Сімпсона чи Гаусса (*m*≥*4*).

*Швидко осцилюючі* $f(x)$**:** представляємо $f(x)=u(x)\exp(i\omega\cdot x)$, де частота ω відома, а амплітуда $u(x)$ мало змінюється за період основного коливання, далі використовуємо КФ *Філона* або КФ на базі *сплайнів.*

*Змінна верхня границя* $F(x)=\int_a^x f(x)dx$**:** для поодиноких обчислень можна використати КФ середніх, Сімпсона, а для значень $x\in\vec{\mathbf{X}}$, де $\vec{\mathbf{X}}$ – впорядкований за зростанням масив, використовуємо адитивність $F(x)=F(X_N)+\int_{X_N}^{x} f(x)dx$, $X_N\le x\le X_{N+1}$, де для обчислення інтегралу використовуємо звичайні КФ.

*Невласні інтеграли:* є декілька варіантів, зокрема використання КФ Лагерра і Ерміта для обчислення інтегралів на півосі $[0,\infty)$ і осі $(-\infty,\infty)$ відповідно.

*m-кратні інтеграли:* для кратності *m* = {2, 3} використовують *послідовне інтегрування*, метод *комірок*, а для великих значень кратності (*m*>4) метод *Монте-Карло*.

# ТЕМА 7

# ЧИСЛОВІ МЕТОДИ РОЗВ'ЯЗУВАННЯ ЗАДАЧІ КОШІ: РІЗНИЦЕВІ, ТИПУ РУНГЕ-КУТТА

Розглянемо задачу Коші для системи ЗДР

$$\begin{cases} \vec{\mathbf{u}}'(x) = \vec{\mathbf{f}}(x, \vec{\mathbf{u}}(x)), \quad x \in (x_0, x_{end}], \\ \vec{\mathbf{u}}(x_0) = \vec{\mathbf{u}}_\mathbf{0} \end{cases} \tag{7.1}$$

і для її наближеного розв'язку будемо використовувати *чисельні* методи:

a) *різницеві* (*однокрокові* і *багатокрокові*);

b) *типу Рунге-Кутта*.

Як перші, так і другі методи бувають *явними* і *неявними*.

## Різницеві однокрокові методи

Розглянемо спочатку скалярний варіант (7.1). Вводимо за змінною $x$ сітку з певним кроком (рівномірним або нерівномірним), виконуємо апроксимацію лівої і правої частини рівняння (7.1) на шаблоні $S[x_n, x_{n+1}]$.

**Явний метод Ейлера**. Схема методу

$$y_{n+1} = y_n + h_n \cdot f(x_n, y_n), \quad y_0 = u_0, \quad n = 0,1,\ldots. \tag{7.2}$$

**Неявний метод Ейлера.** Схема методу

$$y_{n+1} = y_n + h_n \cdot f(x_{n+1}, y_{n+1}), \quad y_0 = u_0, \quad n = 0,1,\ldots. \tag{7.3}$$

**Симетрична схема.** Схема методу

$$y_{n+1} = y_n + 0.5h_n \cdot (f(x_n, y_n) + f(x_{n+1}, y_{n+1})), \quad y_0 = u_0, \quad n = 0,1,\ldots. \tag{7.4}$$

Для автоматичного вибору кроку використовують метод *Рунге* (враховуючи, що методи (7.2)–(7.3) мають точність $O(h)$, а метод (7.4) – точність $O(h^2)$).

## Методи типу Рунге-Кутта

Характерною особливістю цих методів є те, що обчислення проводяться не тільки в точках сітки, але і в деяких проміжних точках (використовуючи спеціальні квадратурні формули):

$$y_{n+1} - y_n = \int_{x_n}^{x_{n+1}} f(\xi, u(\xi))d\xi = h_n \sum_{i=1}^{m} A_i \cdot K_i, \quad y_0 = u_0, \quad n = 0,1,\ldots.$$

Сім'я методів Рунге-Кутта **2-го порядку точності** ($m$=2 ). Схема методу

$$\begin{aligned} &A_1 = 1-\gamma, \quad A_2 = \gamma, \quad \theta = 0.5h_n/\gamma, \\ &K_1 = f(x_n, y_n), \quad K_2 = f(x_n + \theta, y_n + \theta K_1), \quad \gamma \in (0,1]. \end{aligned} \tag{7.5}$$

Найчастіше використовуються такі значення: $\gamma = 0.5, \quad \gamma = 1$.

Методи Рунге-Кутта **4-го порядку точності** ($m$=4). Схема найпопулярнішого методу з цього сімейства має вигляд

$$\begin{aligned} &A_1 = \tfrac{1}{6}, \quad A_2 = \tfrac{1}{3}, \quad A_3 = \tfrac{1}{3}, \quad A_4 = \tfrac{1}{6}, \\ &K_1 = f(x_n, y_n), \quad K_2 = f(x_n + 0.5h_n, y_n + 0.5h_n K_1), \\ &K_3 = f(x_n + 0.5h_n, y_n + 0.5h_n K_2), \quad K_4 = f(x_{n+1}, y_n + h_n K_3). \end{aligned} \tag{7.6}$$

Методи Рунге-Кутта використовуються в стандартних солверах для розв'язання (7.1), а також у методі *прямих*, для розв'язання прикладних задач математичної фізики, зокрема нелінійних [4, с. 49-62], [5, с. 67-70], [9, с. 83-103].

## Поняття жорстких систем диференціальних рівнянь

**Стійкість умовна і абсолютна.** Якщо розглядати модельну задачу Коші для (7.1) з $f(x,u(x))=\lambda\cdot u(x),\ x>0,\ \lambda<0$, то стійкість означає виконання нерівності $|u(x+h)|\le|u(x)|,\ h>0$ [18, с. 247-258].

При використанні різницевого методу, потрібно вимагати, щоб розв'язок різницевої задачі задовольняв нерівність, аналогічну нерівності для розв'язку відповідної диференціальної задачі.

***Означення 7.1.*** Різницевий метод є *абсолютно стійким* при $\forall\ h>0$, і *умовно стійким*, якщо він стійкий при певних обмеженнях на крок *h*.

Для явного методу Ейлера: $|y_{n+1}|\le|1+\lambda h||y_n|$ і маємо обмеження

$$0<h\le 2/|\lambda|. \tag{7.7}$$

Аналогічне обмеження на крок можна отримати для методу Рунге-Кутта 2-го порядку точності ($\gamma=1$), а для 4-го – $|\lambda|\cdot h\le 2.78$.

Для неявного методу Ейлера маємо $|y_{n+1}|\le\left|\dfrac{1}{1-\lambda h}\right||y_n|\Rightarrow 1/(1-\lambda h)<1$ при $0<h\ \&\ \lambda<0$.

Із наведених прикладів бачимо, що явні методи умовно стійкі, а неявні – безумовно стійкі, але вимагають розв'язання нелінійних рівнянь.

Розглянуті методи без зусиль переносяться на системи ЗДР, але виникає проблема з їх стійкістю (обмеження типу (7.7)), пов'язана з різномасштабністю процесів, які описуються системою ЗДР. Проілюструємо це на простому прикладі системи двох рівнянь

$$\begin{cases} u_1'(x)+a_1u_1(x)=0,\\ u_2'(x)+a_2u_2(x)=0, \end{cases}$$

де $x>0;\ a_{1,2}>0,\ a_2>>a_1$ або представимо її у векторній формі

$$\frac{d\vec{\mathbf{u}}(x)}{dx}=\mathbf{A}\cdot\vec{\mathbf{u}}(x),\quad \mathbf{A}=\begin{pmatrix}-a_1 & 0\\ 0 & -a_2\end{pmatrix},\quad \vec{\mathbf{u}}(x)=\begin{pmatrix}u_1(x)\\ u_2(x)\end{pmatrix}. \tag{7.8}$$

Ураховуючи, що $a_{1,2}>0,\ a_2>>a_1$, з деякого *x* ми повинні мати (аналітично) $\vec{\mathbf{u}}(x)=(u_1(x),u_2(x))^T\approx(u_1(x),0)^T$.

Проте чисельно, якщо використовувати явний метод Ейлера, ми маємо обмеження на крок у вигляді нерівності (7.7), що приводить до $h\cdot a_2\le 2$.

***Означення 7.2.*** Нехай матриця **A** – стала, квадратна, розмірності *N*. Система (7.8) називається *жорсткою*, якщо:

**1)** $\mathrm{Re}(\lambda_k)<0, k=1,2,...,N$ – система асимптотично стійка за Ляпуновим;

**2)** $S=\dfrac{\max\limits_k|\mathrm{Re}(\lambda_k)|}{\min\limits_k|\mathrm{Re}(\lambda_k)|}$ – *число жорсткості* системи велике.

де $\lambda_k$ – власні числа матриці **A**.

Для розв'язування жорстких систем на практиці (є готові солвери в пакетах комп'ютерної математики) використовують :

♦ *методи Гіра* – чисто неявний *m*-кроковий різницевий метод (до 10 порядку);

♦ методи *Розенброка*;

♦ *діагонально неявні методи Рунге-Кутта*.

# ТЕМА 8

# РІЗНИЦЕВІ МЕТОДИ РОЗВ'ЯЗАННЯ КРАЙОВИХ ЗАДАЧ ДЛЯ ЗВИЧАЙНИХ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

Для чисельного розв'язання диференціальних рівнянь звичайних і в частинних похідних, як лінійних так і нелінійних, застосовується потужний метод *сіток* або *різницевий метод* [7, 8, 14, 17, 18]. Розглянемо основні ідеї методу різницевих схем (МРС) на прикладі лінійної крайової задачі, записаної в операторному вигляді:

$$Lu(x) = f(x), \quad x \in D; \tag{8.1}$$

$$lu(x) = g(x), \quad x \in \partial D. \tag{8.2}$$

Тут $Lu(x)$, $lu(x)$ – деякі лінійні диференціальні оператори; $f(x)$, $g(x)$ – функції відповідної гладкості, які описують праву частину ЗДР і крайові умови; $\overline{D} = D \cup \partial D \equiv [a,b]$ – область (відрізок), де шукається розв'язок. Для (8.1)–(8.2) виконуються відповідні умови існування та єдиності (задача поставлена коректно).

**Основні поняття методу скінченних різниць** Основні принципи МСР полягають у систематичній дискретизації всього, що фігурує в постановці задачі (8.1)–(8.2).

**1.** Область $\overline{D}$ заміняємо сітковою областю $D_h \cup \partial D_h$. Вводиться *сітка* $\overline{D}_h$ – довільна скінченна множина (неспівпадаючих) точок (*вузлів* сітки):

$$\textit{нерівномірна} \quad \overline{D}_h = \{a = x_1 < x_2 < \ldots < x_N = b, \ \ h_k = x_{k+1} - x_k\}; \tag{8.3}$$

або

$$\textit{рівномірна} \quad \overline{D}_h = \{x_k = a + (k-1)h, \ \ k = 1,2,\ldots, N, \ \ b - a = (N-1)h\}. \tag{8.4}$$

**2.** Функцій, як відомі, так і невідомі, заміняємо сітковими функціями, при цьому будемо позначати $f(x_k) = f_k$, $g(x_k) = g_k$; $u(x_k) = u_k \equiv y_k$, $x_k \in \overline{D}_h$.

**3.** Диференціальні оператори заміняємо на різницеві $Lu(x) \to L_h y$, $lu(x) \to l_h y$ – зважені суми $\sum_{x_j \in S\{x_k\}} c_j y_j$ шуканої сіткової функції на певній сукупності вузлів (*шаблоні*) $S\{x_k\} \in \overline{D}_h$. Найчастіше для цього використовують:

- ✓ *метод безпосередньої заміни диференціальних операторів на формули числового диференціювання (метод різницевої апроксимації);*
- ✓ *інтегро-інтерполяційний метод (метод балансу);*
- ✓ *варіаційно-різницевий метод (метод скінченних елементів).*

Можуть використовуватись й інші методи: *невизначених коефіцієнтів, суматорних тотожностей, апроксимації квадратичного функціонала*.

**4.** Для отримання певних теоретичних оцінок про побудовану *різницеву схему*:

$$L_h y = \varphi(x_k), \quad x_k \in D_h; \tag{8.5}$$

$$l_h y = \mu(x_k), \quad x_k \in \partial D_h. \tag{8.6}$$

– сім'ю систем лінійних алгебраїчних рівнянь (СЛАР) з параметром $h$, вводимо сіткові норми $\|\circ\|_h$, відповідний аналог неперервної норми $\|\circ\|$ такий, щоб при $h \to 0 \Rightarrow \|\circ\|_h \to \|\circ\|$.

Для отримання наближеного дискретного розв'язку задачі (8.1)–(8.2) необхідно розв'язати СЛАР (8.5)–(8.6), а при цьому виникають наступні питання:

1) чи існує її розв'язок?
2) як фактично знаходити розв'язок: який алгоритм реалізувати, чи буде він ефективним при великих значеннях $N$ ?

І звичайно для РС суттєвим є ще одне питання: чи буде сіткова функція збігатися до точного розв'язку при $h \to 0$. Все це вимагає певних теоретичних досліджень РС.

Побудова і аналіз РС у методі МСР базується на таких поняттях, як *апроксимація*, *стійкість* та *збіжність* [14, с. 194-205].

***Означення 8.1.*** *Похибкою апроксимації оператора* $L$ *оператором* $L_h$ називають різницю

$$\psi(x_k) = L_h y - (Lu)_h, \tag{8.7}$$

де $(Lu)_h$ – проектор неперервного оператора на сітку.

***Означення 8.2.*** Оператор $L_h$ має $m$*-й порядок апроксимації в точці* $x_k \in D_h$ (*локальний*), якщо

$$\psi(x_k) = O(h^m) \text{ або } |\psi(x_k)| = Mh^m, \quad m > 0, \tag{8.8}$$

де $M, m \in Const$ і не залежать від кроку сітки.

***Означення 8.3.*** Оператор $L_h$ має $m$*-й порядок апроксимації на сітці*, тобто в деякій сітковій нормі, якщо

$$\|\psi(x_k)\|_h = O(h^m), \quad m > 0. \tag{8.9}$$

***Означення 8.4.*** Сіткова функція $z(x_k) \equiv z_k = y_k - (u)_h$, називається *похибкою різницевої схеми в точці* $x_k \in D_h$.

Якщо підставимо $y_k = z_k + (u)_h$, в рівняння (8.5) $L_h y \equiv L_h z_k + L_h (u)_h = \varphi(x_k)$, то отримаємо, що похибка задовольняє рівняння

$$L_h z_k = \psi_h(x_k). \tag{8.10}$$

***Означення 8.5.*** Сіткова функція $\psi_h(x_k) = \varphi(x_k) - L_h(u)_h$ називається *похибкою апрокси - мації різницевого рівняння* (8.5) *на розв'язку диференціального рівняння* (8.1) *в точці* $x_k \in D_h$.

Виконаємо над функцією $\psi_h(x_k)$ ланцюжок очевидних перетворень

$$\psi_h(x_k) = 0 + \varphi(x_k) - L_h(u)_h = (Lu - f)_h + \varphi(x_k) - L_h(u)_h = [(Lu)_h - L_h(u)_h] + [\varphi(x_k) - (f)_h]$$

і в результаті представимо її у наступному вигляді:

$$\psi_h(x_k) = \psi_{h,1}(x_k) + \psi_{h,2}(x_k),$$

де

$$\psi_{h,1}(x_k) = (Lu)_h - L_h(u)_h, \quad \psi_{h,2}(x_k) = \varphi(x_k) - (f)_h.$$

***Означення 8.6.*** Сіткові функції $\psi_{h,1}(x_k)$ і $\psi_{h,2}(x_k)$ називаються відповідно *похибкою апроксимації диференціального оператора* (див. (8.7)) і *похибкою апроксимації правої частини.*

***Означення 8.7.*** Різницеве рівняння (8.5) *апроксимує диференціальне рівняння* (8.1), якщо

$$\|\psi_h(x_k)\|_h \to 0 \text{ при } |h| \to 0 \tag{8.11}$$

і має $p$ *– й порядок апроксимації*, якщо

$$\|\psi_h(x_k)\|_h \le M_1 |h^p|, \quad p > 0, \tag{8.12}$$

де $M_1, p \in Const$ і не залежать від кроку сітки.

Аналогічно визначаються похибки апроксимації для крайових умов

$$\nu_h(x_k) = [(lu)_h - l_h(u)_h] + [\mu(x_k) - (g)_h]$$

і порядок її апроксимації

$$\|\nu_h(x_k)\|_h \le M_2 |h^r|, \ \ r > 0, \ \ M_2, r \in Const. \tag{8.13}$$

Для дослідження сіткових функцій $\psi_h(x_k), \nu_h(x_k)$ використовують розвинення по $h$ в ряд Тейлора в околі деякого сіткового вузла.

***Означенн 8.8.*** Різницева схема (8.5)–(8.6) називається *коректною*, якщо

1) її розв'язок існує і єдиний при довільних даних $\varphi(x_k), \mu(x_k)$;
2) є *стійкість*, тобто, існують $M_3, M_4 \in Const$, не залежні від кроку сітки такі, що виконується оцінка (стійкість за початковими даними і за правою частиною)

$$\|y_k\|_h \le M_3 \|\varphi(x_k)\|_h + M_4 \|\mu(x_k)\|_h. \tag{8.14}$$

***Означення 8.9.*** Розв'язок різницевої задачі (3.5)–(3.6) *збігається* до розв'язку диференціальної задачі (8.1)–(8.2), якщо при $|h| \to 0$

$$\|z(x_n)\|_h \to 0 \tag{8.15}$$

і має $q$ – й *порядок точності* $O(|h|^q)$, якщо

$$\|z(x_k)\|_h \le \tilde{M} |h|^q, \ \ q > 0, \tag{8.16}$$

де $\tilde{M}, q \in Const$ і не залежні від кроку сітки.

**Теорема 8.1** (*Лакса*) Нехай лінійна диференціальна задача (8.1)–(8.2) поставлена коректно. Різницева задача (8.5)–(8.6) є коректною і апроксимує вихідну задачу на розв'язку $u(x)$. Тоді розв'язок різницевої задачі $y_k$ збігається до розв'язку диференціальної задачі $u(x)$, причому порядок точності співпадає з порядком апроксимації.

Доведення. Враховуючи, що задача для похибки $z_k$ записується у вигляді

$$L_h z_k = \psi_h(x_k);$$
$$l_h z_k = \nu_h(x_k),$$

і має ту саму структуру, що і різницева задача (8.5)–(8.6), на основі її коректності маємо оцінку

$$\|z_k\|_h \le M_3 \|\psi_h(x_k)\|_h + M_4 \|\nu_h(x_k)\|_h.$$

Підставимо сюди відповідні оцінки апроксимації (8.12), (8.13), отримаємо

$$\|z_k\|_h \le M_3 M_1 |h|^p + M_4 M_2 |h|^r \le \tilde{M} |h|^q, \quad q = \min(p, r).$$

Значення теореми Лакса: дослідження збіжності РС розпадається на два окремих етапи дослідження:

a) доведення апроксимації РС на розв'язку вихідної задачі;
b) отримання оцінок типу (8.14) – *апріорних оцінок* для доведення стійкості.

**Різницеві схеми для лінійних задач.** Розглянемо застосування МРС на прикладі лінійної крайової задачі [7, с. 268-271]:

$$Lu(x) \equiv u''(x) - p(x)u(x) = -f(x), \ \ x \in D \equiv (a, b);$$
$$\overline{D} = D \cup \partial D, \quad \partial D = a \cup b; \tag{8.17}$$

$$u(a) = d_1, \ u(b) = d_2. \tag{8.18}$$

Вважатимемо, що задані функції $p(x) \ge 0$, $f(x)$ двічі неперервно диференційовані, а отже $u(x)$ має четверту неперервну похідну.

Введемо рівномірну сітку (8.4) і використаємо для побудови різницевої задачі (РЗ) метод безпосередньої заміни диференціальних операторів на формули числового

диференціювання. Якщо на триточковому шаблоні $S\{x_{k-1}, x_k, x_{k+1}\}$ замінити $u''(x_k)$ на $y_{\bar{x}x,k}$ – *другу різницеву похідну*, то для (8.17) отримаємо РЗ

$$\frac{y_{k-1} - 2y_k + y_{k+1}}{h^2} - p_k y_k = -f_k, \quad k = 2,3,..., N-1; \tag{8.19}$$

$$y_1 = d_1, \quad y_N = d_2. \tag{8.20}$$

Отриману РЗ представимо у вигляді $\mathbf{A}\vec{\mathbf{y}} = -\vec{\mathbf{f}}$ – СЛАР для сіткової функції $\vec{\mathbf{y}} = \{y_1,...,y_N\}^{\mathrm{T}}$:

$$\mathbf{A} = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -(2+h^2 p_2) & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -(2+h^2 p_3) & 1 & 0 & \cdots & 0 \\ 0 & 0 & \dots & \ddots & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 & -(2+h^2 p_{N-2}) & 1 & 0 \\ 0 & \dots & 0 & 0 & 1 & -(2+h^2 p_{N-1}) & 1 \\ 0 & \dots & 0 & 0 & 0 & 0 & -1 \end{pmatrix}, \quad \vec{\mathbf{f}} = \begin{pmatrix} d_1 \\ h^2 f_2 \\ h^2 f_3 \\ \vdots \\ h^2 f_{N-2} \\ h^2 f_{n-1} \\ d_2 \end{pmatrix}.$$

Матриця **A** цієї СЛАР – тридіагональна і враховуючи $p(x) \geq 0$, маємо діагональну перевагу її елементів, а отже розв'язок (8.19)–(8.20) існує і єдиний. Для його знаходження використовують стійкий *метод монотонної лівої/правої прогонки*.

Дослідимо апроксимацію РЗ. Використовуючи розвинення сіткових функцій $\psi(x_k)$ (8.7) і $\psi_h(x_k)$ (8.10) в ряд Тейлора

$$\psi(x_k) = L_h y - (Lu)_h = \frac{y_{k-1} - 2y_k + y_{k+1}}{h^2} - u''(x_k) = \frac{h^2}{12} u^{(4)}(x_k) + O(h^4) = O(h^2),$$

$$\psi_h(x_k) = \left[(u''(x_k) - p(x_k)u(x_k)) - (y_{\bar{x}x,k} - p_k y_k)\right] + \left[-f_k + f(x_k)\right] =$$

$$= \psi_{h,1}(x_k) + \psi_{h,2}(x_k) = -\frac{h^2}{12} u^{(4)}(x_k) = O(h^2)$$

і точну апроксимацію крайових умов $\nu_h(x_k) = 0,\ x_k = a, b$, отримаємо другий порядок апроксимації.

Для отримання *апріорних оцінок* (8.14) скористаємся *принципом максимуму*. З урахуванням того, що маємо наступні співвідношення

$$\|y_k\|_c = \max_{1 \leq k \leq N} |y_k| = \max\left\{|d_1|, |d_2|, \max_{1<k<N} |y_k|\right\} = \max\left\{|d_1|, |d_2|, |y_i|\right\}, \quad (1 < i < N);$$

виконаємо перетворення в рівнянні (8.19), записаному у вузлі з індексом $i$

$$(2 + h^2 p_i) y_i = y_{i-1} + y_{i+1} + h^2 f_i;$$

Проведемо оцінки

$$(2 + h^2 p_i)|y_i| \leq |y_{i-1}| + |y_{i+1}| + h^2 |f_i| \leq 2|y_i| + h^2 |f_i|; \quad |y_i| \leq \left|\frac{f_i}{p_i}\right|,$$

а отже маємо стійкість РС у вигляді $\|y_k\|_c \leq \left\|\frac{f_k}{p_k}\right\|_c + \max(|d_1|, |d_2|)$. З отриманої оцінки і враховуючи вигляд $\psi_h(x_k)$, для похибки РС $z(x_k), x_k \in \overline{D}_h$ виконується нерівність $\|z_k\|_c \leq \frac{h^2}{12} \left\|\frac{u^{(4)}(x_k)}{p_k}\right\|_c$. Таким чином, маємо РС (8.19)–(8.20) другого порядку точності.

Поставимо питання: чи можна на розглянутому триточковому шаблоні $S\{x_{k-1}, x_k, x_{k+1}\}$ отримати для задачі (8.17), (8.18) різницеву схему, точність якої більш високого порядку, ніж другий? Розглянемо випадок $p(x) \equiv p \in Const$. Враховуючи, що крайові умови (8.18) не впливають на порядок РС, апроксимацію (8.18) запишемо наступним чином

$$y_{\bar{x}x,k} - \xi_k y_k = -\varphi_k,$$

де сіткові функції $\xi_k$, $\varphi_k$ необхідно вибрати так, щоб у (8.12) $p > 2$. Запишемо (8.12) і виконаємо ланцюжок нескладних перетворень з урахуванням розв'язку диференціальної задачі

$$\begin{aligned}
\psi_h(x_k) &= \left[(u''(x_k) - pu(x_k)) - (y_{\bar{x}x,k} - \xi_k y_k)\right] + \left[-\varphi_k + f(x_k)\right] = \\
&= (-\varphi_k + f_k) - (p - \xi_k)u(x_k) - \frac{h^2}{12}u^{(4)}(x) + O(h^4) = \\
&= \left| \begin{array}{l} u'' = pu - f, \quad u''' = pu' - f', \\ u^{(4)} = pu'' - f'' = p(pu - f) - f'' = p^2 u - pf - f'' \end{array} \right| = \\
&= \left[-\varphi_k + f_k + \frac{h^2}{12}(pf_k + f_k'')\right] + \left[\xi_k - p\left(1 + p\frac{h^2}{12}\right)\right]u(x_k) + O(h^4).
\end{aligned}$$

З останнього співвідношення, при

$$\tilde{p} = 1 + h^2 p/12, \quad \xi_k = p \cdot \tilde{p}, \quad \varphi_k = \tilde{p} \cdot f_k + h^2 f_k''/12$$

маємо $\psi_h(x_k) = O(h^4)$ ! Такий прийом підвищення апроксимації на розв'язку диференціальної задачі можна застосовувати і для крайових умов II, III-го роду.

В розглянутій задачі (8.17), (8.18) ми мали крайові умови першого роду, тому вони апроксимувались точно. У випадку наявності $u'(x)$ у крайових умовах, використовують певну методику для отримання апроксимації того самого порядку, що і для диференціального рівняння, тобто в (8.13) має бути $r = p$. Розглянемо приклад крайової задачі для рівняння (8.17), де в лівій граничній точці задано умову третього роду

$$u'(a) + g_1 u(a) = -d_1, \tag{8.21}$$

а в правій граничній точці задано умову другого роду

$$u'(b) = d_2. \tag{8.22}$$

<u>*Метод підвищення порядку апроксимації на розв'язку*</u>. Застосуємо його до умови (8.21). Спочатку замінимо $u'(a) \approx y_{x,1}$ – правою різницевою похідною і отримаємо апроксимацію цієї умови з першим порядком

$$\frac{y_2 - y_1}{h} + g_1 y_1 = -d_1.$$

Для отримання порядку $O(h^2)$ виконаємо розклад $y_2$ в ряд Тейлора

$$\frac{\left(y_1 + hu'(a) + 0.5h^2 u''(a) + O(h^3)\right) - y_1}{h} + g_1 y_1 = -d_1,$$

а для обчислення $u''(a)$ скористаємось неперервним продовженням диференціального рівняння (8.17) на межу $x = a$. В результаті отримаємо

$$u'(a) + 0.5h\left[p(a)u(a) - f(a)\right] + O(h^2) + g_1 u(a) = -d_1.$$

Порівнюючи цей вираз із (8.21), бачимо, що для їх рівності з точністю до величин $O(h^2)$ нам необхідно "прибрати" вираз $0.5h[p(a)u(a) - f(a)]$ порядку $O(h)$. Отже, різницева умова, що апроксимує крайову умову (8.21) з другим порядком, буде наступною:

$$\frac{y_2 - y_1}{h} + g_1 y_1 = -d_1 + 0.5h(p_1 y_1 - f_1),$$

і може бути перетворенао до форми

$$-\eta_1 y_1 + y_2 = -\mu_1, \tag{8.23}$$

де $\eta_1 = 1 - hg_1 + 0.5h^2 p_1$, $\mu_1 = hd_1 + 0.5h^2 f_1$.

*Метод фіктивної (законтурної) точки*. Застосуємо його до умови (8.22). За допомогою центральної різниці в точці $b = x_N$ замінимо $u'(b) = \frac{y_{N+1} - y_{N-1}}{2h} + O(h^2)$, а для виключення $y_{N+1}$ скористаємось рівнянням $y_{\bar{x}x,N} - p_N y_N = -f_N$. Отже, різницева умова, яка апроксимує крайову умову (8.22) з другим порядком, буде наступною:

$$\frac{y_{N-1} - 2y_N + (y_{N-1} + 2hd_2)}{h^2} - p_N y_N = -f_N,$$

і може бути перетворена до форми

$$y_{N-1} - \eta_2 y_N = -\mu_2, \tag{8.24}$$

де $\eta_2 = 1 + 0.5h^2 p_N$, $\mu_2 = hd_2 + 0.5h^2 f_N$. Таким чином, розв'язок РС (8.19),(8.23),(8.24) з другим порядком точності збігається до розв'язку крайової задачі (8.17),(8.21),(8.22).

**Різницеві схеми для нелінійних задач.** Розглянемо застосування МРС до простої нелінійної крайової задачі [7, с. 271-275]:

$$\begin{cases} u''(x) = -f(x, u(x)), \quad x \in (a,b); \\ u(a) = d_1, \ u(b) = d_2. \end{cases} \tag{8.25}$$

Використовуючи рівномірну сітку (8.4), аналогічно МСР для лінійної задачі, отимаємо таку нелінійну різницеву задачу:

$$\begin{cases} y_{\bar{x}x,k} = -f(x_k, y_k), \quad k = \overline{2, N-1}; \\ y_1 = d_1, \quad y_N = d_2. \end{cases} \tag{8.26}$$

Для задачі (8.26) можна довести, що вона має порядок апроксимації $O(h^2)$, а також коректність і збіжність з другим порядком до розв'язку задачі (8.25) при умові, що існують і обмежені : $|u^{IV}(x)|$, $|f'_u(x)|$, $|f'_u(x)| \geq m_1 > 0$, $|u^{IV}(x)| \leq M_4$.

Як знайти розв'язок ? Найкраще використати метод Ньютона :

$$\begin{cases} y^{(s+1)}_{\bar{x}x,k} = -f(x_k, y^{(s+1)}_k), \quad k = \overline{2, N-1}; \\ y^{(s+1)}_1 = d_1, \quad y^{(s+1)}_N = d_2. \end{cases}$$

Представимо $y^{(s+1)}_k = y^{(s)}_k + \Delta^{(s)}_k$, де $y^{(s)}_k$ – відома величина, а $\Delta^{(s)}_k$ – деяка добавка. Виконаємо лінеаризацію, отимаємо для $\Delta^{(s)}_k$ на ітерації $s$ таку лінійну різницеву задачу:

$$\begin{cases} \Delta^{(s)}_{\bar{x}x,k} + f'_u(x_k, y^{(s)}_k)\Delta^{(s)}_k = -\left(f(x_k, y^{(s)}_k) + y^{(s)}_{\bar{x}x,k}\right), \quad k = \overline{2, N-1}; \\ \Delta^{(s)}_1 = 0, \quad \Delta^{(s)}_N = 0. \end{cases} \tag{8.27}$$

Збіжність ІП (8.27) буде за умови $\| 0.5 f''_{uu} / f'_u \|_C < 1$ і збіжність квадратична !

# ТЕМА 9

# РС ДЛЯ РОЗВ'ЯЗКУ КРАЙОВИХ ЗАДАЧ ДЛЯ РІВНЯНЬ ГІПЕРБОЛІЧНОГО ТИПУ ПЕРШОГО ПОРЯДКУ

Розглянемо застосування МРС до розв'язання нестаціонарних крайових задач математичної фізики (класифікація рівнянь і постановки основних задач, див., наприклад, в посібниках [1, 2]). Почнемо з одновимірних лінійних рівнянь гіперболіного типу – *рівнянь переносу* [7, с. 334-344].

Нехай в області $\overline{D} = D \cup \partial D_1 \cup \partial D_2 = [a,b] \times [0,T]$ маємо *мішану задачу Коші*:

$$\frac{\partial u(x,t)}{\partial t} + c\frac{\partial u(x,t)}{\partial x} = f(x,t), \quad c = const > 0, \ (x,t) \in D \equiv (a,b] \times (0,T], \tag{9.1}$$

з крайовою умовою

$$u(a,t) = \mu_1(t), \quad t \in \partial D_2 \equiv [0,T] \tag{9.2}$$

і початковою умовою

$$u(x,0) = u0(x), \quad x \in \partial D_1 \equiv [a,b]. \tag{9.3}$$

Вважатимемо, що:

- дані початкової і крайової умов мають достатню гладкість і для них виконуються необхідні *умови узгодження* в точці $(a;0)$;
- функція $f$ неперервно диференційована;
- шукана функція $u$ також достатньо гладка.

**Побудова сімейства різницевих схем.** Уведемо рівномірні сітки

$$\overline{\omega}_h = \left\{x_k = a + (k-1)h, \quad k = \overline{1,N}, \quad b - a = (N-1)h\right\};$$
$$\overline{\omega}_\tau = \left\{t_j = j\tau, \quad j = \overline{0,M}, \quad T = M\tau\right\},$$

і замінимо область $\overline{D}$ сітковою область $\overline{\omega}_{h\tau} = \overline{\omega}_h \times \overline{\omega}_\tau$, а також будемо використовувати наступні позначення для сіткових функцій

$$y_k^j \equiv y_k \equiv y = u(x_k, t_j), \quad y_k^{j+1} \equiv \hat{y}_k \equiv \hat{y} = u(x_k, t_{j+1}).$$

Виберемо чотири сіткові шаблони (рис. 9.1):

**a)** $S\{(x_{k-1},t_j),(x_k,t_j),(x_k,t_{j+1})\}$;

**b)** $S\{(x_k,t_j),(x_{k+1},t_j),(x_k,t_{j+1})\}$;

**c)** $S\{(x_k,t_j),(x_k,t_{j+1}),(x_{k-1},t_{j+1})\}$;

**d)** $S\{(x_{k-1},t_j),(x_k,t_j),(x_k,t_{j+1}),(x_{k-1},t_{j+1})\}$,

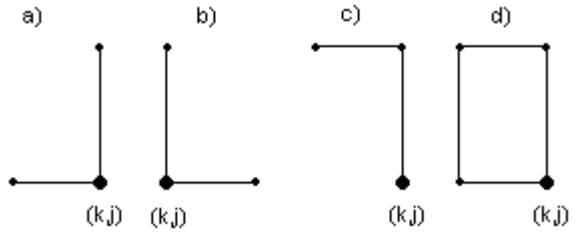



Рис. 9.1 Сіткові шаблони.

на яких, методом безпосередньої заміни диференціальних операторів на різницеві, запишимо для (9.1) наступні різницеві рівняння:

$$\frac{\hat{y}_k - y_k}{\tau} + c\frac{y_k - y_{k-1}}{h} = \overline{\varphi}_{k,j} \equiv f_{k-0.5,\,j-0.5}; \tag{9.4}$$

$$\frac{\hat{y}_k - y_k}{\tau} + c\frac{y_{k+1} - y_k}{h} = f_k^j; \tag{9.5}$$

$$\frac{\hat{y}_k - y_k}{\tau} + c\frac{\hat{y}_k - \hat{y}_{k-1}}{h} = \overline{\varphi}_{k,j}; \tag{9.6}$$

$$\frac{1}{2}\left(\frac{\hat{y}_{k-1} - y_{k-1}}{\tau} + \frac{\hat{y}_k - y_k}{\tau}\right) + \frac{c}{2}\left(\frac{y_k - y_{k-1}}{h} + \frac{\hat{y}_k - \hat{y}_{k-1}}{h}\right) = \overline{\varphi}_{k,j}. \tag{9.7}$$

для сіткових точок $(x_k, t_j)$ з $k = \overline{2,N}; \quad j = \overline{0,M-1}$.

Апроксимація крайової умови

$$y_1^j = \mu_1(t_j), \quad j = \overline{0,\mathrm{M}}, \tag{9.8}$$

і початкової умови

$$y_k^0 = u0(x_k), \quad k = \overline{1,\mathrm{N}}, \tag{9.9}$$

виконується точно.

***Означення 9.1.*** Сукупність точок сітки, які належать лінії (гіперплощині) $t = t_j$, називають *шаром*.

***Означення 9.2.*** Якщо для нестаціонарної задачі різницева схема записана на двох шарах сітки, то таку РС називають *двошаровою*, на трьох – *тришаровою*.

***Означення 9.3.*** Якщо в записі дво(три)шарової РС на верхньому шарі присутнє тільки одне невідоме значення сіткової функції, то таку схему називають *явною*, якщо більше одного – *неявною*.

Отже за цим означенням схеми (9.4), (9.5) явні, а (9.6), (9.7) – неявні.

**Похибка апроксимації.** Розглянемо схему (9.4). Для точки $(x_k, t_j)$ маємо розклади функцій в ряд Тейлора:

$$\hat{y}_k = y_k + \tau \cdot u_t'(x_k, t_j) + 0.5\tau^2 u_{tt}''(x_k, t_j) + O(\tau^3),$$

$$y_{k-1} = y_k - h \cdot u_x'(x_k, t_j) + 0.5h^2 u_{xx}''(x_k, t_j) + O(h^3)$$

$$\overline{\varphi}_{k,j} = f_{k,j} + 0.5\tau \cdot f_t'(x_k, t_j) - 0.5h \cdot f_x'(x_k, t_j) + O(h^2 + \tau^2),$$

які при підстановці у вираз для *похибки*

$$\psi_{k,j} = \left(y_{t,j} + cy_{\bar{x},k} - \overline{\varphi}_{k,j}\right) - \left(u_t'(x_k, t_j) + cu_x'(x_k, t_j) - f(x_k, t_j)\right) =$$

$$= 0.5\tau\left[u_{tt}''(x_k, t_j) - f_t'(x_k, t_j)\right] + 0.5h\left[f_x'(x_k, t_j) - cu_{xx}''(x_k, t_j)\right] + O(h^2 + \tau^2)$$

дають $O(h+\tau)$ при $f \in C^{(1,1)}(D), \; u \in C^{(2,2)}(D)$. Аналогічний порядок маємо в сітковій нормі $\|\circ\|_C$.

<u>*Завдання 1.*</u> Для схеми (9.5) в околі точки $(x_k, t_j)$ і для схеми (9.6) в околі точки $(x_k, t_{j+1})$ отримати похибку $O(h+\tau)$. Для схеми (9.7) в околі точки $(x_{k-0.5}, t_{j+0.5})$ отримати похибку $O(h^2 + \tau^2)$ і необхідні умови гладкості розв'язку $u(x,t)$.

***Означення 9.4.*** Двошарова РС записана у *формі з розділеним групуванням сіткової функції на шарах сітки*, якщо вона представлена у вигляді [6, с. 58]

$$\sum_k \alpha_k \hat{y}_{n+k} = \sum_r \beta_r y_{n+r} + \varphi_{n,j}, \tag{9.10}$$

де сумування на кожному шарі виконується за точками шаблону *S* біля *n*-го вузла, для якого в нумерації коефіцієнтів $\alpha_k$ виконується умова $|\alpha_0| = \max_k |\alpha_k|$.

***Означення 9.5.*** Двошарова РС записана в *канонічній формі*, якщо вона представлена у вигляді [18, с. 349]

$$B(t_j)\frac{\hat{y}_k - y_k}{\tau} + A(t_j)y_k = \varphi_{k,j}, \tag{9.11}$$

де $B(t_j)$, $A(t_j)$ – деякі сіткові оператори на точках шаблону *S*.

***Означення 9.6.*** Лінійна двошарова РС називається *рівномірно стійкою за початковими даними*, якщо

$$\left\| y^I(t) - y^{II}(t) \right\| \le K \left\| y^I(t_*) - y^{II}(t_*) \right\|, \quad t_0 \le t_* < t < T, \tag{9.12}$$

де $K \in const$ і не залежить від $t_*, h$, а $y^I, y^{II}$ – розв'язки РС $A_h y = \varphi$ з різними початковими даними і незмінними правими частинами (можна з $\varphi = 0$).

**Стійкість і збіжність різницевої схеми.** *Ознака рівномірної стійкості*. Якщо $A_h y^I = A_h y^{II}$, то для рівномірної стійкості за початковими даними достатньо, щоб при $\forall j$ виконувалась нерівність

$$\left\| y^I - y^{II} \right\| \le (1 + C\tau) \left\| y^I - y^{II} \right\|, \quad C \in const, \quad C \ge 0. \tag{9.13}$$

***Означення 9.7.*** Нехай лінійна двошарова РС рівномірно стійка за початковими даними і така, що на деякому сітковому шарі $y^I = y^{II}$, а на наступному шарі виконується нерівність

$$\left\| \hat{y}^I - \hat{y}^{II} \right\| \le \vartheta\tau \left\| \varphi^I - \varphi^{II} \right\|, \tag{9.14}$$

де $\vartheta \in const, \ \vartheta > 0$ і не залежить від $\tau, h$, тоді РС *стійка за правою частиною*.

<u>*Дослідження стійкості*</u>. В залежності від вибраної норми, найчастіше використовують наступні способи дослідження стійкості:

***A)*** *принцип максимуму* (у нормі $\|\circ\|_C$);

***B)*** *метод відокремлення змінних (метод Фур'є)* (у нормі $\|\circ\|_{L_2}$);

***C)*** *метод енергетичних нерівностей* (у нормі $\|\circ\|_A, \|\circ\|_{A^{-1}}$);

***D)*** *метод операторних нерівностей* (у нормі $\|\circ\|_A$).

Розглянемо застосування перших двох методів до наших РС.

*Принцип максимуму* для двошарової РС (9.10) формулюється так [7, с. 316-318]:

1) РС рівномірно стійка за початковими даними за умови

$$(1 + C\tau)|\alpha_0| \ge \sum_{k \ne 0} |\alpha_k| + \sum_r |\beta_r|, \quad C \in const, \tag{9.15}$$

2) РС стійка за правою частиною, якщо виконується (9.15) і умова

$$|\alpha_0| - \sum_{k \ne 0} |\alpha_k| \ge \frac{\theta}{\tau}, \quad \theta \in const, \tag{9.16}$$

Найчастіше в оцінках (9.15), (9.16) обирають $C = 0$ і $\theta = 1$.

Застосуємо цей спосіб для однорідної схеми (9.4), яку попередньо представимо у формі (9.10)

$$\alpha_0 \cdot \hat{y}_k = \beta_{-1} \cdot y_{k-1} + \beta_0 \cdot y_k,$$

а отже $\alpha_0 = \frac{1}{\tau}, \ \beta_{-1} = \frac{c}{h}, \ \beta_0 = \frac{1}{\tau} - \frac{c}{h}$. Підставимо ці значення в (9.15) при $C = 0$, отримаємо нерівність $\frac{1}{\tau} - \frac{c}{h} \ge \left| \frac{1}{\tau} - \frac{c}{h} \right|$, розв'язок якої дає *умову Куранта*

$$c\tau \le h. \tag{9.17}$$

Таким чином, за виконання умови Куранта, РС (9.4) є *умовно-стійкою* за початковими даними в нормі $\|\circ\|_C$. При виконанні (9.17), виконується і умова (9.16), а отже маємо умовну стійкість схеми (9.4) за правою частиною.

<u>*Завдання 2.*</u> Для однорідної схеми (9.6) виконати аналогічні дослідження її стійкості у нормі $\|\circ\|_C$.

*Метод відокремлення змінних*. Якщо двошарову однорідну РС у формі (9.11) перепишемо у формі (9.10) $B(t_j)\hat{y}_k = (B(t_j) - \tau \cdot A(t_j))y_k$, то легко отримаємо рівняння для похибки $B(t_j)\cdot \delta\, \hat{y}_k = (B(t_j) - \tau \cdot A(t_j))\cdot \delta\, y_k$. Шукатимемо частинний розв'язок цього рівняння методом відокремлення змінних [7, с. 318-320]:

$$\delta\, y_k^j = \rho_q^j \exp(iqx_k),\ \ q = 0,\pm1,\pm2,...;\ \ i = \sqrt{-1}\,. \tag{9.18}$$

При цьому $\delta\, \hat{y}_k = \rho_q^{j+1} \exp(iqx_k) = \rho_q \cdot \delta\, y_k^j$, де $\rho_q$ – множник зростання $q$-ї гармоніки при переході з шару на шар. Враховуючи ознаку рівномірної стійкості (9.13), для схем з постійними операторами $B(t_j) = B,\ \ A(t_j) = A$, після підстановки (9.18) отримаємо ознаку стійкості за початковими даними у формі умови на *множник зростання* $\rho_q$ при довільному індексі гармоніки $q$

$$|\rho_q| \le 1 + C\tau,\ \ C \in const, \tag{9.19}$$

(частіше обирають $C = 0$).

Підставимо (9.18) в однорідну схему (9.4) і отримаємо вираз для значення множника зростання $\rho_q$

$$\rho_q = 1 - \frac{c\tau}{h}(1 - \exp(-iqh)).$$

Якщо умова Куранта (9.17) не виконується, то для тих гармонік, у яких $\cos(qh) = -1$, модуль множника зростання $|\rho_q| = \left|1 - \frac{2c\tau}{h}\right| = \frac{2c\tau}{h} - 1 > 1$, а отже амплітуди цих гармонік необмежено зростають і РС (9.4) не стійка. При виконанні умови (9.17), різницева задача (9.4), (9.8), (9.9) збігається до розв'язку диференціальної задачі зі швидкістю $O(h+\tau)$.

Для однорідної схеми (9.5):

$$\rho_q = 1 + \gamma(1 - \exp(iqh)) = 1 + \gamma(1 - \cos(qh)) - i\gamma \sin(qh),\ \ \gamma = \frac{c\tau}{h};$$

$$|\rho_q|^2 = 1 + 4\gamma(1+\gamma)\sin^2(0.5qh).$$

З останньої рівності видно, що $|\rho_q| > 1$ при $\forall \gamma$, а отже ця схема є *абсолютно нестійкою* і при $c > 0$ не використовується для розрахунків.

Для однорідної схеми (9.6):

$$\rho_q(1 + \gamma(1 - \exp(-iqh))) = 1;\ \ \Rightarrow\ \ |\rho_q| \le 1,\ \forall q,$$

а отже ця схема є *безумовно стійкою*. Різницева задача (9.6), (9.8), (9.9) збігається до розв'язку диференціальної задачі зі швидкістю $O(h+\tau)$.

Для однорідної схеми (9.7):

$$\rho_q = \exp(-iqh)\frac{(1+\gamma) + (1-\gamma)\exp(iqh)}{(1+\gamma) + (1-\gamma)\exp(-iqh)};\ \ \Rightarrow\ \ |\rho_q| = 1,\ \forall q,$$

а отже ця схема є *безумовно стійкою*. Критерій стійкості за правою частиною (9.16) виконується при $\theta = 2$. Різницева задача (9.7)–(9.9) безумовно збігається до розв'язку $u(x,t) \in C^{(3,3)}(D)$ диференціальної задачі (9.1)–(9.3) у нормі $\|\circ\|_{L_2}$ зі швидкістю $O(h^2 + \tau^2)$.

**Знаходження розв'язку.** Для РС з шаблонами **a)**, **c)**, **d)** маємо наступний алгоритм розрахунку:

1. Обираємо сітку і визначаємо незалежні від $k, j$ константи $A, B, C$;
2. Визначаємо сіткову функцію $y_k^0 = u0(x_k),\ \ k = \overline{1,N}$;

3. Послідовно, для $j = 0,1,...,M-1$ визначаємо

**a)** значення $y_1^{j+1} = \mu_1\left(t_{j+1}\right)$

**b)** сіткову функцію $y_k^{j+1}$

шаблон **a)** $y_k^{j+1} = A * y_{k-1} + B * y_k + C * \overline{\varphi}_{k,j}, \quad k = \overline{2,N};$
$A = \gamma, \ B = 1-\gamma, \ C = \tau.$

шаблон **c)** $y_k^{j+1} = A * y_{k-1}^{j+1} + B * y_k + C * \overline{\varphi}_{k,j}, \quad k = \overline{2,N};$
$B = \left(1+\gamma\right)^{-1}, \ A = \gamma * B, \ C = \tau * B.$

шаблон **d)** $y_k^{j+1} = A * \left(y_k - y_{k-1}^{j+1}\right) + y_{k-1} + C * \overline{\varphi}_{k,j}, \quad k = \overline{2,N};$
$B = \left(1+\gamma\right)^{-1}, \ A = \left(1-\gamma\right) * B, \ C = 2 * \tau * B.$

Тепер розглянемо варіант $c = const < 0$. У цьому випадку для рівняння (9.1) задається крайова умова у правій точці границі $\partial D_1$

$$u(b,t) = \mu_2(t), \quad t \in \partial D_2. \tag{9.20}$$

З розглянутих РС використовують явну схему (9.5), яка є умовностійкою при умові $|c|\tau \le h$ Неявну безумовно стійку схему записують у вигляді

$$\frac{\hat{y}_k - y_k}{\tau} + c\frac{\hat{y}_{k+1} - \hat{y}_k}{h} = \overline{\varphi}_{k,j}, ; \quad k = N-1, N-2,...,1. \tag{9.21}$$

У випадках, коли $u(x,t)$ недостатньо гладка або швидкозмінна, змінна $c(x,t) > 0$, виникає необхідність застосування нерівномірних сіток за $x,t$: $h_k = x_k - x_{k-1}$, $\tau_j = t_{j+1} - t_j$ і використовують безумовно стійку *явно-неявну* схему. Розрахунок нового значення $\hat{y}_k$ проводять наступним чином:

**1.** обчислюють спочатку умову Куранта (9.17) для даної комірки $\hat{c}_k\tau_j \le h_k$;

**2.** якщо вона виконується, то використовуємо явну схему

$$\frac{\hat{y}_k - y_k}{\tau_j} + \hat{c}_k\frac{y_k - y_{k-1}}{h_k} = \overline{\varphi}_{k,j}, \tag{9.22а}$$

інакше використовуємо неявну схему

$$\frac{\hat{y}_{k-1} - y_{k-1}}{\tau_j} + \hat{c}_k\frac{\hat{y}_k - \hat{y}_{k-1}}{h_k} = \overline{\varphi}_{k,j}, \text{ при } \quad \hat{c}_k\tau_j \le h_k. \tag{9.22б}$$

Цікаве порівняння схем (9.4), (9.6), (9.22б), (9.22) порядку $O(h+\tau)$ [1, с. 340]: неявна схема (9.6) досить надійна в розрахунках, але за точністю вона програє явній схемі (9.4) і неявній схемі (9.22б), а також безумовно стійкій явно-неявній схемі (9.22).

У випадку знакозмінного коефіцієнта $c(x,t)$ у рівнянні (9.1) задаються обидві крайові умови (9.2) і (9.20). Вихідне рівняння апроксимують неявною безумовно стійкою схемою, яку записують у вигляді

$$\frac{\hat{y}_k - y_k}{\tau} + \hat{c}_k\frac{\hat{y}_{k+1} - \hat{y}_{k-1}}{2h} = \overline{\varphi}_{k,j}, \quad k = \overline{2,N-1}. \tag{9.22}$$

Для розрахунків використовують *монотонну прогонку*, стійкість якої буде тільки при виконанні умови $|c|\tau \le h$.

Розглянуті РС можна узагальнити на *рівняння переносу з поглинанням*

$$\frac{\partial u(x,t)}{\partial t} + c\frac{\partial u(x,t)}{\partial x} + p * u(x,t) = f(x,t),$$
$$c = const > 0,\ \ p = const > 0.$$

а саме, використовуючи шаблон **a)** :

варіант 1: $\frac{\hat{y}_k - y_k}{\tau} + c\frac{y_k - y_{k-1}}{h} + p \cdot y_k = \overline{\varphi}_{k,j};$

варіант 2: $\frac{\hat{y}_k - y_k}{\tau} + c\frac{y_k - y_{k-1}}{h} + p \cdot \hat{y}_k = \overline{\varphi}_{k,j};$

і використовуючи шаблон **c)** :

$$\frac{\hat{y}_k - y_k}{\tau} + c\frac{\hat{y}_k - \hat{y}_{k-1}}{h} + p \cdot \hat{y}_k = \overline{\varphi}_{k,j}.$$

Стійкість РС варіантів 1, 2 шаблону **a)** може бути тільки при виконанні умови Куранта. Ці схеми відрізняються тільки способом апроксимації члена з поглинанням, але варіант 1 має слабку стійкість, а варіант 2 не тільки стійкий, але і хорошо обумовлений (похибки не зростають, а спадають при $t \to \infty$). Стійкість РС для шаблону **c)** – безумовна, і хорошо обумовлена.

Розглянуті нами РС можна узагальнити а на *багатовимірні рівняння переносу* [7, с. 344–353].

Для квазілінійних задач з $c(x,t,u)$, $f(x,t,u)$ побудова різницевих задач значно ускладнюється в зв'язку з виникненням *слабких розривів* (похідних розв'язку) і *сильних розривів* (самого розв'язку) [7, с. 354–366]. При наявності для таких задач певних фізичних *законів збереження*, необхідно будувати так звані *консервативні* різницеві схеми – схеми, для яких мають місце різницеві аналоги законів збереження.

**Поняття про консервативні схеми для квазілінійних рівнянь переносу**

Розглянемо побудову РС для *нев'язкого рівняння Бюргерса*, яке може бути представлене в двох, еквівалентних між собою, формах – *недивергентній* (9.23) і *дивергентній* (9.24)

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0, \tag{9.23}$$

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial\left(u^2\right)}{\partial x} = 0, \tag{9.24}$$

Покажемо, що для рівняння (4.24) має місце певний фізичний закон збереження. Проінтегруємо це рівняння по окремій комірці сітки $\overline{\omega}_{h\tau}$, і отримаємо точне інтегральне співвідношення

$$\int_{x_{n-1}}^{x_n}\left(u(x,t_{m+1}) - u(x,t_m)\right)dx + \frac{1}{2}\int_{t_m}^{t_{m+1}}\left(u^2(x_n,t) - u^2(x_{n-1},t)\right)dx = 0. \tag{9.25}$$

Аналогічно можна отримати співвідношення для всієї області $\overline{D}$

$$\int_a^b\left(u(x,T) - u(x,0)\right)dx + \frac{1}{2}\int_0^T\left(u^2(b,t) - u^2(a,t)\right)dt = 0. \tag{9.26}$$

Якщо співвідношення (9.25) просумувати по всіх комірках сітки $n = \overline{2,N}$, $m = \overline{0,M-1}$, і скористатися адитивністю інтеграла, то отримаємо (9.26). Отже маємо висновок: закон збереження у всій області є точне наслідування закону збереження в окремій комірці.

Тепер перейдемо до побудови різницевих рівнянь, які будуть апроксимувати (9.23), (9.24). Будемо вважати, що крайові і початкові умови для цих рівняння такі, що

коректно і однозначно визначають функцію $u(x,t)>0$ і не мають ні *слабких розривів розв'язку* (похідних $u(x,t)$), а ні *сильних розривів* (самої функції $u(x,t)$).

Використаємо шаблон **c)** і побудуємо аналоги схеми (9.6) для (9.23), (9.24)

$$\frac{\hat{y}_k - y_k}{\tau} + \hat{y}_k \frac{\hat{y}_k - \hat{y}_{k-1}}{h} = 0, \tag{9.27}$$

$$\frac{\hat{y}_k - y_k}{\tau} + \frac{1}{2}\frac{(\hat{y}_k)^2 - (\hat{y}_{k-1})^2}{h} = 0. \tag{9.28}$$

Спробуємо отримати різницевий аналог закону (9.26) для наших схем і почнемо з (9.28)

$$\sum_{k=2}^{N}\sum_{j=0}^{M-1}\left(\frac{\hat{y}_k - y_k}{\tau} + \frac{1}{2}\frac{(\hat{y}_k)^2 - (\hat{y}_{k-1})^2}{h}\right) * h\tau = 0;$$

$$h\sum_{k=2}^{N}\left(y_k^M - y_k^0\right) + \frac{1}{2}\tau\sum_{j=0}^{M-1}\left((\hat{y}_N)^2 - (\hat{y}_1)^2\right) = 0.$$

Останнє співвідношення є аналогом (9.26) для сіткової функції, де перший інтеграл обчислено по КФ правих прямокутників, а другий – по КФ лівих прямокутників.

***Означення 9.8.*** Різницева схема називається *консервативною*, якщо для неї виконується сітковий аналог закону збереження диференціальної задачі. РС, для якої це не виконується, називається *неконсервативною* [7, с. 363-366].

Виконаємо аналогічні перетворення для схеми (9.27):

$$\sum_{k=2}^{N}\sum_{j=0}^{M-1}\left(\frac{\hat{y}_k - y_k}{\tau} + \hat{y}_k\frac{\hat{y}_k - \hat{y}_{k-1}}{h}\right) * h\tau = 0;$$

$$h\sum_{k=2}^{N}\left(y_k^M - y_k^0\right) + \tau\sum_{k=2}^{N}\sum_{j=0}^{M-1}\left((\hat{y}_k)^2 - \hat{y}_k\hat{y}_{k-1} \pm \frac{1}{2}(\hat{y}_{k-1})^2\right) = 0;$$

$$h\sum_{k=2}^{N}\left(y_k^M - y_k^0\right) + \frac{1}{2}\tau\sum_{j=0}^{M-1}\left((\hat{y}_N)^2 - (\hat{y}_1)^2\right) + \Delta = 0,$$

де

$$\Delta = \frac{1}{2}\tau\sum_{k=2}^{N}\sum_{j=0}^{M-1}(\hat{y}_k - \hat{y}_{k-1})^2 > 0$$

це *дисбаланс*, на який порушується сітковий аналог закону (9.26), а отже схема (9.27) розв'язує ІНШУ фізичну задачу!

*Завдання 3.* Використати шаблон **a)** для побудови РС, яка апроксимує рівняння (9.24), і дослідити її на консервативність.

# ТЕМА 10

# РС ДЛЯ РОЗВ'ЯЗКУ КРАЙОВИХ ЗАДАЧ ДЛЯ РІВНЯНЬ ПАРАБОЛІЧНОГО ТИПУ

## Різницеві схеми для лінійних одновимірних задач

Потрібно знайти неперервний в області $\overline{D} = D \cup \partial D = [a,b] \times [0,T]$ розв'язок *мішаної задачі Коші* для рівняння параболічного типу:

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2} + f(x,t), \quad (x,t) \in D \equiv (a,b) \times (0,T], \tag{10.1}$$

з крайовими умовами першого роду

$$u(a,t) = \mu_1(t), \quad u(b,t) = \mu_2(t), \quad t \in [0,T] \tag{10.2}$$

і початковою умовою

$$u(x,0) = u0(x), \quad x \in [a,b]. \tag{10.3}$$

Будемо вважати, що:

- початкові і крайові дані мають достатню гладкість і для них виконуються необхідні *умови узгодження* в точках $(a;0)$, $(b;0)$;
- функція $f$ неперервно диференційована;
- шукана функція $u$ також достатньо гладка.

**Побудова сімейства різницевих схем.** Уведемо рівномірні сітки

$$\overline{\omega}_h = \{x_k = a + (k-1)h, \quad k = \overline{1,N}, \quad b - a = (N-1)h\};$$
$$\overline{\omega}_\tau = \{t_j = j\tau, \quad j = \overline{0,M}, \quad T = M\tau\},$$

і замінимо область $\overline{D}$ сітковою область $\overline{\omega}_{h\tau} = \overline{\omega}_h \times \overline{\omega}_\tau$, а також будемо використовувати наступні позначення [17, с. 276-282]

$$\dot{u} = \frac{\partial u}{\partial t}, \quad Lu = \frac{\partial^2 u}{\partial x^2}; \quad \dot{u} = Lu + f, \quad L\dot{u} = L^2 u + Lf, \quad L^2 u = L\dot{u} - Lf;$$

$$y_{t,j} = \frac{\hat{y}_k - y_k}{\tau}, \quad \Lambda y = y_{\bar{x}x,k};$$

$$0 \le \sigma \le 1; \quad y^{(\sigma)} \equiv Y = \sigma \cdot \hat{y} + (1-\sigma) \cdot y = 0.5(\hat{y} + y) + \tau(\sigma - 0.5) y_{t,j}.$$

На шеститочковому шаблоні $S\{(x_{k-1},t_j),(x_k,t_j),(x_{k+1},t_j),(x_{k-1},t_{j+1}),(x_k,t_{j+1}),(x_{k+1},t_{j+1})\}$ рівнянню (10.1) поставимо у відповідність однопараметричну сім'ю двошарових схем – *схему з вагою*

$$y_{t,j} = \Lambda Y + \varphi, \quad k = \overline{2,N-1}; \; j = \overline{0,M-1}. \tag{10.4}$$

Крайові і початкові умови апроксимуються точно

$$y_1^j = \mu_1(t_j), \quad y_N^j = \mu_2(t_j), \quad j = \overline{1,M}; \tag{10.5}$$

$$y_k^0 = u0(x_k), \quad k = \overline{1,N}. \tag{10.6}$$

Від вибору параметра $\sigma$ залежить точність і стійкість схеми (10.4). Розглянемо найбільш уживані частинні випадки параметра $\sigma$.

*Явна схема*. При $\sigma = 0$ на шаблоні $S\{(x_{k\mp1},t_j),(x_k,t_j),(x_k,t_{j+1})\}$ схему (10.4) запишемо у формі (9.10)

$$\frac{1}{\tau}y_k^{j+1}=\frac{1}{h^2}y_{k-1}^{j}+\left(\frac{1}{\tau}-\frac{2}{h^2}\right)y_k^{j}+\frac{1}{h^2}y_{k+1}^{j}+\varphi, \qquad (10.7)$$
$$k=2,\ldots,N-1,$$

де введено позначення

$$\varphi\equiv\bar{\varphi}_{k,j}=f\left(x_k,\bar{t}_j\right),\ \bar{t}_j\equiv t_{j+0.5}=t_j+0.5\tau\,.$$

Використовуючи формулу (10.7) послідовно для шарів $j=\overline{1,M-1}$ і додаючи умови (10.5), (10.6), можемо знайти сіткову функцію для всієї області $\bar{\omega}_{h\tau}$.

*Неявна схема*. При $0<\sigma<1$ схема (10.4), записана у формі (9.10), має вигляд

$$-\frac{\sigma}{h^2}y_{k-1}^{j+1}+\left(\frac{1}{\tau}+\frac{2\sigma}{h^2}\right)y_k^{j+1}-\frac{\sigma}{h^2}y_{k+1}^{j+1}=\frac{1-\sigma}{h^2}y_{k-1}^{j}+\left(\frac{1}{\tau}-\frac{2(1-\sigma)}{h^2}\right)y_k^{j}+\frac{1-\sigma}{h^2}y_{k+1}^{j}+\varphi, \qquad (10.8)$$
$$k=2,\ldots,N-1.$$

*Чисто неявна схема*. При $\sigma=1$ на шаблоні $S\{(x_k,t_j),(x_{k\mp1},t_{j+1}),(x_k,t_{j+1})\}$ схему (10.4) запишемо у формі (9.10)

$$-\frac{1}{h^2}y_{k-1}^{j+1}+\left(\frac{1}{\tau}+\frac{2}{h^2}\right)y_k^{j+1}-\frac{1}{h^2}y_{k+1}^{j+1}=\frac{1}{\tau}y_k^{j}+\varphi, \qquad (10.9)$$
$$k=2,\ldots,N-1.$$

*Симетрична схема (схема Кранка-Ніколсон)*. При $\sigma=0.5$ схема (10.4), записана у формі (9.10), має вигляд

$$-\frac{0.5}{h^2}y_{k-1}^{j+1}+\left(\frac{1}{\tau}+\frac{1}{h^2}\right)y_k^{j+1}-\frac{0.5}{h^2}y_{k+1}^{j+1}=\frac{0.5}{h^2}y_{k-1}^{j}+\left(\frac{1}{\tau}-\frac{1}{h^2}\right)y_k^{j}+\frac{0.5}{h^2}y_{k+1}^{j}+\varphi, \qquad (10.10)$$
$$k=2,\ldots,N-1.$$

Для неявних схем зі значенням $0<\sigma\le1$, визначення $y_k^{j+1}$ на новому шарі передбачає розв'язання СЛАР, яка утворюється відповідно рівняннями (10.8) | (10.9) | (10.10) і умовами (10.5). В програмній реалізації зазвичай рівняння (10.8) | (10.9) | (10.10) домножаються на $\tau$. Ці СЛАР ефективно розв'язується *монотонною прогонкою*, яка є стійкою.

**Похибка апроксимації.** Позначимо через $z=y-u\left(x_k,t_j\right)$ *похибку розв'язку* і підставимо $y=z+u\left(x_k,t_j\right)$ у РЗ (10.4)–(10.6). Отримаємо для $z$ різницеву задачу

$$\begin{cases} z_{t,j}=\Lambda Z+\psi, \quad k=\overline{2,N-1},\ j=\overline{0,M-1};\\ z_1^j=0,\ \ z_N^j=0,\ \ j=\overline{1,M};\\ z_k^0=0,\ \ k=\overline{1,N}, \end{cases} \qquad (10.11)$$

де $\psi$ *– похибка на розв'язку* задачі (10.1)–(10.3) має вигляд

$$\psi=\Lambda u^{(\sigma)}+\varphi-u_{t,j}. \qquad (10.12)$$

Знайдемо її величину в околі точки $\left(x_k,\bar{t}_j\right)$. Враховуючи, що:

$$\hat{u}=\bar{u}+\frac{\tau}{2}\dot{\bar{u}}+\frac{\tau^2}{8}\ddot{\bar{u}}+\frac{\tau^3}{48}\dddot{\bar{u}}+O\left(\tau^4\right),\ u=\bar{u}-\frac{\tau}{2}\dot{\bar{u}}+\frac{\tau^2}{8}\ddot{\bar{u}}-\frac{\tau^3}{48}\dddot{\bar{u}}+O\left(\tau^4\right);$$
$$u^{(\sigma)}=0.5(\hat{u}+u)+\tau(\sigma-0.5)u_{t,j},\ 0.5(\hat{u}+u)=\bar{u}+O\left(\tau^2\right),\ u_{t,j}=\dot{\bar{u}}+O\left(\tau^2\right);$$
$$\Lambda u=Lu+h^2/12\cdot L^2u+O\left(h^4\right);$$

отримаємо наступний ланцюжок перетворень для $\psi$ :

$$\psi=L\bar{u}+h^2/12\cdot L^2\bar{u}+\tau(\sigma-0.5)L\dot{\bar{u}}+\varphi-\dot{\bar{u}}+O\left(h^4+\tau^2\right)=$$
$$=\left(L\bar{u}-\dot{\bar{u}}+\bar{f}\right)+\left(\varphi-\bar{f}\right)+h^2/12\cdot\left(L\ddot{\bar{u}}-L\bar{f}\right)+\tau(\sigma-0.5)L\dot{\bar{u}}+O\left(h^4+\tau^2\right)=$$

$$= \left(\varphi - \bar{f}\right) + h^2/12 \cdot \left(L\bar{u} - L\bar{f}\right) + \tau(\sigma - 0.5) L\bar{u} + O\left(h^4 + \tau^2\right) =$$
$$= \left(h^2/12 + \tau(\sigma - 0.5)\right) L\bar{u} + \left(\varphi - \bar{f} - h^2/12 \cdot L\bar{f}\right) + O\left(h^4 + \tau^2\right)$$

і аналізуючи отримане співвідношення, маємо такі оцінки для порядку похибки

$$\psi = \begin{cases} O\left(h^2 + \tau\right), & \varphi = \bar{f} \ \ \& \ \ \sigma \neq 0.5 \ \ \& \ \ 0 \le \sigma \le 1; \\ O\left(h^2 + \tau^2\right), & \varphi = \bar{f} \ \ \& \ \ \sigma = 0.5; \\ O\left(h^4 + \tau^2\right), & \varphi = \bar{f} + h^2/12 \cdot L\bar{f} \ \ \& \ \ \sigma = \sigma_* = 0.5 - h^2/12 \cdot \tau^{-1}. \end{cases} \tag{10.13}$$

Схема з $\sigma = \sigma_*$ називається *схемою підвищеного порядку точності*.

За наявності у крайових умовах (10.2) похідних, для отримання необхідного порядку апроксимації можна використовувати як *метод законтурних точок*, так і *метод підвищення порядку апроксимації на розв'язку*.

**Стійкість і збіжність різницевих схем.** *Дослідження стійкості* РС виконаємо для задачі (10.4)–(10.6) двома способами: принципом максимуму (у нормі $\|\circ\|_C$) і методом відокремлення змінних (у нормі $\|\circ\|_{L_2}$).

Застосуємо *принцип максимуму*. Порівнюючи формули однорідної схеми (10.8) і (9.16), отримаємо

$$\alpha_0 = \frac{1}{\tau} + \frac{2\sigma}{h^2},\ \alpha_{-1} = \alpha_1 = -\frac{2\sigma}{h^2},\ \beta_0 = \frac{1}{\tau} - \frac{2(1-\sigma)}{h^2},\ \beta_{-1} = \beta_1 = \frac{1-\sigma}{h^2}.$$

Підставимо ці значення в (9.15) при $C = 0$, отримаємо нерівність $\frac{1}{\tau} - \frac{2(1-\sigma)}{h^2} \ge \left|\frac{1}{\tau} - \frac{2(1-\sigma)}{h^2}\right|$, яка виконується завжди (безумовно) при $\sigma = 1$. Для схем з $\sigma \in [0,1)$ маємо умову $\tau \le \frac{h^2}{2(1-\sigma)}$, а отже умовну стійкість за початковими даними. Умова (9.16) при $\theta = 1$ виконується для будь-яких співвідношень кроків $h, \tau$ і значень $\sigma$.

Тепер розглянемо *метод відокремлення змінних*. Якщо підставимо (9.18) в однорідну схему (10.8), то отримаємо вираз для множника зростання $q$-ї гармоніки при переході з шару на шар:

$$\rho_q = \frac{1 - (1-\sigma) 4h^{-2}\tau \sin^2(0.5qh)}{1 + \sigma 4h^{-2}\tau \sin^2(0.5qh)}.$$

З останньої рівності видно, що $\rho_q < 1$ при $\sigma \ge 0$. Залишається перевірити виконання нерівності $-1 \le \rho_q$. З отриманої формули для $\rho_q$, маємо ланцюжок нерівностей

$$\sigma \ge \frac{1}{2} - \frac{h^2}{4\tau \sin^2(0.5qh)} \ge \frac{1}{2} - \frac{h^2}{4\tau}. \tag{10.14}$$

Таким чином, для явної схеми ($\sigma = 0$) маємо умовну стійкість при виконанні нерівності $\tau \le 0.5h^2$. Виконання нерівності (10.14) для значень $\sigma = \{1;\ 0.5;\ \sigma_*\}$ легко перевіряється, а отже маємо *безумовну* стійкість. Критерій стійкості за правою частиною (9.16) при $\theta = 1$ також виконується.

Різницева задача (10.4) – (10.5), за умов виконання стійкості, відповідної гладкості вхідних даних і шуканої функції, збігається до розв'язку диференціальної задачі (10.1)–(10.3) у нормах $\|\circ\|_C$, $\|\circ\|_{L_2}$ з порядком, що співпадає з порядком апроксимації (10.13).

# ТЕМА 11

# РС ДЛЯ РОЗВ'ЯЗКУ КРАЙОВИХ ЗАДАЧ ДЛЯ РІВНЯНЬ ЕЛІПТИЧНОГО ТИПУ. ІТЕРАЦІЙНИЙ МЕТОД ЗЕЙДЕЛЯ РОЗВ'ЯЗАННЯ ВІДПОВІДНИХ СЛАР

## Різницеві схеми для рівняння Пуассона з крайовими умовами Діріхле

Розглянемо задачу Діріхле для рівняння Пуассона: знайти неперервну в прямокутній області $\overline{D} = D \cup \partial D = \{(x_1, x_2) \in [0, l_1] \times [0, l_2]\}$ функцію $u(x_1, x_2)$, яка справджує рівняння

$$\begin{gathered} \Delta u(x_1, x_2) = -f(x_1, x_2), \\ x \equiv (x_1, x_2) \in D, \end{gathered} \tag{11.1}$$

і крайові умови

$$u(x) = \mu(x), \quad x \in \partial D. \tag{11.2}$$

Тут введено позначення

$$\Delta u = L_1 u + L_2 u, \quad L_\alpha u = \frac{\partial^2 u}{\partial x_\alpha^2}, \ \alpha = 1,2.$$

Вважаємо при цьому, що при заданих функціях $f(x)$, $\mu(x)$, розв'язок поставленої задачі існує, єдиний і достатньо гладкий.

На відміну від крайових задач для рівнянь гіперболічного і параболічного типу (нестаціонарних), крайова задача (11.1), (11.2) – стаціонарна, а рівняння (11.1), з огляду класифікації ДРЧП, є еліптичного типу.

**Побудова різницевої схеми.** Крайові задачі, які розглядались у попередніх темах, записувались в однозв'язних просторових областях: відрізок, прямокутних, *p*-вимірний паралелепіпед. Відповідно і сіткові області $\overline{\omega}_h$ у різницевих схемах мали поділ на внутрішні сіткові вузли $\omega_h$ і граничні $\partial\omega_h$, при цьому жодних проблем з введенням сітки не виникало.

Якщо ми будемо розглядати постановку цих задач в однозв'язних плоских областях типу трикутник, трапеція, то тоді виникає певна «проблема граничних точок», а саме – помістити вузли сіткової границі $\partial\omega_h$ на границю $\partial D$ неперервної області так, щоб $\partial\omega_h \in \partial D$. Як варіанти вирішення цієї проблеми є:

a) введення рівномірних сіток, кроки яких зв'язані певною пропорцією $h_1/h_2 = \frac{m}{n}$;

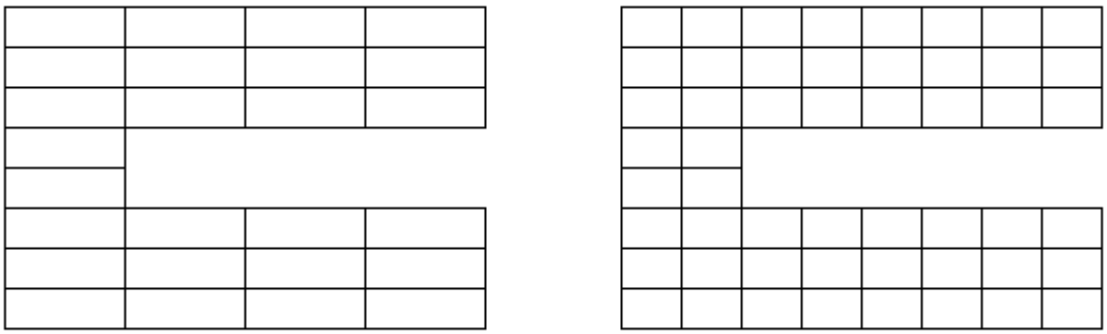
b) введення нерівномірних сіток.

Набагато складніші завдання постають при побудові сітки для розв'язання задач у довільних областях (спрощений варіант – область складена з прямокутників, трикутників, трапецій), як однозв'язних, так і неоднозв'язних [4, 6, 9-11]. Тут до «проблеми граничних точок» додається ще й обов'язкова вимога для сітки і сіткового шаблона $S(x)$, на якому записується РС – *умова зв'язності сітки*. Розглянемо ці поняття, які є загальними при побудові *p*-вимірних просторових сіток, як для стаціонарних, так і для нестаціонарних крайових задач.

Нехай у *p*-вимірному евклідовому просторі маємо скінченну множину точок – сітку $\Omega$. Кожній точці $x \in \Omega$ співставимо один і тільки один шаблон $S(x)$ – деяку підмножину точок $\Omega$, якій належить точка $x$.

***Означення 11.1.*** *Околом* точки $x$ назвемо множину $S'(x)=S(x)\setminus\{x\}$, причому ця множина може бути і пустою.

***Означення 11.2.*** Сітку $\Omega$ будемо називати *зв'язною сіткою*, якщо для довільних двох вузлів $x_0$, $x_0'$ таких, що хоча б один з них має непустий окіл, існує множина вузлів $x_i\in\Omega$, $i=1,2,...,m$, таких що $x_1\in S'(x_0)$, $x_2\in S'(x_1)$, …, $x_m\in S'(x_{m-1})$, $x_0'\in S'(x_m)$. Отже, з довільного вузла $x_0$ сітки $\Omega$, шаблон РС $S(x)$ можна послідовно перемістити («протягти», використовуючи точки сітки $\Omega$) у вузол $x_0'$. Наприклад, якщо маємо зображений на Рис.11.2 шаблон $S(x)$ для сітки $\Omega_1$ (рис.11.1), то з жодного вузла нижнього|верхнього прямокутника ми його не «протягнемо» через область, яка їх поєднує. Отже така сітка – *незв'язна*, на відміну від сітки $\Omega_2$ (рис.11.1).



$\Omega_1$ $\Omega_2$

Рис.11.1 Прямокутні сітки в однозв'язній області: $\Omega_1$ – незв'язна сітка; $\Omega_2$ – зв'язна.

Ці загальні вимоги до побудови просторових сіток ми будемо використовувати при розв'язанні практичних задач по даній темі.

Тепер перейдемо до побудови РС для задачі (11.1)–(11.2). Аналогічно двовимірним рівнянням параболічного типу, уведемо в просторовій області $\overline{D}$ прямокутну рівномірну сітку $\overline{\omega}_{\overline{h}}$:

$$\overline{\omega}_{\overline{h}}=\omega_{\overline{h}}\cup\partial\omega_{\overline{h}}=\overline{\omega}_{h_1}\times\overline{\omega}_{h_2},\ \overline{\omega}_{h_\alpha}=\left\{x_\alpha^{(k_\alpha)}=(k_\alpha-1)h_\alpha,\ k_\alpha=\overline{1,N_\alpha},\ l_\alpha=(N_\alpha-1)h_\alpha\right\},\ \alpha=1,2.$$

і для апроксимації диференціальних операторів (11.1) будемо використовувати мінімальну кількість точок $\overline{\omega}_{\overline{h}}$ – п'ятиточковий шаблон $S(x)=\{x\}\cup S'(x)$, $x=\left(x_1^{(k_1)},x_2^{(k_2)}\right)$, $S'(x)=\left\{\left(x_1^{(k_1\mp1)},x_2^{(k_2\mp1)}\right)\right\}$, зображений на рис.11.2.

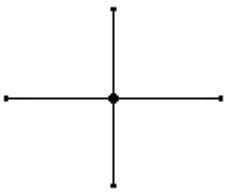


Рис.11.2 П'ятиточковий шаблон $S(x)=\{x\}\cup S'(x)$ з центральною точкою $x=\left(x_1^{(k_1)},x_2^{(k_2)}\right)$.

Задачі (11.1), (11.2) поставимо у відповідність наступну різницеву задачу:

$$\Lambda y(x)=-\varphi(x),\quad x\in\omega_{\overline{h}};\tag{11.3}$$

$$y(x)=\mu(x),\quad x\in\partial\omega_{\overline{h}},\tag{11.4}$$

де позначено

$$\Lambda y = \Lambda_1 y + \Lambda_2 y, \quad \Lambda_\alpha y = y_{(\bar{x}_\alpha x_\alpha), k_\alpha}, \quad \alpha = 1,2;$$
$$\varphi(x) - f(x) = O\left(|h|^2\right), \quad |h|^2 = h_1^2 + h_1^2.$$

**Похибка апроксимації.** Якщо $f \in C^{(2)}(D)$, то $\varphi(x) = f(x)$, $x \in \omega_{\bar{h}}$ і для похибки апроксимації (11.3) отримаємо, що він співпадає з похибкою апроксимації диференціального оператора різницевим:

$$\psi = (\Lambda y(x) - \varphi(x)) - (Lu(x) - f(x)) = \Lambda(x) - Lu(x), \quad x \in \omega_{\bar{h}}.$$

Враховуючи, що $\Lambda_\alpha u = L_\alpha u + \frac{h_\alpha^2}{12} L_\alpha^2 u + O\left(h_\alpha^4\right)$, маємо

$$\psi = \frac{h_1^2}{12} L_1^2 u + \frac{h_1^2}{12} L_1^2 u + O\left(h_1^4 + h_2^4\right), \quad \mathrm{u} \in C^{(4)}(\mathrm{D}),$$

а позначивши $M_4 = \max\limits_{\bar{D},\alpha} \left| \frac{\partial^4 u}{\partial x_\alpha^4} \right|$, отримаємо

$$|\psi| \le M_4 \frac{|h|^2}{12} = O\left(h_1^2 + h_2^2\right).$$

**Стійкость і збіжність РС.** Для доведення стійкості за крайовими умовами і за правою частиною, а також доведення збіжності, використовують *принцип максимуму* для РС, побудований на аналозі принципа максимуму для еліптичних задач у диференціальному вигляді. При проведенні цього дослідження, сімейство РС для стаціонарних задач записують у *канонічному вигляді* [18, с. 293]

$$A(\eta) y(\eta) = \sum_{\xi} B(\eta, \xi) y(\xi) + F(\eta), \quad \eta \in S(x), \quad \xi \in S'(x) \tag{11.5}$$

де коефіцієнти цього рівняння пов'язані певними співвідношеннями – *умовами додатності коефіцієнтів* [18, с. 294]

$$A(\eta) > 0, \quad B(\eta, \xi) > 0, \quad A(\eta) - \sum_{\xi} B(\eta, \xi) \ge 0.$$

Форма (11.5) при $x \in \omega_{\bar{h}}$ представляє різницеву схему (11.3). Для межових точок $x \in \partial\omega_{\bar{h}}$ шаблоном $S'(x)$ є пуста множина і (11.5) приймає вигляд $A(\eta) y(\eta) = F(\eta) \equiv A(\eta)\mu(\eta)$, а отже представляє крайові умови (11.4).

З огляду значного об'єму викладок по застосуванню принципа максимуму для еліптичних задач ми його розглядати не будемо.

Використовуючи відповідні теореми принципа максимуму, для різницевої задачі (11.3), (11.4) маємо їх стійкість за крайовими умовами і за правою частиною, а також збіжність з порядком, який відповідає порядку апроксимації [18, с. 294-304].

**Знаходження розв'язку.** Для стаціонарних різницевих задач типу (11.3), (11.4) досить складним є питання фактичного обчислення сіткової функції. Із ще більшими труднощами ми стикаємся при узагальненні цієї задачі для розмірностей $p$>2. Це зумовлено тим, що матриця **A** таких СЛАР характеризується:

- високим порядком – число невідомих $\approx N^p$ для $p$-вимірної схеми з $N$ інтервалами по кожній координаті;
- стрічковою будовою, при цьому стрічка слабко заповнена і має півширину $N^{p-1}$;
- великим числом обумовленості $cond(\mathbf{A}) \approx O\left(h^{-2}\right)$;
- реалізація *точних* методів (типу Гаусса) вимагає кількість операцій $\approx N^{3p-2}$.

У зв'язку з цим, для деяких частинних випадків еліптичних задач, розроблені спеціальні точні методи (детальний опис див. [16]) :

1. *швидке перетворення Фур'є*;
2. *метод повної редукції*;
3. *метод матричної прогонки*.

У випадку складних областей $\overline{D}$, або загальних виразів диференціальних операторів для еліптичних задач, отримані РЗ розв'язують за допомогою ітераційних методів (ІМ). Почнемо з простих за реалізацією методів – *Якобі* і *Зейделя*. З огляду представлення ІМ у вигляді *однокрокового ІП в канонічному вигляді*:

$$\mathbf{B}\frac{\vec{y}^{(s+1)}-\vec{y}^{(s)}}{\tau_{(s+1)}}+\mathbf{A}\vec{y}^{(s)}=\vec{f},$$

маємо: метод Якобі – *явний*, а метод Зейделя – *неявний*, при цьому матриця $\mathbf{B}$ – трикутна, що дає можливість легко знайти обернену. Ці методи мають досить прості алгоритми, які можна легко реалізувати.

*Алгоритм ІМ Якобі* для системи (11.3), (11.4) записуємо у вигляді:

**1.** $H=2\left(h_1^2+h_2^2\right),\ A=h_2^2/H,\ B=h_1^2/H,\ C=h_1^2h_2^2/H;$

**2.** $y(x)=\mu(x),\quad x\in\partial\omega_{\bar{h}};$

**3.** $s=0:\ y^{(0)}(x)=$ лінійна інтерполяція по вертикалі|горизонталі крайових умов, $x\in\omega_{\bar{h}};$

**4.** $s=0,1,2,...:$

$$y_{k_1,k_2}^{(s+1)}=A*\left(y_{k_1-1,k_2}^{(s)}+\underline{y_{k_1+1,k_2}^{(s)}}\right)+B*\left(y_{k_1,k_2-1}^{(s)}+\underline{y_{k_1,k_2+1}^{(s)}}\right)+C*f_{k_1,k_2},\qquad k_1=2,N_1-1,\ k_2=2,N_2-1; \tag{11.6}$$

до виконання умови $\left\|y^{(s+1)}-y^{(s)}\right\|_C\le\varepsilon,\ 0<\varepsilon<1;$ кількість ітерацій $s\approx O\left(|h|^{-2}\right).$

*Алгоритм ІМ Зейделя* для системи (11.3), (11.4) записуємо аналогічно, змінюємо тільки розрахункову формулу (11.6) на наступну:

$$y_{k_1,k_2}^{(s+1)}=A*\left(y_{k_1-1,k_2}^{(s+1)}+\underline{y_{k_1+1,k_2}^{(s)}}\right)+B*\left(y_{k_1,k_2-1}^{(s+1)}+\underline{y_{k_1,k_2+1}^{(s)}}\right)+C*f_{k_1,k_2},\qquad k_1=2,N_1-1,\ k_2=2,N_2-1; \tag{11.7}$$

Якщо сіткову функцію зберігати у вигляді прямокутної матриці, де $k_1$ і $k_2$ – індекси рядка і стовпця, то реалізація цих алгоритмів дійсно проста, але швидкість збіжності невисока.

Розглянемо ще одного представника неявних ІМ – *метод релаксації* $\tau_{(s+1)}=\omega,$ $\omega=const\in(0,2)$. При значенні $\omega=1$ він співпадає з методом Зейделя, а при $\omega\in(1,2)$ має назву *методу верхньої релаксації*. Для його реалізації змінимо в алгоритмі методу Зейделя

1. $H=2\left(h_1^{-2}+h_2^{-2}\right),\ C=\omega/H,\ A=h_1^{-2}C,\ B=h_2^{-2}C,\ D=1-\omega.$

і розрахункову формулу (11.7) на наступну

$$y_{k_1,k_2}^{(s+1)}=A*\left(y_{k_1-1,k_2}^{(s+1)}+y_{k_1+1,k_2}^{(s)}\right)+B*\left(y_{k_1,k_2-1}^{(s+1)}+\underline{y_{k_1,k_2+1}^{(s)}}\right)+C*f_{k_1,k_2}+D*y_{k_1,k_2}^{(s)},\qquad k_1=2,N_1-1,\ k_2=2,N_2-1; \tag{11.8}$$

Для задачі Діріхле при умові $(h_1=h_2=h)\&(l_1=l_2=l)$, відоме оптимальне значення $\omega$, яке визначається за формулою $\omega_{opt}=\left(0.5+\sin\left(0.5\pi*h/l\right)\right)^{-1}$ і число ітерацій $\approx O\left(|h|^{-1}\right).$

Звичайно є і потужніші (за числом ітерацій) ітераційні методи розв'язання РС для еліптичних крайових задач.

# ТЕМА 12

# РС ДЛЯ РОЗВ'ЯЗКУ КРАЙОВИХ ЗАДАЧ ДЛЯ РІВНЯНЬ ГІПЕРБОЛІЧНОГО ТИПУ ДРУГОГО ПОРЯДКУ

## РС для одновимірних рівнянь другого порядку гіперболічного типу

Розглянемо в області $\overline{D} = D \cup \partial D = [a,b]\times[0,T]$ крайову задачу для рівняння гіперболічного типу (фізична модель – коливання однорідної струни [2, с. 27]):

$$\frac{\partial^2 u(x,t)}{\partial t^2} = \frac{\partial^2 u(x,t)}{\partial x^2} + f(x,t), \quad (x,t) \in D \equiv (a,b)\times(0,T], \tag{12.1}$$

з крайовими умовами першого роду

$$u(a,t) = \mu_1(t), \quad u(b,t) = \mu_2(t), \quad t \in [0,T] \tag{12.2}$$

і початковими умовами

$$u(x,0) = u0(x), \tag{12.3}$$

$$\frac{\partial u(x,0)}{\partial t} = u1(x), \quad x \in [a,b]. \tag{12.4}$$

Будемо вважати, що:

- діні початкових і крайових умов мають достатню гладкість і для них виконуються необхідні *умови узгодження* в точках $(a;0)$, $(b;0)$;
- функція $f$ неперервно диференційована;
- шукана функція $u$ також достатньо гладка,

а отже розв'язок задачі (12.1)–(12.4) існує, єдиний і неперервно залежить від вхідних даних.

**Побудова сімейства різницевих схем.** Уведемо рівномірні сітки

$$\overline{\omega}_{h\tau} = \overline{\omega}_h \times \overline{\omega}_\tau; \quad \overline{\omega}_h = \{x_k = a + (k-1)h, \quad k = \overline{1,N}, \quad b - a = (N-1)h\};$$
$$\overline{\omega}_\tau = \{t_j = j\tau, \quad j = \overline{0,M}, \quad T = M\tau\},$$

і замінимо область $\overline{D}$ сітковою область $\overline{\omega}_{h\tau}$, а також будемо використовувати наступні позначення [17, с. 332-335]

$$\dot{u} = \frac{\partial u}{\partial t}, \quad \ddot{u} = \frac{\partial^2 u}{\partial t^2}, \quad Lu = \frac{\partial^2 u}{\partial x^2}; \quad \ddot{u} = Lu + f, \quad L\ddot{u} = L^2 u + Lf, \quad L^2 u = L\ddot{u} - Lf;$$
$$y_{t,j} = \frac{\hat{y}_k - y_k}{\tau}, \quad y_{\bar{t}t,j} = \frac{y_{t,j} - y_{\bar{t},j}}{\tau} = \frac{\hat{y}_k - 2y_k - \breve{y}_k}{\tau^2}; \quad \Lambda y = y_{\bar{x}x,k};$$
$$0 \le \sigma \le \tfrac{1}{2}; \quad y^{(\sigma)} \equiv Y = \sigma\hat{y} + (1-2\sigma)y + \sigma\breve{y} = y + \sigma\tau^2 y_{\bar{t}t,j}.$$

На дев'ятиточковому шаблоні (рис. 12.1)

$$S = S\{(x_{k-1},t_{j-1}),(x_k,t_{j-1}),(x_{k+1},t_{j-1}),(x_{k-1},t_j),(x_k,t_j),(x_{k+1},t_j),(x_{k-1},t_{j+1}),(x_k,t_{j+1}),(x_{k+1},t_{j+1})\}$$

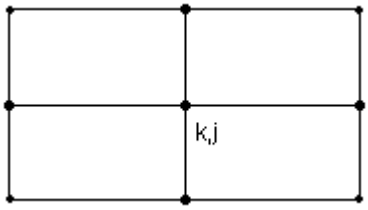



Рис.12.1 Дев'ятиточковий шаблон $S$ з центральною точкою $(x_k, \tau_j)$.

рівнянню (12.1) поставимо у відповідність однопараметричну сім'ю *тришарових схем*

$$y_{\bar{t}t,j} = \Lambda Y + \varphi, \quad k = \overline{2,N-1}; \; j = \overline{1,M-1}. \tag{12.5}$$

Крайові умови (12.2) і початкова умова (12.3) апроксимуються точно

$$y_1^j = \mu_1(t_j), \quad y_N^j = \mu_2(t_j), \quad j = \overline{1, M}; \tag{12.6}$$

$$y_k^0 = u0(x_k), \quad k = \overline{1, N}. \tag{12.7}$$

Для отримання апроксимації умови (12.4) з другим порядком по $\tau$ (враховуючи, що $y_{\bar{t}t,j} = \ddot{u}(x_k, \tau_j) + O(\tau^2)$), використаємо метод законтурних точок:

$$\begin{cases} \dfrac{y_k^1 - y_k^{-1}}{2\tau} = u1(x_k), \quad O(\tau^2), \\ \dfrac{y_k^1 - 2y_k^0 + y_k^{-1}}{\tau^2} = y_{\bar{x}x,k}^0 + f(x_k, 0), \quad O(\tau^2 + h^2). \end{cases}$$

З цієї системи виключаємо $y_k^{-1}$ і отримуємо таке рівняння

$$y_k^1 = y_k^0 + \tau * u1(x_k) + 0.5\tau^2 \left( y_{\bar{x}x,k}^0 + f(x_k, 0) \right), \quad k = \overline{2, N-1}. \tag{12.8}$$

Від вибору параметра $\sigma$ залежить точність і стійкість схеми (12.5), тому розглянемо найбільш уживані частинні випадки значень цього параметра.

*Явна схема*. При $\sigma = 0$ на шаблоні *"хрест"* (на рис. 12.1 позначений «жирнішими» крапками) $S\{(x_{k-1}, t_j), (x_k, t_j), (x_{k+1}, t_j), (x_k, t_{j-1}), (x_k, t_{j+1})\}$ схему (12.5) запишемо у вигляді наступної розрахункової формули

$$y_k^{j+1} = 2y_k^j - y_k^{j-1} + \tau^2 \left( y_{\bar{x}x,k}^j + f \right), \quad k = \overline{2, N-1}, \; j = \overline{1, M-1}, \tag{12.9}$$

де $\varphi = f \equiv f_{k,j} = f(x_k, t_j)$.

Алгоритм розрахунку:

1. Визначаємо сіткову функцію $y_k^0$, $k = \overline{1, N}$ за формулою (12.7).
2. Знаходимо $y_k^1$, $k = \overline{1, N}$, використовуючи формули (12.6) при $j = 1$ і (12.8).
3. Використовуючи формули (12.6) і (12.9), послідовно для $j = \overline{1, M-1}$, знаходимо сіткову функцію на решті шарів сітки $\overline{\omega}_\tau$.

*Неявна схема*. При $0 < \sigma \le \frac{1}{2}$ схему (12.5) запишемо у вигляді:

$$\sigma\gamma^2 y_{k-1}^{j+1} - (1 + 2\sigma\gamma^2) y_k^{j+1} + \sigma\gamma^2 y_{k+1}^{j+1} = -F_k, \quad k = \overline{2, N-1}, \tag{12.10}$$

де $\gamma = \tau / h$, а компоненти вектора $\vec{F}$ обчислюються наступним чином

$$F_k = \sigma\gamma^2 y_{\bar{x}x,k}^{j-1} - y_k^{j-1} + (1 - 2\sigma)\gamma^2 y_{\bar{x}x,k}^j + 2y_k^j + \tau^2 f.$$

Алгоритм розрахунку: Спочатку використовуємо описані вище пп. **1.-2.**, а для кожного значення $j = \overline{1, M-1}$ отримуємо СЛАР, яка утворюється формулами (12.6) і (12.10). Ці СЛАР ефективно розв'язуються *монотонною прогонкою*, яка є стійкою.

**Похибка апроксимації.** Позначимо через $z = y - u(x_k, t_j)$ *похибку розв'язку* і підставимо $y = z + u(x_k, t_j)$ в РЗ (12.5)–(12.8). Отримаємо для $z$ різницеву задачу

$$\begin{cases} z_{\bar{t}t,j} = \Lambda Z + \psi, \quad k = \overline{2, N-1}, \; j = \overline{0, M-1}; \\ z_1^j = 0, \quad z_N^j = 0, \quad j = \overline{1, M}; \\ z_k^0 = 0, \quad k = \overline{1, N}; \\ z_{t,0}(x_k) = \nu_k = O(\tau^2), \quad k = \overline{2, N-1}. \end{cases} \tag{12.11}$$

де $\psi$ – *похибка на розв'язку* задачі (12.1)–(12.4) має вигляд

$$\psi = \Lambda u^{(\sigma)} + \varphi - u_{\bar{t}t,j}. \tag{12.12}$$

Знайдемо її величину в околі точки $(x_k, t_j)$. Враховуючи, що:

$$\hat{u} = u + \tau * u_{t,j}, \; \breve{u} = u - \tau * u_{\bar{t},j}; \; u^{(\sigma)} = u + \sigma\tau^2 u_{\bar{t}t,j}; \; u_{\bar{t}t,j} = \ddot{u} + O(\tau^2); \; \Lambda u = Lu + h^2/12 \cdot L^2 u + O(h^4),$$

отримаємо наступний ланцюжок перетворень для $\psi$:

$$\psi = \Lambda u + \sigma\tau^2 \Lambda u_{\bar{t}t,j} + \varphi - u_{\bar{t}t,j} = Lu + h^2/12 \cdot L^2 u + \sigma\tau^2 L\ddot{u} + \varphi - \ddot{u} + O\left(h^4 + \tau^2\right) \pm f =$$
$$= \left(Lu - \ddot{u} + f\right) + \left(\varphi - f + \sigma\tau^2 Lf\right) + \left(h^2/12 + \sigma\tau^2\right) L^2 u + O\left(h^4 + \tau^2\right) =$$
$$= \left(\varphi - f + \sigma\tau^2 Lf\right) + \left(h^2/12 + \sigma\tau^2\right) L^2 u + O\left(h^4 + \tau^2\right).$$

Аналізуючи отримане співвідношення, маємо *оцінку похибки*

$$\psi = \begin{cases} O\left(h^2 + \tau^2\right), & \varphi = f \ \ \& \ \ \forall \sigma \in \left[0, \tfrac{1}{2}\right]; \\ O\left(h^4 + \tau^2\right), & \varphi = f - \sigma\tau^2 Lf \ \& \sigma = \sigma_*. \end{cases} \tag{12.13}$$

Схема з $\sigma = \sigma_* \equiv \tilde{\sigma} - h^2/12 \cdot \tau^{-2}$, $\tilde{\sigma} \in Const$, $\tilde{\sigma} \geq 0.25(1-\varepsilon)^{-1}$, $\varepsilon > 0$ називається *схемою підвищеного порядку точності*.

За наявності у крайових умовах (12.2) похідних, для отримання необхідного порядку апроксимації можна використовувати як *метод законтурних точок*, так і *метод підвищення порядку апроксимації на розв'язку*. Наприклад, розрахункові формули для крайових умов третього роду [17, с. 334–335].

**Стійкость і збіжність РС.** Дослідження стійкості для задачі (12.5)–(12.8) виконаємо методом Фур'є. Для цього підставимо $y_k^j = \rho_q^j \exp(iqx_k)$, $q = 0, \pm 1, \pm 2, \ldots$; $i = \sqrt{-1}$. в однорідну схему (12.5) та отримаємо для множника зростання $q$-ї гармоніки при переході з шару на шар квадратне рівняння $(\rho_q)^2 - 2\beta\rho_q + 1 = 0$, де $\beta = 1 - 2\sigma\theta^2/\left(1 + 4\sigma\theta^2\right)$, $\theta = \gamma \sin(0.5qh)$. Стійкість буде при $|\rho_q|_1 = |\rho_q|_2 = 1$, а отже отримуємо нерівність $|\beta| \leq 1$, що дає *умову стійкості*

$$\gamma^2(1 - 4\sigma) \leq 1 \text{ або } \sigma \geq 0.5(1+\varepsilon) - 0.25\gamma^{-2} \ \ \& \ \ \forall \varepsilon > 0. \tag{12.14}$$

Із цієї нерівності маємо висновки відносно *стійкості за початковими даними*:

- явна схема $\sigma = 0$ стійка при виконанні *умови Куранта* $\gamma \leq 1$;
- схема з $\sigma \in \left(0, \tfrac{1}{4}\right)$ також *умовно стійка* ($\gamma \leq \sqrt{1 - 4\sigma}$);
- схема з $\sigma \in [\tfrac{1}{4}, \tfrac{1}{2}]$ *абсолютно стійка*.

Для дослідження стійкості РЗ за правою частиною використовують *метод відокремлення змінних* і *принцип суперпозиції* [17, с. 335-343]. При виконанні наступного співвідношення між кроками сітки $\gamma^2 \leq (1+\varepsilon)^{-1}$, $\varepsilon > 0$ і для сіткової функції з однорідними крайовими умовами справедлива нерівність :

$$\left\|y^j\right\| \leq \sqrt{\frac{1+\varepsilon}{\varepsilon}} \left( \left\|y^0\right\| + \left\|y_{t,0}\right\|_{A^{-1}} + \sum_{m=0}^{j-1} \tau \left\|\varphi_{k,m}\right\|_{A^{-1}} \right),$$

якщо $\sigma \geq 0$. *Негативна норма* (норма в $H_{A^{-1}}$) для $A = -\Lambda$ в просторі $H$ (просторі сіткових функцій з $y_1 = 0,\ y_N = 0$) визначається за формулою

$$\|y\|_{A^{-1}} = \sqrt{\left(A^{-1}y, y\right)} = \sqrt{\sum_{k=2}^{N-1} \frac{(y_k)^2}{\lambda_k}}, \ \ \lambda_k = \frac{4}{h^2} \sin^2\left(\frac{\pi(k-1)h}{2}\right).$$

Розв'язок різницевої задачі (12.5)–(12.8), за умов виконання стійкості, відповідної гладкості вхідних даних і шуканої функції, збігається до розв'язку диференціальної задачі (12.1)–(12.4) у відповідних нормах з порядком, що співпадає з порядком апроксимації (12.13).

# ФОРМУЛИ ТА АЛГОРИТМИ РЕАЛІЗАЦІЇ ДЕЯКИХ ТЕМ

## 1. ОБЧИСЛЕННЯ ВИЗНАЧЕНОГО ІНТЕГРАЛА ДЛЯ ТАБЛИЧНО ЗАДАНОЇ ПІДІНТЕГРАЛЬНОЇ ФУНКЦІЇ

Розглянемо обчислення визначеного інтеграла

$$I = \int_a^b f(x)dx \tag{1}$$

для випадку табличного завдання на нерівномірній сітці підінтегральної функції :

$$\begin{aligned} &\vec{\mathbf{X}} = [X_i]_{i=0}^n, \;\; a = X_0 < X_1 < ... < X_{n-1} < X_n = b, \;\; h_i = X_i - X_{i-1}; \\ &\vec{\mathbf{F}} = [F_i]_{i=0}^n, \;\; F_i = f(X_i). \end{aligned} \tag{2}$$

**<u>Формула трапецій.</u>** Використовуючи адитивність інтеграла, для (1) спочатку отримаємо $I = \int_a^b f(x)dx = \sum_{i=1}^{n} \int_{X[i-1]}^{X[i]} f(x)dx$, а після заміни $f(x)$ на лінійну функцію

$$L_1(x) = F_{i-1}\frac{x - X_i}{X_{i-1} - X_i} + F_i\frac{x - X_{i-1}}{X_i - X_{i-1}} = h_i^{-1}\left[F_i(x - X_{i-1}) - F_{i-1}(x - X_i)\right]$$

і інтегрування на відрізку $[X_{i-1}, X_i]$, отримаємо таку розрахункову КФ [3, с. 87]

$$I \approx 0.5\sum_{i=1}^{n} h_i(F_{i-1} + F_i). \tag{3}$$

**<u>Формула Сімпсона.</u>** Для побудови цієї формули необхідно щоб *n* було парне !
Використовуючи адитивність інтеграла, спочатку для (1) отримаємо вираз

$$I = \int_a^b f(x)dx = \sum_{i=1,(2)}^{n-1} \int_{X[i-1]}^{X[i+1]} f(x)dx = \int_{X[0]}^{X[2]} f(x)dx + \int_{X[2]}^{X[4]} f(x)dx + ... + \int_{X[n-2]}^{X[n]} f(x)dx,$$

а після заміни $f(x)$ на відрізку $[X_{i-1}, X_{i+1}]$ на квадратичну функцію

$$L_2(x) = F_{i-1}\frac{(x - X_i)(x - X_{i+1})}{h_i(h_i + h_{i+1})} + F_i\frac{(x - X_{i-1})(x - X_{i+1})}{h_i(-h_{i+1})} + F_{i+1}\frac{(x - X_{i-1})(x - X_i)}{(h_i + h_{i+1})h_{i+1}}$$

і обчислення інтегралів (деталізуємо викладки тільки для того, який біля $F_{i-1}$) :

$$\int_{X[i-1]}^{X[i+1]} \frac{(x - X_i)(x - X_{i+1})}{h_i(h_i + h_{i+1})}dx = \int_{X[i-1]}^{X[i+1]} \frac{(x - X_i)^2 - h_{i+1}(x - X_i)}{h_i(h_i + h_{i+1})}d(x - X_i) =$$

$$= \left(\frac{(x - X_i)^3}{3h_i(h_i + h_{i+1})} - \frac{h_{i+1}(x - X_i)^2}{2h_i(h_i + h_{i+1})}\right)\Bigg|_{X[i-1]}^{X[i+1]} = \frac{h_{i+1}^3 - (-h_i)^3}{3h_i(h_i + h_{i+1})} - \frac{h_{i+1}(h_{i+1}^2 - h_i^2)}{2h_i(h_i + h_{i+1})} =$$

$$= \frac{2(h_i^2 - h_i h_{i+1} + h_{i+1}^2) - 3h_{i+1}(h_{i+1} - h_i)}{6h_i} = \frac{2h_i^2 + h_i h_{i+1} - h_{i+1}^2}{6h_i} = \frac{(h_i + h_{i+1})(2h_i - h_{i+1})}{6h_i};$$

$$\int_{X[i-1]}^{X[i+1]} \frac{(x - X_{i-1})(x - X_{i+1})}{h_i h_{i+1}}dx = -\frac{(h_i + h_{i+1})^3}{6h_i h_{i+1}}; \qquad \int_{X[i-1]}^{X[i+1]} \frac{(x - X_{i-1})(x - X_i)}{(h_i + h_{i+1})h_{i+1}}dx = \frac{(h_i + h_{i+1})(2h_{i+1} - h_i)}{6h_{i+1}};$$

отримаємо наступне наближення для (1)

$$I \approx \frac{1}{6}\sum_{i=1,(2)}^{n-1} \frac{h_i + h_{i+1}}{h_i h_{i+1}}\left[(2h_i - h_{i+1})h_{i+1}F_{i-1} + (h_i + h_{i+1})^2 F_i + (2h_{i+1} - h_i)h_i F_{i+1}\right].$$

Якщо ввести позначення $o_i = h_{i+1}/h_i$, то отримаємо таку розрахункову КФ

$$I \approx \frac{1}{6}\sum_{i=1,(2)}^{n-1} \left\{h_i(1 + o_i)\left[(2 - o_i)F_{i-1} + \frac{1}{o_i}(1 + o_i)^2 F_i + \left(2 - \frac{1}{o_i}\right)F_{i+1}\right]\right\}. \tag{4}$$

Для чисельної стійкості усі квадратурні коефіцієнти (4) мають бути додатні, отже маємо *умову стійкості* $0.5 \le o_i \le 2$. Таким чином, на відміну від формули трапецій, формулою Сімпсона можна скористатись не для всіх табличних завдань (2).

## 2. МЕТОД КУТТА-МЕРСОНА РОЗВ'ЯЗАННЯ ЗАДАЧІ КОШІ ДЛЯ СИСТЕМИ ЗДР

Розглянемо задачу Коші для системи *n* ЗДР

$$\begin{cases} \dfrac{d\vec{\mathbf{u}}(x)}{dx} = \vec{\mathbf{f}}(x, \vec{\mathbf{u}}(x)), \quad x \in (x_0, x_{end}], \\ \vec{\mathbf{u}}(x_0) = \vec{\mathbf{u}}_0 \end{cases} \tag{1}$$

і для її наближеного розв'язку будемо використовувати методи типу Рунге-Кутта 4-го порядку точності з *автоматичним вибором кроку інтегрування*. Зазвичай для цього застосовують метод *Рунге-Ромберга*, що стандартно приводить до **11**-разового обчислення $\vec{\mathbf{f}}(x, \vec{\mathbf{u}}(x))$. Але існують методи цієї групи, а саме метод *Кутта-Мерсона* (*п'ятиетапний метод Рунге-Кутта 4-го порядку точності*, або *метод вкладених форм*) [3, с. 551-556], який дозволяє скоротити виклики правої частини (1) до кількості **5**.

**Алгоритм методу Кутта-Мерсона.** [15, с. 180–181]

**1.** Обчислюємо початкові дані $\vec{\mathbf{y}} = \vec{\mathbf{u}}_0$, фіксуємо заданий користувачем крок $h_O = h_{old}$.

**2.** Циклічно, для $Xotr_i$ – кожної точки виведення результатів з діапазону знаходження розв'язку $(x_0, x_{end}]$, виконуємо наступні дії:

фіксуємо черговий інтервал знаходження розв'язку $[x_b, x_e]$; поточний крок $h = h_O$; початкові значення $x_O = x_b$, $\vec{\mathbf{y}}_O = \vec{\mathbf{y}}$.

**2.1.** Циклічно, поки $x < x_e$, виконуємо наступні дії:

обчислюємо значення нової точки інтегрування $x = x_O + h$ (при цьому, можливо, коригуємо значення $x$ і $h$ у випадку $x > x_e$); обнулюємо значення $kd$ – лічильника кількості поділів кроку.

**2.1.1.** Циклічно, виконуємо дії, направлені на знаходження розв'язку $\vec{\mathbf{y}}$ в новій точці інтегрування $x$ з реалізацією автоматичного вибору кроку.

Розрахунки проводимо за наступними формулами (векторний запис):

**2.1.1.1.** Значення $h3 = h/3$, коефіцієнти $\vec{\mathbf{K}}_{1,2,3,4,5}$ і поправка $\vec{\mathbf{w}}$

$$\begin{aligned}
&\vec{\mathbf{K}}_1 = h3 \cdot \vec{\mathbf{f}}(x_O, \vec{\mathbf{y}}_O); \\
&\vec{\mathbf{K}}_2 = h3 \cdot \vec{\mathbf{f}}(x_O + h3, \vec{\mathbf{y}}_O + \vec{\mathbf{K}}_1); \\
&\vec{\mathbf{K}}_3 = h3 \cdot \vec{\mathbf{f}}(x_O + h3, \vec{\mathbf{y}}_O + 0.5 \cdot (\vec{\mathbf{K}}_1 + \vec{\mathbf{K}}_2)); \\
&\vec{\mathbf{K}}_4 = h3 \cdot \vec{\mathbf{f}}(x_O + 0.5h, \vec{\mathbf{y}}_O + 0.375 \cdot (\vec{\mathbf{K}}_1 + 3 \cdot \vec{\mathbf{K}}_3)); \\
&\vec{\mathbf{w}} = \vec{\mathbf{y}}_O + 1.5 \cdot \vec{\mathbf{K}}_1 - 4.5 \cdot \vec{\mathbf{K}}_3 + 6 \cdot \vec{\mathbf{K}}_4; \\
&\vec{\mathbf{K}}_5 = h3 \cdot \vec{\mathbf{f}}(x, \vec{\mathbf{w}}).
\end{aligned}$$

**2.1.1.2.** $\vec{\mathbf{y}}$ – розв'язок в новій точці інтегрування $x$: $\vec{\mathbf{y}} = \vec{\mathbf{y}}_O + 0.5 \cdot (\vec{\mathbf{K}}_1 + 4 \cdot \vec{\mathbf{K}}_4 + \vec{\mathbf{K}}_5)$.

**2.1.1.3.** Покомпонентно, для $j = 0,1,..., n$, знаходимо

$d = 0.2 \cdot |w_j - y_j|$ – похибку обчислень;

$g = \begin{cases} \varepsilon, & |y_j| \le 1, \\ \varepsilon * |y_j|, & |y_j| > 1; \end{cases}$ – допоміжне значення;

$condition1 = d \le g$ – умова, що $\vec{\mathbf{y}}$ знайдено з необхідною точністю ε;

$condition2 = 32 \cdot d \le g$ – умова, що новий крок можна подвоїти.

**2.1.1.4.** Якщо $condition1$ виконано для всіх $j = 0,1,..., n$, то покладаємо $\vec{\mathbf{y}}_O = \vec{\mathbf{y}}$ для

точки $x_O = x$ і виходимо з циклу **2.1.1**, інакше зменшуємо крок вдвічі ($h=h/2$), лічильник *kd* збільшуємо на 1. У випадку *kd* > *kdiv* – перевищення заданої max-ї кількості поділів кроку при автоматичному виборі кроку даний алгоритм аварійно припиняється з відповідним кодом завершення.

**2.2.** Якщо *condition*2 виконано для всіх $j = 0,1,...,n$, то крок подвоюється $h = 2*h$о, але значення $h$ не може перевищувати $h_{\text{old}}$ – початковий крок.

Якщо *condition*2 не виконується для якогось $j = j_*$, то крок $h$ не змінюється.

## 3. МЕТОДИ ПОБУДОВИ РІЗНИЦЕВИХ ЗАДАЧ ПІДВИЩЕНОГО ПОРЯДКУ ТОЧНОСТІ ДЛЯ КРАЙОВИХ ЗАДАЧ ЗДР

Розглянемо крайову задачу для ЗДР

$$\begin{gathered} u''(x) - p\cdot u(x) = -f(x),\ x \in (a,b), \\ p \ge 0,\ \ x \in (a,b); \end{gathered} \tag{1}$$

$$\begin{gathered} -d_0 u'(a) + d_1 u(a) = d_2, \\ d_3 u'(b) + d_4 u(b) = d_5; \\ |d_0| + |d_1| \ne 0 \,\&\, |d_3| + |d_4| \ne 0, \\ d_0 = d_3 = 0 \,\&\, p \ne 0. \end{gathered} \tag{2}$$

Якщо вхідна функція $f(x)$ (і звичайно шукана $u(x)$) мають достатню гладкість, то поставимо собі мету: для (1), (2) отримати на триточковому шаблоні різницеву задачу з порядком апроксимації $O(h^4)$. Це дасть можливість глибше зрозуміти теорію метода підвищення апроксимації на розв'язку диференціальної задачі, а також отримати практичні навички його застосування.

Введемо рівномірну сітку

$$\overline{\omega}_h = \left\{ x_k = a + k\cdot h,\ k = \overline{0,n},\ b - a = nh \right\} \tag{3}$$

Апроксимацію рівняння (1) для внутрішніх точок $\overline{\omega}_h$, з урахуванням, що використовується шаблон $S\{x_{k-1}, x_k, x_{k+1}\}$, запишемо наступним чином [11, с. 92-93]

$$y_{\bar{x}x,k} - \xi_k y_k = -\varphi_k, \tag{4}$$

де сіткові функції $\xi_k$, $\varphi_k$ необхідно визначити таким чином, щоб порядок апроксимації був $O(h^4)$. Запишемо похибку і виконаємо ланцюжок нескладних перетворень з урахуванням розв'язку диференціальної задачі

$$\begin{aligned} \psi_h(x_k) &= \left[ \left( u''(x_k) - pu(x_k) \right) - \left( y_{\bar{x}x,k} - \xi_k y_k \right) \right] + \left[ -\varphi_k + f(x_k) \right] = \\ &= \left( -\varphi_k + f_k \right) - \left( p - \xi_k \right) u(x_k) - \frac{h^2}{12} u^{(4)}(x) + O(h^4) = \\ &= \left| \begin{array}{l} u'' = pu - f, \\ u''' = pu' - f', \\ u^{(4)} = pu'' - f'' = p(pu - f) - f'' = p^2 u - pf - f'' \end{array} \right| = \\ &= \left[ -\varphi_k + f_k + \frac{h^2}{12}\left( pf_k + f_k'' \right) \right] + \left[ \xi_k - p\left( 1 + p\frac{h^2}{12} \right) \right] u(x_k) + O(h^4). \end{aligned}$$

З останнього співвідношення при

$$\tilde{p} = 1 + \frac{h^2}{12} p, \quad \xi_k = p\cdot\tilde{p}, \quad \varphi_k = \tilde{p}\cdot f_k + \frac{h^2}{12} f_k'' \tag{5}$$

маємо $\psi_h(x_k) = O(h^4)$, а, отже, для (4) отримуємо таку СЛАР :

$$\begin{cases} A_k y_{k-1} - C_k y_k + B_k y_{k+1} = -F_k, \\ k = 1, n-1; \end{cases} \qquad (4^*)$$

де $A_k = B_k = 1,\ C_k = 2 + h^2 p \cdot \tilde{p},\ F_k = h^2\left(\tilde{p} \cdot f_k + \frac{h^2}{12} f_k''\right)$.

Систему (4*) доповнимо двома рівняннями, які апроксимують крайові умови (2) також з порядком $O(h^4)$. Тут доречно використати саме метод підвищення апроксимації на розв'язку диференціальної задачі, враховуючи наявність крайових умов III-го роду і необхідність отримати порядок $O(h^4)$.

Спочатку запишемо найпростіше, порядку $O(h^1)$, різницеве рівняння для апроксимації крайової умови в точці $x = a$:

$$-d_0 \frac{y_1 - y_0}{h} + d_1 y_0 = d_2, \qquad (6)$$

і для $y_1 = u(x_0 + h)$, за допомогою формули Тейлора знайдемо:

$$y_1 = u(x_0 + h) = u(a) + hu'(a) + \frac{h^2}{2}u''(a) + \frac{h^3}{6}u'''(a) + \frac{h^4}{24}u^{(4)}(a) + O(h^5). \qquad (7)$$

Підставимо (7) в (6) і виконаємо ланцюжок нескладних перетворень з урахуванням розв'язку диференціальної задачі:

$$-d_0 \frac{\left[u(a) + hu'(a) + \frac{h^2}{2}u''(a) + \frac{h^3}{6}u'''(a) + \frac{h^4}{24}u^{(4)}(a) + O(h^5)\right] - y_0}{h} + d_1 y_0 = d_2;$$

$$\left(-d_0 u'(a) + d_1 y_0 - d_2\right) - d_0\left(\frac{h}{2}u''(a) + \frac{h^2}{6}u'''(a) + \frac{h^3}{24}u^{(4)}(a)\right) + O(h^4) = 0;$$

$$-d_0\left[\frac{h}{2}\left(\underbrace{pu(a) - f(a)}_{u''(a)}\right) + \frac{h^2}{6}\left(\underbrace{pu'(a) - f'(a)}_{u'''(a)}\right) + \frac{h^3}{24}\left(\underbrace{p^2 u(a) - pf(a) - f''(a)}_{u^{(4)}(a)}\right)\right] + O(h^4) = 0;$$

$$-d_0 \frac{h}{2}\left(py_0 - f(a)\right) + \frac{h^2}{6}\left(p\left(\underbrace{d_2 - d_1 u(a)}_{-d_0 u'(a)}\right) + d_0 f'(a)\right) - d_0 \frac{h^3}{24}\left(p^2 y_0 - pf(a) - f''(a)\right) + O(h^4) = 0;$$

$$-d_0 \frac{h}{2}\left(py_0 - f(a)\right) - \frac{h^2}{6}\left(p\left(d_1 y_0 - d_2\right) - d_0 f'(a)\right) - d_0 \frac{h^3}{24}\left(p^2 y_0 - pf(a) - f''(a)\right) = O(h^4). \qquad (8)$$

Таким чином, щоб умова (6) була нульовою рівністю порядку $O(h^4)$, потрібно від (6) відняти ліву частину виразу (8) порядку $O(h^1)$, в результаті отримаємо рівняння :

$$-d_0 \frac{y_1 - y_0}{h} + d_1 y_0 +$$
$$+ d_0 \frac{h}{2}\left(py_0 - f(a)\right) + \frac{h^2}{6}\left(p\left(d_1 y_0 - d_2\right) - d_0 f'(a)\right) + d_0 \frac{h^3}{24}\left(p^2 y_0 - pf(a) - f''(a)\right) = d_2,$$

яке перетворимо до форми

$$-C_0 y_0 + B_0 y_1 = -F_0. \qquad (9)$$

Виконаємо ланцюжок нескладних перетворень і групувань коефіцієнтів

$$\frac{d_0}{h} y_1 - \frac{d_0}{h} y_0 - d_1 y_0 -$$
$$- d_0 \frac{h}{2}\left(py_0 - f(a)\right) - p\frac{h^2}{6}\left(d_1 y_0 - d_2\right) + d_0 \frac{h^2}{6} f'(a) - d_0 \frac{h^3}{24}\left(p^2 y_0 - pf(a) - f''(a)\right) = -d_2;$$

$$-\left(\frac{d_0}{h}+d_0 p\frac{h}{2}+d_0 p^2\frac{h^3}{24}+d_1+d_1 p\frac{h^2}{6}\right)y_0+\frac{d_0}{h}y_1=$$
$$=-\left(d_2+d_2 p\frac{h^2}{6}+d_0\frac{h}{2}f(a)+d_0 p\frac{h^3}{24}f(a)+d_0\frac{h^2}{6}f'(a)+d_0\frac{h^3}{24}f''(a)\right),$$

отримаємо:

$$C_0=\frac{d_0}{h}+d_0 p\frac{h}{2}\tilde{p}+d_1\tilde{\tilde{p}};\ \ B_0=\frac{d_0}{h};\ \ F_0=d_0\left(\frac{h}{2}\tilde{p}f(a)+\frac{h^2}{6}\left(\frac{h}{4}f''(a)+f'(a)\right)\right)+d_2\tilde{\tilde{p}},\tag{10}$$

де $\tilde{\tilde{p}}=1+p\frac{h^2}{6}$.

Аналогічно вчинимо з крайовою умовою в точці $x=b$ :

$$d_3\frac{y_n-y_{n-1}}{h}+d_4 y_n=d_5,\tag{11}$$

і для $y_{n-1}=u(x_n-h)$ за допомогою формули Тейлора знайдемо:

$$y_{n-1}=u(b-h)=u(b)-hu'(b)+\frac{h^2}{2}u''(b)-\frac{h^3}{6}u'''(b)+\frac{h^4}{24}u^{(4)}(b)+O\left(h^5\right).\tag{12}$$

Підставимо (12) в (11) і виконаємо ланцюжок нескладних перетворень з урахуванням розв'язку диференціальної задачі:

$$d_3\frac{y_n-\left[u(b)-hu'(b)+\frac{h^2}{2}u''(b)-\frac{h^3}{6}u'''(b)+\frac{h^4}{24}u^{(4)}(b)+O\left(h^5\right)\right]}{h}+d_4 y_n=d_5;$$

$$\left(d_3u'(b)+d_4y_n-d_5\right)-d_3\left(\frac{h}{2}u''(b)-\frac{h^2}{6}u'''(b)+\frac{h^3}{24}u^{(4)}(b)\right)+O\left(h^4\right)=0;$$

$$-d_3\left[\frac{h}{2}\left(\underbrace{pu(b)-f(b)}_{u''(b)}\right)-\frac{h^2}{6}\left(\underbrace{pu'(b)-f'(b)}_{u'''(b)}\right)+\frac{h^3}{24}\left(\underbrace{p^2u(b)-pf(b)-f''(b)}_{u^{(4)}(b)}\right)\right]+O\left(h^4\right)=0;$$

$$-d_3\frac{h}{2}\left(py_n-f(b)\right)+p\frac{h^2}{6}\left(\underbrace{d_5-d_4u(b)}_{d_3u'(b)}\right)-d_3\frac{h^2}{6}f'(b)-d_3\frac{h^3}{24}\left(p^2y_n-pf(b)-f''(b)\right)=O\left(h^4\right);$$

$$-d_3\frac{h}{2}\left(py_n-f(b)\right)-p\frac{h^2}{6}\left(d_4y_n-d_5\right)-d_3\frac{h^2}{6}f'(b)-d_3\frac{h^3}{24}\left(p^2y_n-pf(b)-f''(b)\right)=O\left(h^4\right).\tag{13}$$

Таким чином, щоб умова (11) була нульовою рівністю порядку $O\left(h^4\right)$, потрібно від (11) відняти ліву частину виразу (13) порядку $O\left(h^1\right)$, в результаті отримаємо рівняння :

$$d_3\frac{y_n-y_{n-1}}{h}+d_4y_n+$$
$$+d_3\frac{h}{2}\left(py_n-f(b)\right)+p\frac{h^2}{6}\left(d_4y_n-d_5\right)+d_3\frac{h^2}{6}f'(b)+d_3\frac{h^3}{24}\left(p^2y_n-pf(b)-f''(b)\right)=d_5,$$

яке перетворимо до форми

$$A_n y_{n-1}-C_n y_n=-F_n.\tag{14}$$

Виконаємо ланцюжок нескладних перетворень і групувань коефіцієнтів

$$\frac{d_3}{h}y_{n-1}-\frac{d_3}{h}y_n-d_4y_n-$$
$$-d_3\frac{h}{2}\left(py_n-f(b)\right)-p\frac{h^2}{6}\left(d_4y_n-d_5\right)-d_3\frac{h^2}{6}f'(b)-d_3\frac{h^3}{24}\left(p^2y_n-pf(b)-f''(b)\right)=-d_5;$$
$$\frac{d_3}{h}y_{n-1}-\left(\frac{d_3}{h}+d_3p\frac{h}{2}+d_3p^2\frac{h^3}{24}+d_4+d_4p\frac{h^2}{6}\right)y_n=$$
$$=-\left(d_5+d_5p\frac{h^2}{6}+d_3\frac{h}{2}f(b)-d_3\frac{h^2}{6}f'(b)+d_3\frac{h^3}{24}\left(pf(b)+f''(b)\right)\right),$$

отримаємо:

$$A_n=\frac{d_3}{h};\ C_n=\frac{d_3}{h}+d_3p\frac{h}{2}\tilde{p}+d_4\tilde{\tilde{p}};\ F_n=d_3\left(\frac{h}{2}\tilde{p}f(b)+\frac{h^2}{6}\left(\frac{h}{4}f''(b)-f'(b)\right)\right)+d_5\tilde{\tilde{p}}. \tag{15}$$

Отримана система лінійних алгебраїчних рівнянь, яка складається з рівняння (9), системи (4*) і рівняння (14), дає можливість знайти числовий розв'язок (1), (2). Враховуючи, що матриця цієї СЛАР є тридіагональною, а аналіз значень $A_k$, $C_k$, $B_k$ для вхідних даних диференціальної задачі вказує на діагональну перевагу, то можна використати для розв'язання СЛАР метод монотонної правої/лівої прогонки.

## 4. РІЗНИЦЕВІ СХЕМИ ТА ЇХ ПРОГРАМНА РЕАЛІЗАЦІЯ ДЛЯ РІВНЯННЯ ПАРАБОЛІЧНОГО ТИПУ

Отримаємо для крайової задачі

$$\frac{\partial u(x,t)}{\partial t}=K\cdot\frac{\partial^2u(x,t)}{\partial x^2}-p(x,t)\cdot u(x,t)+f(x,t),\ \ (x,t)\in(a,b)\times(0,T], \tag{1}$$
$$K>0,\ \ p(x,t)\ge 0;$$
$$u(a,t)=\mu_1(t),\ \ u(b,t)=\mu_2(t),t\in(0,T]; \tag{2}$$
$$u(x,0)=u0(x),\ \ x\in[a,b] \tag{3}$$

відповідні різницеві задачі (РЗ) і представимо їх у формі, зручній для програмної реалізації з подальшим отриманням як числових, так і графічних 2-D, 3-D візуалізацій розв'язку.

Введемо рівномірні сітки

$$\overline{\omega}_h=\left\{x_k=a+k\cdot h,\ k=\overline{0,n},\ b-a=nh\right\};\ \overline{\omega}_\tau=\left\{\tau_j=j\cdot\tau,\ \ j=\overline{0,m},\ T=m\tau\right\}. \tag{4}$$

На шеститочковому шаблоні $S\{(x_{k-1},t_j),(x_k,t_j),(x_{k+1},t_j),(x_{k-1},t_{j+1}),(x_k,t_{j+1}),(x_{k+1},t_{j+1})\}$ рівнянню (1) поставимо у відповідність однопараметричну сім'ю двошарових схем – *схему з вагою* [18, с. 272-279]

$$y_{t,j}=K\cdot(Y)_{\bar{x}x,k}-pY+\varphi,\ \ k=\overline{1,n-1};\ j=\overline{0,m-1}, \tag{5}$$

де позначено

$$0\le\sigma\le 1;\quad y^{(\sigma)}\equiv Y=\sigma\cdot\hat{y}+(1-\sigma)\cdot y;$$
$$p\equiv\overline{p}_{k,j}=p(x_k,\bar{t}_j),\ \varphi\equiv\overline{\varphi}_{k,j}=f(x_k,\bar{t}_j),\ \bar{t}_j\equiv t_{j+0.5}=t_j+0.5\tau.$$

Крайові і початкові умови апроксимуються точно

$$y_0^j=\mu_1(t_j),\ \ y_n^j=\mu_2(t_j),\ \ j=\overline{1,m}; \tag{6}$$
$$y_k^0=u0(x_k),\ \ k=\overline{0,n}. \tag{7}$$

Від вибору параметра $\sigma$ залежить точність і стійкість схеми (5)–(7). Розглянемо найбільш уживані частинні випадки параметра $\sigma$.

**Явна схема.** При $\sigma = 0$ на шаблоні $S\{(x_{k\mp1}, t_j), (x_k, t_j), (x_k, t_{j+1})\}$ схему (5) запишемо у формі

$$y_k^{j+1} = \gamma \cdot y_{k-1}^j + (1 - 2\gamma - \tau \cdot p) y_k^j + \gamma \cdot y_{k+1}^j + \tau \cdot \varphi, \quad k = \overline{1, n-1}, \tag{8}$$

де $\gamma = K\tau \cdot h^{-2}$.

**Неявна схема.** При $0 < \sigma < 1$ схема (5) матиме вигляд

$$\theta \cdot y_{k-1}^{j+1} - (1 + 2\theta + \tau\sigma \cdot p) y_k^{j+1} + \theta \cdot y_{k+1}^{j+1} = -\tilde{\varphi}, \quad k = \overline{1, n-1}. \tag{9}$$

$$\theta = \sigma \cdot \gamma, \quad \eta = (1 - \sigma) \cdot \gamma, \quad \tilde{\varphi} = w_k^j + \tau \cdot \varphi,$$

$$w_k^j = \eta \cdot y_{k-1}^j + (1 - 2\eta - \tau(1 - \sigma)p) y_k^j + \eta \cdot y_{k+1}^j.$$

**Чисто неявна схема.** При $\sigma = 1$ на шаблоні $S\{(x_k, t_j), (x_{k\mp1}, t_{j+1}), (x_k, t_{j+1})\}$ схему (5) запишемо у формі

$$\gamma \cdot y_{k-1}^{j+1} - (1 + 2\gamma + \tau \cdot p) y_k^{j+1} + \gamma \cdot y_{k+1}^{j+1} = -(y_k^j + \tau \cdot \varphi), \quad k = \overline{1, n-1}. \tag{10}$$

Для програмної реалізації розрахунків за схемами (8)–(10) спочатку обчислюємо початкові значення (7). Потім, циклічно по $j = \overline{0, m-1}$:

**a)** знаходимо значення сіткової функції на границях $x = a,\ x = b$, використовуючи формули (6);

**b)** визначаємо $y_k^{j+1}$ у внутрішніх точках ($k = \overline{1, n-1}$) шару $j+1$, при цьому для явної схеми маємо алгебраїчний розрахунок за формулою (8), а для неявних схем з вагою $0 < \sigma \le 1$ – розв'язання СЛАР (9)|(10) методом монотонної правої/лівої прогонки.

Значення $\vec{\mathbf{y}}^j$, необхідні для виведення (графічної візуалізації), потрібно для відповідних шарів $j = \tilde{J}$ запам'ятовувати в додатковому масиві $\vec{\mathbf{Y}}(\tilde{J})$.

## 5. РІЗНИЦЕВІ СХЕМИ ТА ЇХ ПРОГРАМНА РЕАЛІЗАЦІЯ ДЛЯ РІВНЯННЯ ПУАССОНА

Розглянемо для рівняння Пуассона крайову задачу для випадку прямокутної області $\overline{D} = D \cup \partial D = \{(x_1, x_2) \in [0, a] \times [0, b]\}$:

$$\Delta u(x_1, x_2) = -f(x_1, x_2), \quad x \equiv (x_1, x_2) \in D \tag{1}$$

і функція $u(x_1, x_2)$ на частині границі $\partial D1, \partial D2$ задовольняє умови Діріхле, а на іншій частині $\partial D3, \partial D4$ – умови Неймана:

$$u(a, x_2) = \mu 1(x), \quad x \in \partial D1 = \{(x_1 = a) \,\&\, [0, b]\}; \tag{2}$$

$$u(x_1, b) = \mu 2(x), \quad x \in \partial D2 = \{[0, a] \,\&\, (x_2 = b)\}; \tag{3}$$

$$\frac{\partial u(0, x_2)}{\partial x_1} = \mu 3(x), \quad x \in \partial D3 = \{(x_1 = 0) \,\&\, [0, b]\}; \tag{4}$$

$$\frac{\partial u(x_1, 0)}{\partial x_2} = \mu 4(x), \quad x \in \partial D4 = \{[0, a] \,\&\, (x_2 = 0)\}. \tag{5}$$

Метою цієї роботи є: отримання практичних навичків побудови РЗ з наявністю різнотипових граничних умов, представлення отриманої РЗ у формі, зручній для програмної реалізації з подальшою графічною 3-D візуалізацією розв'язку.

Уведемо в просторовій області $\overline{D}$ прямокутну рівномірну сітку $\overline{\omega}_{\bar{h}}$:

$$\overline{\omega}_{\bar{h}} = \omega_{\bar{h}} \cup \partial\omega_{\bar{h}} = \overline{\omega}_{h1} \times \overline{\omega}_{h2},$$

$$\overline{\omega}_{h1} = \{x_1^{(i)} = i \cdot h1,\ i = \overline{0, n1},\ a = n1 \cdot h1\},$$

$$\overline{\omega}_{h2} = \{x_2^{(j)} = j \cdot h2,\ j = \overline{0, n2},\ b = n2 \cdot h2\}.$$

Рівнянню (1) поставимо у відповідність наступне різницеве рівняння на внутрішніх точках сіткової області ( $x \in \omega_{\bar{h}}$ ):

$$\frac{y_{i-1,j} - 2y_{i,j} + y_{i+1,j}}{h1^2} + \frac{y_{i,j-1} - 2y_{i,j} + y_{i,j+1}}{h2^2} = -f_{i,j},\ i = \overline{1, n1-1},\ j = \overline{1, n2-1}; \qquad (6)$$

де позначено $y_{i,j} = u\left(x_1^{(i)}, x_2^{(j)}\right),\ f_{i,j} = f\left(x_1^{(i)}, x_2^{(j)}\right)$.

Крайові умови (2),(3) апроксимуються точно:

$$y_{n1,j} = \mu 1\left(x_2^{(j)}\right),\quad j = \overline{0, n2}; \qquad (7)$$

$$y_{i,n2} = \mu 2\left(x_1^{(i)}\right),\quad i = \overline{0, n1-1}. \qquad (8)$$

Для отримання апроксимації крайової умови (4) використаємо апроксимацію порядку $O\left(h1^2\right)$ центральною різницевою похідною у вузлі $i$=0, а для виключення законтурної точки – рівняння (6) також у вузлі $i$=0:

$$\frac{y_{1,j} - y_{-1,j}}{2 \cdot h1} = \mu 3\left(x_2^{(j)}\right),$$

$$\frac{y_{-1,j} - 2y_{0,j} + y_{1,j}}{h1^2} + \frac{y_{0,j-1} - 2y_{0,j} + y_{0,j+1}}{h2^2} = -f_{0,j},\ j = \overline{1, n2-1}.$$

Після виключення законтурних точок $y_{-1,j}$, отримаємо

$$\frac{2y_{1,j} - 2y_{0,j} - 2 \cdot h1 \cdot \mu 3_j}{h1^2} + \frac{y_{0,j-1} - 2y_{0,j} + y_{0,j+1}}{h2^2} = -f_{0,j},\ j = \overline{1, n2-1}. \qquad (9)$$

Аналогічним чином отримаємо апроксимацію крайової умови (5) з порядком $O\left(h2^2\right)$:

$$\frac{y_{i-1,0} - 2y_{i,0} + y_{i+1,0}}{h1^2} + \frac{2y_{i,1} - 2y_{i,0} - 2 \cdot h2 \cdot \mu 4_i}{h2^2} = -f_{i,0},\ i = \overline{1, n1-1}.$$

Залишається записати умову для вузла ($i$=0, $j$=0). Використаємо (9) для $j$=0, в яке підставимо $y_{0,-1} = y_{0,1} - 2 \cdot h2 \cdot \mu 4_0$:

$$\frac{2y_{1,0} - 2y_{0,0} - 2 \cdot h1 \cdot \mu 3_0}{h1^2} + \frac{2y_{0,1} - 2y_{0,0} - 2 \cdot h2 \cdot \mu 4_0}{h2^2} = -f_{0,0}.$$

Тепер можна записати різницеву задачу, яка апроксимує вихідну диференціальну (1)–(5) з порядком $O\left(h1^2 + h2^2\right)$, у вигляді, зручному для подальшого застосування ітераційних методів (ІМ):

$$\begin{cases} y_{n1,j} = \mu 1_j,\ j = \overline{0, n2}; \\ y_{i,n2} = \mu 2_i,\ i = \overline{0, n1-1}; \end{cases} \qquad (10)$$

$$\begin{cases} 2\left(h1^{-2} + h2^{-2}\right) y_{0,0} = 2 \cdot h1^{-2} y_{1,0} + 2 \cdot h2^{-2} y_{0,1} + f_{0,0} - 2 \cdot h1^{-1} \mu 3_0 - 2 \cdot h2^{-1} \mu 4_0; \\ 2\left(h1^{-2} + h2^{-2}\right) y_{i,0} = h1^{-2}\left(y_{i-1,0} + y_{i+1,0}\right) + 2 \cdot h2^{-2} y_{i,1} + f_{i,0} - 2 \cdot h2^{-1} \mu 4_i,\ i = \overline{1, n1-1}; \\ 2\left(h1^{-2} + h2^{-2}\right) y_{0,j} = 2 \cdot h1^{-2} y_{1,j} + h2^{-2}\left(y_{0,j-1} + y_{0,j+1}\right) + f_{0,j} - 2 \cdot h1^{-1} \mu 3_j,\ j = \overline{1, n2-1}; \\ 2\left(h1^{-2} + h2^{-2}\right) y_{i,j} = h1^{-2}\left(y_{i-1,j} + y_{i+1,j}\right) + h2^{-2}\left(y_{i,j-1} + y_{i,j+1}\right) + f_{i,j},\ i = \overline{1, n1-1},\ j = \overline{1, n2-1}. \end{cases} \qquad (11)$$

Зауважимо, що рівняння (10) не вимагають застосування ітерацій.

Будемо використовувати неявний ітераційний метод – *метод релаксації* з параметром $\omega = const \in (0,2)$ (для $\omega = 1$ він співпадає з методом Зейделя).

**Алгоритм методу релаксації**

**1.** Обчислюємо деякі допоміжні константи :

$$H = 2\left(h1^{-2} + h2^{-2}\right),\ C = \omega/H,\ A = h1^{-2}\cdot C,\ B = h2^{-2}\cdot C,\ D = 1-\omega;$$
$$A2 = 2A,\ B2 = 2B,\ E1 = 2h1^{-1},\ E2 = 2h2^{-1}.$$

**2.** Для початкового наближення (при $S$ = 0) задаємо деяке значення $y_{i,j}^{(0)} = W_{i,j}$, при $i = \overline{0, n1-1}$, $j = \overline{0, n2-1}$. Значення $W_{i,j}$ можна обирати $W_{i,j}$= 0, або $W_{i,j} = 0.25\cdot f_{i,j}$, або $W_{i,j} = \mu 2_i / b \cdot x_2^{(j)}$. Обчислюємо також граничні значення за формулами (10).

**3.** Циклічно, по $S$ = 0,1,…, виконуємо розрахунки для рівнянь (11) у наступній послідовності :

$$y_{0,0}^{(S+1)} = A2\cdot y_{1,0}^{(S)} + B2\cdot y_{0,1}^{(S)} + C\cdot\left(f_{0,0} - E1\cdot\mu 3_0 - E2\cdot\mu 4_0\right) + D\cdot y_{0,0}^{(S)};$$

$i = \overline{1, n1-1}$:
$$y_{i,0}^{(S+1)} = A\cdot\left(y_{i-1,0}^{(S+1)} + y_{i+1,0}^{(S)}\right) + B2\cdot y_{i,1}^{(S)} + C\cdot\left(f_{i,0} - E2\cdot\mu 4_i\right) + D\cdot y_{i,0}^{(S)};$$

$j = \overline{1, n2-1}$:
$$y_{0,j}^{(S+1)} = A2\cdot y_{1,j}^{(S)} + B\cdot\left(y_{0,j-1}^{(S+1)} + y_{0,j+1}^{(S)}\right) + C\cdot\left(f_{0,j} - E1\cdot\mu 3_j\right) + D\cdot y_{0,j}^{(S)},$$
$$y_{i,j}^{(S+1)} = A\cdot\left(y_{i-1,j}^{(S+1)} + y_{i+1,j}^{(S)}\right) + B\cdot\left(y_{i,j-1}^{(S+1)} + y_{i,j+1}^{(S)}\right) + C\cdot f_{i,j} + D\cdot y_{i,j}^{(S)},\ i = \overline{1, n1-1}.$$

Умовою закінчення циклу по $S$ є виконання для вузлів $i = \overline{0, n1-1}$, $j = \overline{0, n2-1}$ такої нерівності :

$$\left|y_{i,j}^{(S+1)} - y_{i,j}^{(S)}\right| \le \varepsilon .$$

Якщо сіткову функцію $y_{i,j}^{(S)}$ зберігати у вигляді прямокутної матриці **Y**=(Y$_{i,j}$), де $i$ і $j$ – індекси рядка і стовпця, то реалізація цього алгоритму дійсно проста, причому для $y_{i,j}^{(S)}$, $y_{i,j}^{(S+1)}$ в програмі достатньо однієї матриці **Y**. Якщо оперативної пам'яті достатньо, то, враховуючи значну кількість ітерацій, бажано (для зменшення кількості операцій) до початку ітераційного процесу обчислити матрицю **F**, з наступними елементами $F_{i,j}$ :

$$F_{i,j} = \begin{cases} f_{0,0} - E1\cdot\mu 3_0 - E2\cdot\mu 4_0, & (i,\ j) = (0,\ 0); \\ f_{i,0} - E2\cdot\mu 4_i, & (i,\ j) = \left(\overline{1, n1-1},\ 0\right); \\ f_{0,j} - E1\cdot\mu 3_j, & (i,\ j) = \left(0,\ \overline{1, n2-1}\right); \\ f_{i,j}, & (i,\ j) = \left(\overline{1, n1-1},\ \overline{1, n2-1}\right). \end{cases}$$

Для побудови 3-D графіка розв'язку **Y** використовуємо наявні бібліотеки і відповідні вбудовані функції.

## 6. РІЗНИЦЕВІ СХЕМИ ТА ЇХ ПРОГРАМНА РЕАЛІЗАЦІЯ ДЛЯ РІВНЯННЯ ГІПЕРБОЛІЧНОГО ТИПУ

Отримаємо для крайової задачі

$$\frac{\partial^2 u(x,t)}{\partial t^2} = K\cdot\frac{\partial^2 u(x,t)}{\partial x^2} - p(x,t)\cdot u(x,t) + f(x,t),\ (x,t)\in(a,b)\times(0,T], \qquad (1)$$
$$K > 0,\quad p(x,t) \ge 0;$$
$$u(a,t) = \mu_1(t),\ u(b,t) = \mu_2(t),\ t\in(0,T]; \qquad (2)$$
$$u(x,0) = u0(x),\ x\in[a,b], \qquad (3)$$
$$\frac{\partial u(x,0)}{\partial t} = u1(x),\ x\in[a,b] \qquad (4)$$

відповідні РЗ і представимо їх у формі, зручній для програмної реалізації з подальшим отриманням як числових, так і графічних 2-D, 3-D візуалізацій розв'язку.

Введемо рівномірні сітки

$$\overline{\omega}_h = \left\{x_k = a + k\cdot h,\ k = \overline{0, n},\ b - a = nh\right\};\ \overline{\omega}_\tau = \left\{\tau_j = j\cdot\tau,\ j = \overline{0, m},\ T = m\tau\right\}. \qquad (5)$$

На дев'ятиточковому шаблоні $S=S\{(x_{k\mp1},t_{j\mp1}),(x_k,t_{j\mp1}),(x_{k\mp1},t_j),(x_k,t_j)\}$ рівнянню (1) поставимо у відповідність однопараметричну сім'ю тришарових схем – *схему з вагою*

$$y_{\bar{t}t,j}=K\cdot(Y)_{\bar{x}x,k}-pY+\varphi,\quad k=\overline{1,n-1};\ j=\overline{1,m-1}, \tag{6}$$

де позначено

$$0\le\sigma\le0.5;\quad y^{(\sigma)}\equiv Y=\sigma\cdot\hat{y}+(1-2\sigma)y+\sigma\cdot\breve{y};$$
$$p\equiv p_{k,j}=p(x_k,t_j),\quad \varphi\equiv\varphi_{k,j}=f(x_k,t_j).$$

Крайові умови (2) і початкова умова (3) апроксимуються точно

$$y_1^j=\mu_1(t_j),\quad y_N^j=\mu_2(t_j),\quad j=\overline{1,m}; \tag{7}$$

$$y_k^0=u0(x_k),\quad k=0,1,...,n. \tag{8}$$

Для отримання апроксимації умови (4) з другим порядком по $\tau$ використаємо метод законтурних точок:

$$\begin{cases} y_k^1-y_k^{-1}=2\tau\cdot u1(x_k),\ O(\tau^2),\\ y_k^1-2y_k^0+y_k^{-1}=\tau^2\cdot\left(K\cdot y_{\bar{x}x,k}^0-p(x_k,0)y_k^0+f(x_k,0)\right),\ O(\tau^2+h^2).\end{cases}$$

З цієї системи виключаємо $y_k^{-1}$, вводимо позначення $\gamma=\tau/h$ і отримуємо таке рівняння

$$y_k^1=0.5\gamma^2K(y_{k-1}^0+y_{k+1}^0)+(1-\gamma^2K-0.5\tau^2p_{k,0})y_k^0+\tau\cdot u1_k+0.5\tau^2f_{k,0},\quad k=\overline{1,n-1}. \tag{9}$$

Від вибору параметра $\sigma$ залежить точність і стійкість схеми (6)–(8), (9). Розглянемо найбільш уживані частинні випадки параметра $\sigma$.

**<u>Явна схема.</u>** При $\sigma=0$ схема (6) стійка при $\gamma\le1$ і для шарів $j=\overline{1,m-1}$ має вигляд

$$y_k^{j+1}=\gamma^2K(y_{k-1}^j+y_{k+1}^j)+(2-2\gamma^2K-\tau^2p_{k,j})y_k^j-y_k^{j-1}+\tau^2f_{k,j},\quad k=\overline{1,n-1}. \tag{10}$$

**<u>Неявна схема.</u>** Схема (6) при $\sigma\in(0,\frac14)$ умовно стійка ($\gamma\le\sqrt{1-4\sigma}$), а для $\sigma\in[\frac14,\frac12]$ – абсолютно стійка і на шарах $j=\overline{1,m-1}$ має вигляд

$$\sigma\gamma^2K\cdot y_{k-1}^{j+1}-(1+2\sigma\gamma^2K+\sigma\tau^2p_{k,j})y_k^{j+1}+\sigma\gamma^2K\cdot y_{k+1}^{j+1}=-F_k,\quad k=\overline{1,n-1}, \tag{11}$$

де компоненти вектора $\vec{F}$ обчислюються наступним чином

$$F_k=w_k+v_k+\tau^2f_{k,j},$$
$$w_k=\sigma\tau^2K\cdot y_{xx,k}^{j-1}-(1+\sigma\tau^2p_{k,j})y_k^{j-1}=$$
$$=\sigma\gamma^2K(y_{k-1}^{j-1}+y_{k+1}^{j-1})-(1+2\sigma\gamma^2K+\sigma\tau^2p_{k,j})y_k^{j-1},$$
$$v_k=(1-2\sigma)\tau^2K\cdot y_{\bar{x}x,k}^j+(2-(1-2\sigma)\tau^2p_{k,j})y_k^j=$$
$$=(1-2\sigma)\gamma^2K(y_{k-1}^j+y_{k+1}^j)+[2-2(1-2\sigma)\gamma^2K-(1-2\sigma)\tau^2p_{k,j}]y_k^j.$$

Схема програмної реалізації несуттєво відрізняється від схеми для параболічної КЗ. Так, у п. **а)**, крім визначення $y_k^0$, необхідно ще визначити і значення $y_k^1$, використовуючи формули (7) в крайових точках ($k=\{0,n\}$) і формули (9) у внутрішніх точках ($k=\overline{1,n-1}$).

# ІНДИВІДУАЛЬНІ ЗАВДАННЯ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗАННЯ

**№_1. (5 балів)** Розв'язати СЛАР

$$\left(\begin{array}{ccccc|c} 10 & -1 & 1 & 1 & 3 & 1 \\ -1 & 15 & -2 & 0 & 1 & 2 \\ 1 & -2 & 20 & -3 & 1 & -3 \\ 1 & 0 & -3 & 25 & -4 & 3 \\ 3 & 1 & 1 & -4 & 35+K & -1 \end{array}\right)$$

ітераційним методом

1) *Якобі;*
2) *Зейделя;*
3) *верхньої релаксації;; простої ітерації;*
4) *мінімальних нев'язок;*
5) *найшвидшого спуску*

з точністю *eps*=$10^{-7}$. Перевірити знайдений розв'язок точним методом

1) *Гаусса;*
2) *простим LU-методом (факторизація);*
3) *квадратного кореня.*

Варіант індивідуального завдання визначається :

***СЛАР : ітераційний : точний = K : Var(K,6) : Var(K,3),***

де *K* – номер за списком студента в групі.

**№_2. (5 балів)** Знайти з точністю *eps*=$10^{-7}$ наближений розв'язок рівняння

1) $x^4 = 2.23x + 20.47$;
2) $x^4 = 4x^2 - 5.23x + 19.325$;
3) $x^4 = -6.123x + 2.25$;
4) $x^3 = 2.34x^2 - 3.567$;
5) $x^5 = 2.34x^2 - 3.567$;
6) $x^5 = 2.34x^2 - 3x + 2.7$;
7) $x^3 = 4.087x^2 + 20.34$;
8) $x^3 = 2.7x - 1.534$

ітераційним методом

1) *простої ітерації;*
2) *Ейткена-Стеффенсена;*
3) *Ньютона;*
4) *січних;*
5) *Курчатова.*

Варіант індивідуального завдання визначається :

***рівняння : метод = Var(K,8) : Var(K,5).***

**№_3. (10 балів)** Для функції

1) $\sin(x),\ x \in [0, \pi/2]$;
2) $\cos(x),\ x \in [0, \pi/2]$;

*3)* $\sin(x),\ x \in [\pi/2, \pi]$;

*4)* $\cos(x),\ x \in [\pi/2, \pi]$

на рівномірній сітці $\vec{X} = [X_0, X_1, X_2, X_3, X_4]$

**A.** наблизити поліномом, побудованим за формулами:

*1) Лагранжа;*

*2) Ньютона;*

**B.** знайти $u'(X_i)$ на шаблоні, який складається із:

*1) 3-х точок;*

*2) 4-х точок;*

*3) 5-х точок;*

**C.** обчислити визначений інтеграл за методом:

*1) трапецій;*

*2) Сімпсона.*

Варіант індивідуального завдання визначається :

***функція : n.A : n.B : n.C = Var(K,4) : Var(K,2) : Var(K,3) : Var(K,2).***

**№_4. (10 балів)** Розв'язати задачу Коші

$$\begin{cases} u'(x) = \sin(x - u(x)), \\ u(0) = K \end{cases}$$

на відрізку [0,1] з кроком *h*=0.01 *методом Рунге-Кутта* другого порядку точності (без автоматичного вибору кроку). Побудувати графік розв'язку. Використати значення параметра

*1)* $\gamma$ = 0.25;

*2)* $\gamma$ = 0.5;

*3)* $\gamma$ = 0.75;

*4)* $\gamma$ = 1.

Варіант індивідуального завдання визначається :

***задача : параметр = K : Var(K,4).***

**№_5. (10 балів)** Різницевим методом другого порядку точності знайти наближений розв'язок рівняння

$$\frac{d^2u(x)}{dx^2} - K \cdot x \cdot u(x) = -K - x^3 \sin(x),\ x \in (0,1)$$

при наступних крайових умовах :

*1)* $u(0) = 1,\ u'(1) = 0$;

*2)* $u'(0) + 2u(0) = 0,\ u(1) = 1$;

*3)* $u'(0) - u(0) = 0,\ u'(1) = 0$;

*4)* $u'(0) = 0,\ u(1) = 2$;

*5)* $u(0) = 1,\ u'(1) - u(1) = 0$;

*6)* $u'(0) = 1,\ u'(1) = 2$;

*7)* $u'(0) - u(0) = 0,\ u'(1) - u(1) = 0$;

*8)* $u'(0) = 1,\ u'(1) + u(1) = 1$;

*9)* $u'(0) - 2u(0) = 1,\ u'(1) - 2u(1) = 2$;

*10)* $u(0) = 2,\ u'(1) + u(1) = 2$.

Побудувати графік розв’язку з кроком $h$=0.05.
Варіант індивідуального завдання визначається :

***рівняння : кр.умови = K : Var(K,10).***

**№_6. (15 балів)** Побудувати різницеву задачу з порядком апроксимації $O(h+\tau)$ на шаблоні $S\{(x_i,t_j),(x_{i-1},t_{j+1}),(x_i,t_{j+1})\}$ для мішаної крайової задачі :

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial t}+\dfrac{1}{K}\dfrac{\partial u(x,t)}{\partial x}=f(x,t), \\ (x,t)\in(0,1]\times(0,2]; \\ u(x,0)=u0(x), \quad x\in[0,1]; \\ u(0,t)=g(t), \quad t\in(0,2]. \end{cases}$$

Чисельно розв’язати ($h$=0.01, τ=0.01) для наступних даних: $f(x,t)=\sin(x+t)-t$; $u0(x)=x(1-x^K)$; $g(t)=5t$. Побудувати графіки наближеного розв'язку для $t$={0,1,2}.
***K*** – номер варіанта індивідуального завдання.

**№_7. (15 балів)** За допомогою

1) *явної σ = 0;*
2) *неявної σ = 0.25;*
3) *неявної σ = 0.5;*
4) *неявної σ = 0.75;*
5) *неявної σ = 1*

різницевої схеми знайти наближений розв’язок мішаної крайової задачі :

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial t}=K\dfrac{\partial^2 u(x,t)}{\partial x^2}-p(x,t)u(x,t)+f(x,t), \\ (x,t)\in(0,1]\times(0,2]; \\ u(x,0)=u0(x), \quad x\in[0,1]; \\ u(0,t)=g_1(t),\ u(1,t)=g_2(t), \quad t\in(0,2] \end{cases}$$

для наступних даних: $f(x,t)=\sin(x+2t)-t$; $p(x,t)=2x+t$; $u0(x)=x(1-x^K)$; $g_1(t)=2t$; $g_2(t)=0.5t$. Побудувати графіки наближеного розв'язку для $t$={0,1,2}.
Варіант індивідуального завдання визначається :

***схема : задача = Var(K,5) : K.***

**№_8. (15 балів)** За допомогою різницевої схеми знайти наближений розв’язок крайової задачі :

$$\begin{cases} \dfrac{\partial^2 u(x,y)}{\partial x^2}+\dfrac{\partial^2 u(x,y)}{\partial y^2}=-f(x,y), \\ (x,y)\in(0,1)\times(0,2); \\ u(1,y)=g_1(y), \quad y\in[0,2]; \\ u(x,2)=g_2(x), \quad x\in[0,1]; \\ u(0,y)=g_3(y), \quad y\in[0,2]; \\ u(x,0)=g_4(x), \quad x\in[0,1] \end{cases}$$

при наступних даних:

$$f(x,y)=K\cdot(x+y),$$

$$g_1(y)=0;\ g_2(x)=0;\ g_3(y)=4-y^2;\ g_4(x)=4(1-x^2).$$

Побудувати графік наближеного розв'язку.

*K* – номер варіанта індивідуального завдання.

**№_9. (15 балів)** За допомогою різницевої схеми знайти наближений розв’язок крайової задачі :

$$\begin{cases} \dfrac{\partial^2 u(x,t)}{\partial t^2}=K\dfrac{\partial^2 u(x,t)}{\partial x^2}-p(x,t)u(x,t)+f(x,t), \\ \quad (x,t)\in(0,1]\times(0,2]; \\ u(0,t)=g_1(t),\quad u(1,t)=g_2(t),\qquad t\in(0,2]; \\ u(x,0)=u0(x),\quad \dfrac{\partial u(x,0)}{\partial t}=u1(x),\quad x\in[0,1] \end{cases}$$

при наступних даних:

$$f(x,t)=\sin(x+t)-t;\ p(x,t)=K\cdot x+0.5t;$$
$$u0(x)=x(1-x);\ u1(x)=2-x;\ g_1(t)=2t;\ g_2(t)=t.$$

Побудувати графік наближеного розв'язку $u(x,t)$.

***K*** – номер варіанта індивідуального завдання.

*Зауваження 1*. Функція ***Var(K,p)*** – циклічність по ***K*** з періодом ***p***, визначається наступним чином :

$$Var(K,p)=\begin{cases} i, & i\neq 0, \\ p, & i=0. \end{cases}\quad \text{де } i=K\%p.$$

*Зауваження 2*. Для написання програмного коду числового розв’язання поставлених задач використати алгоритмічну мову Python [12, 13], а також створене програмне забезпечення, яке представлено в наступному розділі і оформлено у вигляді відповідних модулів. Крім модулів, у цій частині посібника розміщено приклади використання функцій із наведених модулів для розв’язання тестових задач.

# ОПИС МОДУЛІВ ДЛЯ РЕАЛІЗАЦІЇ ЧИСЛОВИХ МЕТОДІВ

## 0. Загальні сервісні модулі.

Модуль ***util_nle*** містить набір допоміжних функцій для операцій друку систем лінійних алгебраїчних рівнянь, одновимірних масивів та графіків.

Модуль ***util_poly*** містить набір допоміжних функцій для операцій (обчислення значення в точці, сума/різниця, множення, ділення, похідна, друк) роботи з поліномами виду $P_N(x) = p_0 x^{N-1} + ... + p_{N-1}x + p_N$ ( $p_0 \neq 0$ виключення для нуль-полінома $P_0(x) = 0$), який моделюємо списком $[p_0, p_1, ..., p_{N-1}, p_N]$.

**Програмний код Python функцій модуля *util_nle* :**

```
""" Модуль допоміжних програм для СЛАР і побудови 2-D графіків."""

import numpy as np
import matplotlib.pyplot as plt

def r_or_c(n):
    '''Перетворення n в (Re(n), Im(n)) для n є complex.

    '''
    if isinstance(n, complex):
        return (n.real, n.imag)
    else :
        return n

def put_A_b(A, b, form):
    '''Друк СЛАР у вигляді розширеної матриці (A b).

    '''
    n =len(b)
    print("         С Л А Р :")
    for i in range(n):
        for j in range(n):
            print(form % r_or_c(A[i, j]), end = '')
        print(form % r_or_c(b[i]))

def put_answer(M, A, b, err, x, form, r):
    '''Друк результатів.

    '''
    n =len(b)
    print("\nРозв'язок %s :" % M)
    if err == 0 :
        for i in range(n):
            print(form % r_or_c(x[i]), end = '')
            if (i+1) % r == 0 : print()
        print("\nНев'язка  :")
        B = np.dot(A, x) - b
        for i in range(n):
            print(form % r_or_c(B[i]), end = '')
            if (i+1) % r == 0 : print()
        if n % r != 0 : print()
    elif err == 1 :
```

```
        print("Матриця не квадратна")
    elif err == 2 :
        print("розм. b не відповідає розм. A")
    else :
        print("Матриця вироджена")

def put_A3_b(a, c, b, f, n, form):
    '''Друк тридіагональної СЛАР у вигляді розширеної матриці.

       Тридіагональна СЛАР з розширеною матрицею :
            (a -c b -f)
    '''
    A = np.zeros((n,n))
    for i in range(n):
        if i > 0 : A[i,i-1] = a[i]
        A[i,i] = -c[i]
        if i < n-1 : A[i,i+1] = b[i]
    print("\n          С Л А Р :")
    for i in range(n):
        for j in range(n):
            print(form % r_or_c(A[i, j]), end = '')
        print(form % r_or_c(-f[i]))
    return A

def put_arr(M, X, form, r):
    '''Друк вектора.

    '''
    n = len(X)
    print("%s" % M)
    for i in range(n):
        print(form % r_or_c(X[i]), end = '')
        if (i+1) % r == 0 : print()
    print()

def movespinesticks():
    '''Перемістити осі у нульову позицію

    '''
    ax = plt.gca()  #отримати поточний об'єкт класу axes
    # зробити праву та верхню осі невидимими:
    ax.spines['top'].set_color('none')
    ax.spines['right'].set_color('none')
    # перенести нижню вісь у позицію y=0:
    ax.xaxis.set_ticks_position('bottom')
    ax.spines['bottom'].set_position(('data',0))
    # перенести ліву вісь у позицію x == 0:
    ax.yaxis.set_ticks_position('left')
    ax.spines['left'].set_position(('data',0))
```

**Програмний код Python функцій модуля *util_poly* :**

```
""" Модуль допоміжних програм для роботи з поліномами.

    Поліном Pn(x)=P0*x^n+...+Pn представляємо списком
    P = [P0,P1,...,Pn], де P0<>0 (аналогічно СКМ MatLab)
"""
```

```
from copy import deepcopy
import numpy as np


def delzeroes(P):
    '''Видаляє з Pn(x) нульові коефіцієнти.

       Якщо всі коефіцієнти =0, то це 0-поліном 0*x**0.
    '''
    while len(P) > 0:
        if P[0] != 0: break
        P = np.array(P[1:])
    if len(P) == 0: P = np.array([0])  # якщо всі нулі
    return P


def poly_val(P, t):
    '''Обчислення значення z = Pn(t) за схемою Горнера.

    '''
    n = len(P);
    z = P[0]
    for i in range(1, n):
        z = P[i] + t * z
    return z


def put_poly(M, P):
    '''Друк Pn(x).

    '''
    n = len(P)
    s = ''
    f = "%g"
    for i in range(n):
        if P[i] != 0:
            j = n - i - 1
            if j == 0:
                st = f % P[i]
            elif j == 1:
                st = (f + "*x") % P[i]
            else:
                st = (f + "*x^%i") % (P[i], j)
            s = s + st
        elif i == 0:
            s = s + f % P[i]
        f = "%+g"
    print(M + ":= " + s)


def add_sub(P, op, Q):
    '''R = (P op Q) - сума|різниця поліномів P,Q.

    '''
    nP = len(P)
    nQ = len(Q)
    if nP >= nQ:
        j = nP - nQ
```

```
        R = P.copy()
        R[j:] = R[j:] + (Q if op == "+" else -Q)
    else:
        j = nQ - nP
        R = Q.copy() if op == "+" else -Q.copy()
        R[j:] = P + R[j:]
    return delzeroes(R)


def conv(P, Q):
    '''R = P * Q - добуток поліномів P,Q.
    '''
    nP = len(P)
    nQ = len(Q)
    R = np.zeros(nP + nQ - 1)
    for k1 in range(nP):
        for k2 in range(nQ):
            R[k1 + k2] += P[k1] * Q[k2]
    return delzeroes(R)


def deconv(P, Q):
    '''H - частка, R - остача ділення полінома P на Q (P=H*Q+R).

    '''
    R = P.copy()
    H = np.zeros(1)  # P=H*Q+R
    kQ = len(Q) - 1
    cQ = Q[0]
    if kQ == 0 and cQ == 0:
        err = 1  # Ділення на 0-поліном
    else:
        err = 0
        kR = len(R) - 1
        i = kR - kQ
        while i >= 0:
            o = np.zeros(i + 1)
            o[0] = R[0] / cQ
            R = add_sub(R, "-", conv(Q, o))
            H = add_sub(H, "+", o)
            kR = len(R) - 1
            i = kR - kQ
    return err, H, R


def derive(P, n=1):
    '''n-та похідна полінома P (за угодою - перша).

    '''
    nP = len(P)
    if nP == 1:
        return np.zeros(1)
    else:
        R = P.copy()
        nR = nP
        for i in range(n):
            m = nR - 1
            for k in range(nR):
```

```
                R[k] = (m - k) * R[k]
            R = np.array(R[0:m])
            nR -= 1
        return delzeroes(R)


if __name__ == "__main__":
    P = np.array([0, 5, 1, 0, 3, -2])
    P = delzeroes(P)
    put_poly("P(%i,x)" % (len(P) - 1), P)

    t = 2
    print("P(%i,x=%g)=%g" % (len(P) - 1, t, poly_val(P, t)))

    Q = np.array([-5, 0, 3, -3, 2])
    put_poly("Q(%i,x)" % (len(Q) - 1), Q)

    PpQ = add_sub(P, "+", Q)
    put_poly("P+Q=PpQ(%i,x)" % (len(PpQ) - 1), PpQ)

    PmQ = add_sub(P, "-", Q)
    put_poly("P-Q=PmQ(%i,x)" % (len(PmQ) - 1), PmQ)

    MP = conv(P, PmQ)
    put_poly("P*PmQ=MP(%i,x)" % (len(MP) - 1), MP)

    err, P_, Q_ = deconv(MP, PmQ)
    if err == 0:
        put_poly("MP/PmQ=P_(%i,x)" % (len(P_) - 1), P_)
        put_poly("Q_(%i,x)" % (len(Q_) - 1), Q_)

    Q_ = derive(P)
    put_poly("Q_=(P(%i,x))'" % (len(P) - 1), Q_)
    Q_ = derive(P, 3)
    put_poly("Q_=(P(%i,x))'''" % (len(P) - 1), Q_)
```

## I. Модуль точних методів розв'язання СЛАР Ax=b.

**В модуль *slae_em* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *gauss(A,b)* | розв'язок СЛАР Ax=b методом Гаусса |
| *lu_simple(A,b)* | розв'язок СЛАР A*x=b простим LU-методом (факторизація) |
| *sqrt_r(A,b)* | розв'язок СЛАР A*x=b, (A=A') методом квадратних коренів |
| *sqrt_c(A,b)* | розв'язок СЛАР A*x=b, (A=A*) методом квадратних коренів |
| *progm(a,c,b,f)* | розв'язок СЛАР з тридіагональною матрицею (монотонна права прогонка) |
| *progn(a,c,b,f)* | розв'язок СЛАР з тридіагональною матрицею (немонотонна прогонка) |

**Програмний код Python функцій модуля :**

```
""" Модуль точних методів розв'язання СЛАР Ax=b."""

from copy import deepcopy
import numpy as np
from math import sqrt


def gauss(A, b):
    '''Розв'язання СЛАР Ax=b методом Гаусса.

       метод Гаусса з вибором оптимального елемента по стовпцю.
       Вхідні параметри:
       A    - квадратна матриця СЛАР;
       b    - вектор правої частини СЛАР;
       Вихідні параметри:
       x    - розв'язок СЛАР;
       err  - код закінчення:=
              0 - знайдено розв'язок;
              1 - матриця не квадратна;
              2 - розм. b не відповідає розм. A;
              3 - матриця вироджена.
    '''
    n, m = A.shape
    k, = b.shape
    if n != m:
        x = None
        err = 1
    elif m != k:
        x = None
        err = 2
    else:
        # прямий хід м.Гаусса :
        k = 0
        while k < n:
            a = 0
            ik = k
            i = k
            while i < n:  # пошук max-елемента по стовпцю k
                w = np.abs(A[i, k])
                if a < w:
                    a = w
```

```
                    ik = i
                i += 1
            if a == 0:
                x = None
                err = 3
            else:
                if ik != k:  # провідний елемент в ik-му рядку
                    # обмін елементів k і ik рядків
                    j = k
                    a = b[k]
                    b[k] = b[ik]
                    b[ik] = a
                    while j < n:
                        a = A[k, j]
                        A[k, j] = A[ik, j]
                        A[ik, j] = a
                        j += 1
                # ділення на провідний елемент
                j = k
                while j < n:
                    if j == k:
                        a = A[k, j]
                        A[k, j] = 1
                        b[k] = b[k] / a
                    else:
                        A[k, j] = A[k, j] / a
                    j += 1
                # виключення x[k] з рядків k<j<n
                i = k + 1
                while i < n:
                    a = A[i, k]
                    A[i, k] = 0
                    b[i] -= a * b[k]
                    j = k + 1
                    while j < n:
                        A[i, j] -= a * A[k, j]
                        j += 1
                    i += 1
            k += 1
        # зворотний хід м.Гаусса :
        err = 0
        x = np.empty(n)
        k = n - 1
        x[k] = b[k]
        while k > 0:
            j = k
            k -= 1
            x[k] = b[k] - np.vdot(A[k, j:n], x[j:n])
    return x, err


def lu_simple(A, b):
    '''Розв'язання СЛАР A*x=b простим LU-методом (факторизація).

       Вхідні  дані:
        A   - матриця (всі головні мінори <>0 !), A=L*U;
        b   - відомий вектор (права частина);
       Вихідні дані:
```

```
    L,U - факторизовані матриці (розміщені на місці A)
          L= (a[i,j]), i>=j;
          U= {1, i=j; (a[i,j]), i<j);
    x   - розв'язок;
    err - код закінчення:=
           0 - розв'язок знайдено;
          >0 - розв'язок не знайдено
                (ділення на 0 на етапі факторизації A=L*R);
          <0 - розв'язок не знайдено
                (ділення на 0 на етапі розв'язання L*x=b).
'''
n = b.size
# виконання факторизації A
a = A[0, 0]
if a == 0:
    x = None
    err = 1
    return None, None, x, err
A[0, 1:n] /= a
for m in range(1, n):
    for i in range(m, n):
        s = 0
        for k in range(0, m):
            s += A[i, k] * A[k, m]
        A[i, m] -= s
    a = A[m, m]
    if a == 0:
        x = None
        err = m
        return None, None, x, err
    for j in range(m + 1, n):
        s = A[m, j]
        for k in range(0, m):
            s -= A[m, k] * A[k, j]
        A[m, j] = s / a
# Результат факторизації L*U=A
L = np.zeros((n, n))
U = np.zeros((n, n))
for i in range(n):
    for j in range(n):
        if i > j:
            L[i, j] = A[i, j]
        elif i == j:
            L[i, j] = A[i, j]
            U[i, j] = 1
        else:
            U[i, j] = A[i, j]
# розв'язання L*X=b
x = deepcopy(b)
x[0] /= A[0, 0]
for i in range(1, n):
    a = A[i, i]
    if a == 0:
        x = None
        err = -i
        return A, x, err
    s = x[i] - np.vdot(A[i, 0:i], x[0:i])
    x[i] = s / a
```

```
    # розв'язання R*x=X
    for i in range(n - 2, -1, -1):
        x[i] -= np.vdot(A[i, i + 1:n], x[i + 1:n])
    err = 0
    return L, U, x, err


def sqrt_r(A, b):
    '''Розв'язання СЛАР  A*x=b, (A=A') методом квадратних коренів.

       Вхідні  дані:
        A   - симетрична додатньо визначена матриця,
              яка зберігається в вигляді вектора
              (верхня трикутна матриця, записана по рядкам);
        b   - відомий вектор (права частина СЛАР);
       Вихідні дані:
        x   - розв'язок;
        err - код закінчення:=
              1 - розв'язок не знайдено (A<=0 | det(A)=0);
              0 - розв'язок знайдено.
    '''
    n = len(b)
    x = np.ones(n)
    m = 0
    r = n
    p = 0
    for i in range(0, n):
        if m == 0:
            if A[m] == 0:
                err = 1
                print('SQRT_R: обрив процесу, A<=0 !')
                return x, err
        for j in range(0, n - i):
            s1 = 0
            s2 = 0
            t = i
            for k in range(0, i):
                e = A[t] * x[k]
                s1 += e * A[t + j]
                s2 += e * b[k]
                t += n - k - 1
            mj = m + j
            A[mj] -= s1
            if j == 0:
                if abs(A[m]) == 0:
                    err = 1
                    print('SQRT_R: обрив процесу, A<=0 !')
                    return x, err
                if A[m] < 0:
                    x[i] = -1
                    p = 1
                A[m] = sqrt(A[m] * x[i])
            else:
                A[mj] = A[mj] / A[m] * x[i]
        b[i] = (b[i] - s2) / A[m] * x[i]
        m += r
        r -= 1
    m = len(A) - 1
```

```
    r = 1
    for i in range(n - 1, -1, -1):
        s2 = 0
        for k in range(1, n - i):
            s2 += b[i + k] * A[m + k]
        b[i] = (b[i] - s2) / A[m]
        r += 1
        m -= r
    if p == 1: print('SQRT_R: A<0 !')
    err = 0
    x = deepcopy(b)
    return x, err


def sqrt_c(A, b):
    '''Розв'язання СЛАР  A*x=b, (A=A*) методом квадратних коренів.

       Вхідні  дані:
        A   - ермітова матриця;
        b   - відомий вектор (права частина СЛАР);
       Вихідні дані:
        x   - розв'язок;
        err - код закінчення:=
              1 - при перевірці вхідних даних знайдена помилка і
                виконання функції аварійно закінчено
              0 - перевірка вхідних даних пройшла успішно.
    '''
    n = len(b)
    x = np.zeros(n, dtype=np.complex128)
    if np.any(np.any(A != A.T.conj())):  # чи є матриця ермітовою ?
        err = 1
        print('SQRT_C: матриця A не ермітова !')
    else:
        err = 0
        d = np.ones(n, "int")
        # формування матриці S (на місці матриці A) і вектора d=diag(D)
        for i in range(0, n):
            s = A[i, i].real
            for k in range(0, i):
                s -= d[k] * abs(A[k, i]) ** 2
            if s < 0:
                d[i] = -1
                s = -s
            z = sqrt(s)
            A[i, i] = z
            z *= d[i]
            for j in range(i + 1, n):
                s = A[i, j]
                for k in range(0, i):
                    s -= d[k] * A[k, i].conjugate() * A[k, j]
                A[i, j] = s / z

        #  розв'язування СЛАР   S'*y = b
        y = np.zeros(n, dtype=np.complex128)
        for i in range(0, n):
            s = b[i]
            for k in range(0, i):
                s -= A[k, i].conjugate() * y[k]
```

```
            y[i] = s / A[i, i]

        #  розв'язування СЛАР   (D*S)*x=y
        for i in range(n - 1, -1, -1):
            s = 0
            for j in range(i + 1, n):
                s += A[i, j] * x[j]
            x[i] = (y[i] / d[i] - s) / A[i, i]

    return x, err


def progm(a, c, b, f):
    '''Монотонна права прогонка для СЛАР з тридіагональною матрицею.

       Метод монотонної прогонки для СЛАР,
       яка записана у формі :
       a(k)*x(k-1) - c(k)*x(k) + b(k)*x(k+1) = - f(k),
              k = 0,1,...,N;  a(0) = b(N) = 0,
       при виконанні умов :
          |c(k)| >= |a(k)| + |b(k)|, k=0,...,N;
            a(k) <> 0,               k=1,...,N;
            b(k) <> 0,               k=0,...,N-1;
       Вхідні  дані:
        a - піддіагональ;
       -c - діагональ матриці;
        b - наддіагональ;
       -f - права частина.
       Вихідні дані:
        x - розв'язок.
    '''
    n = c.size
    N = n - 1
    x = np.zeros(n)
    alfa = np.zeros(n)
    beta = np.zeros(n)
    # пряма прогонка :
    k = 0
    while k < N:
        j = k + 1
        z = c[k] - a[k] * alfa[k]
        alfa[j] = b[k] / z
        beta[j] = (f[k] + a[k] * beta[k]) / z
        k = j
    # зворотна прогонка :
    x[N] = (f[N] + a[N] * beta[N]) / (c[N] - a[N] * alfa[N])
    k = N
    while k > 0:
        j = k - 1
        x[j] = alfa[k] * x[k] + beta[k]
        k = j
    return x


def progn(a, c, b, f):
    '''Немонотонна прогонка для СЛАР з тридіагональною матрицею.

       Метод немонотонної прогонки для СЛАР,
       яка записана у формі :
```

```
    a(k)*x(k-1) - c(k)*x(k) + b(k)*x(k+1) = - f(k),
        k = 0,1,...,N;  a(0) = b(N) = 0,
   Вхідні  дані:
    a - піддіагональ;
   -c - діагональ матриці;
    b - наддіагональ;
   -f - права частина.
   Вихідні дані:
    x - розв'язок.
'''

n = c.size
N = n - 1
NN = N - 1
x = np.zeros(n)
alfa = np.zeros(n)
beta = np.zeros(n)
ka = np.zeros(n, 'int')
ka[1] = 1
te = np.zeros(n, 'int')
# пряма прогонка :
C = c[0]
A = a[1]
F = f[0]
Q = f[1]
k = 0
while k < N:
    j = k + 1
    if abs(C) >= abs(b[k]):
        alfa[j] = b[k] / C
        beta[j] = F / C
        C = c[j] - A * alfa[j]
        F = Q + A * beta[j]
        te[j] = ka[k]
        ka[j] = j
        if k != NN:
            j += 1
            A = a[j]
            Q = f[j]
    else:
        alfa[j] = C / b[k]
        beta[j] = -F / b[k]
        C = c[j] * alfa[j] - A
        F = Q - c[j] * beta[j]
        te[j] = j
        ka[j] = ka[k]
        if k != NN:
            j1 = j
            j += 1
            A = a[j] * alfa[j1]
            Q = f[j] + a[j] * beta[j1]
    k += 1
# зворотна прогонка :
x[ka[N]] = F / C
j = N
k = NN
```

```
    while k >= 0:
        x[te[j]] = alfa[j] * x[ka[j]] + beta[j]
        j = k
        k -= 1
    return x
```

**Програмний код Python головного модуля *main_slae_em* для тестування *slae_em*:**

```
from util_lne import put_A_b, put_answer, put_A3_b
from slae_em import *

Ap = np.array([[3.0, 1, -1, 2],
               [-5, 1, 3, -4],
               [2, 0, 1, -1],
               [1, -5, 3, -3]])
bp = np.array([6.0, -12, 1, 3])
put_A_b(Ap, bp, " %12.6f")

AA = deepcopy(Ap);
bb = deepcopy(bp)
x = np.linalg.solve(AA, bb)
put_answer("NumPy-солвер для СЛАР", Ap, bp, 0, x, " %12.6f", 5)

AA = deepcopy(Ap);
bb = deepcopy(bp)
x, err = gauss(AA, bb)
put_answer("м.Гаусса", Ap, bp, err, x, " %12.6f", 5)

AA = deepcopy(Ap);
bb = deepcopy(bp)
L, U, x, err = lu_simple(AA, bb)
put_answer("м.факторизації", Ap, bp, err, x, " %12.6f", 5)
if err == 0:
    print("L =\n", L);
    print("U =\n", U)
    B = np.dot(L, U);
    print("Перевірка L*U=A:=\n", B, "\n")

A = np.array([[13.0, 1, -2, 2],
              [1, 11, 3, -4],
              [-2, 3, 14, -1],
              [2, -4, -1, 16]])
b = np.array([14.0, -16, 20, 52])
n = b.size
put_A_b(A, b, " %12.6f")

k = (n + 1) * n // 2
AA = np.zeros(k)
bb = deepcopy(b)
k = -1
for i in range(n):
    for j in range(i, n):
        k += 1
        AA[k] = A[i, j]
x, err = sqrt_r(AA, bb)
```

```
put_answer("м.квадратних коренів для A=A'",
           A, b, err, x, " %12.6f", 5)
print("Порівняйте отриману матрицю L\n",
      "(факторизація A=L'*L)\n", AA)
L = np.linalg.cholesky(A)
c = L.T.conj()
print("і матрицю\n", c)
print("ПЕРЕВІРКА:  A=L'*L\n", np.dot(L, c), "\n")

AA = np.array([[13.0, 1j, -2],
               [-1j, 11, 3j],
               [-2, -3j, 14]], dtype=np.complex128)
y = np.array([1 - 1j, -1, 2j], dtype=np.complex128)
bb = np.dot(AA, y)
Ao = deepcopy(AA)
x, err = sqrt_c(AA, bb)
put_A_b(Ao, bb, " %8.4f%+.4fj")
put_answer("м.квадратних коренів для A=A*",
           Ao, bb, 0, x, " %8.4f%+.4fj", 5)

a = np.array([0, 1, 2, -3.0])
c = np.array([10, 5, 8, 4.0])
b = np.array([1, 2, -3, 0.0])
f = np.array([10, 1, -2, 5.0])
n = f.size
bb = deepcopy(-f)

AA = put_A3_b(a, c, b, f, n, " %12.6f")
x = progm(a, c, b, f)
put_answer("м.монотонної прогонки", AA, -f, 0, x, " %12.6f", 5)

a = np.ones(12)
a[0] = 0
a[11] = 0
c = np.ones(12)
b = np.ones(12)
b[0] = 0
b[11] = 0
f = np.zeros(12)
f[0] = 1
n = f.size

AA = put_A3_b(a, c, b, f, n, " %3d")
x = progn(a, c, b, f)
put_answer("м.немонотонної прогонки", AA, -f, 0, x, " %3d", 20)

x = np.linalg.lstsq(Ap, bp, rcond=-1)
x = x[0]
put_answer("м.мінімізації функціонала Φ(x)=‖A*x-b‖^2",
           Ap, bp, 0, x, " %12.6f", 5)
```

**Результат виконання *`main_slae_em`*:**

```
    СЛАР:
   3.000000   1.000000  -1.000000   2.000000   6.000000
  -5.000000   1.000000   3.000000  -4.000000 -12.000000
   2.000000   0.000000   1.000000  -1.000000   1.000000
   1.000000  -5.000000   3.000000  -3.000000   3.000000
```

```
Розв'язок NumPy-солвер для СЛАР :
    1.000000   -1.000000    2.000000    3.000000
Нев'язка  :
   -0.000000    0.000000    0.000000    0.000000
Розв'язок м.Гаусса :
    1.000000   -1.000000    2.000000    3.000000
Нев'язка  :
    0.000000    0.000000   -0.000000   -0.000000
Розв'язок м.факторизації :
    1.000000   -1.000000    2.000000    3.000000
Нев'язка  :
    0.000000    0.000000    0.000000   -0.000000
L =
 [[ 3.         0.         0.         0.        ]
 [-5.         2.66666667 0.         0.        ]
 [ 2.        -0.66666667 2.         0.        ]
 [ 1.        -5.33333333 6.         2.5       ]]
U =
 [[ 1.         0.33333333 -0.33333333  0.66666667]
 [ 0.         1.         0.5        -0.25      ]
 [ 0.         0.         1.         -1.25      ]
 [ 0.         0.         0.         1.        ]]
Перевірка L*U=A:=
 [[ 3.  1. -1.  2.]
 [-5.  1.  3. -4.]
 [ 2.  0.  1. -1.]
 [ 1. -5.  3. -3.]]

       С Л А Р :
   13.000000    1.000000   -2.000000    2.000000   14.000000
    1.000000   11.000000    3.000000   -4.000000  -16.000000
   -2.000000    3.000000   14.000000   -1.000000   20.000000
    2.000000   -4.000000   -1.000000   16.000000   52.000000
Розв'язок м.квадратних коренів для A=A' :
    1.000000   -1.000000    2.000000    3.000000
Нев'язка  :
    0.000000    0.000000    0.000000   -0.000000
Порівняйте отриману матрицю L  (факторизація A=L'*L)
 [ 3.60555128  0.2773501  -0.5547002   0.5547002   3.30500786  0.95426283
 -1.25683397  3.57514897  0.14182409  3.75400612]
і матрицю
 [[ 3.60555128  0.2773501  -0.5547002   0.5547002 ]
 [ 0.         3.30500786  0.95426283 -1.25683397]
 [ 0.         0.         3.57514897  0.14182409]
 [ 0.         0.         0.         3.75400612]]
ПЕРЕВІРКА:  A=L'*L
 [[ 13.   1.  -2.   2.]
 [  1.  11.   3.  -4.]
 [ -2.   3.  14.  -1.]
 [  2.  -4.  -1.  16.]]

       С Л А Р :
  13.0000+0.0000j   0.0000+1.0000j  -2.0000+0.0000j  13.0000-18.0000j
  -0.0000-1.0000j  11.0000+0.0000j   0.0000+3.0000j -18.0000-1.0000j
  -2.0000+0.0000j  -0.0000-3.0000j  14.0000+0.0000j  -2.0000+33.0000j
Розв'язок м.квадратних коренів для A=A* :
   1.0000-1.0000j  -1.0000+0.0000j   0.0000+2.0000j
Нев'язка  :
   0.0000-0.0000j   0.0000+0.0000j   0.0000+0.0000j

       С Л А Р :
   -10.000000    1.000000    0.000000    0.000000  -10.000000
     1.000000   -5.000000    2.000000    0.000000   -1.000000
     0.000000    2.000000   -8.000000   -3.000000    2.000000
     0.000000    0.000000   -3.000000   -4.000000   -5.000000
Розв'язок м.монотонної прогонки :
    1.000000    0.000000   -1.000000    2.000000
Нев'язка  :
    0.000000    0.000000    0.000000    0.000000
```

С Л А Р :

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 |
| 1 | -1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | -1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | -1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | -1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | -1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | -1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | -1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 |

Розв'язок м.немонотонної прогонки :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | -1 | -1 | 0 | 1 | 1 | 0 | -1 | -1 | 0 |

Нев'язка :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Розв'язок м.мінімізації функціонала Ф(x)=‖A*x-b‖^2 :

| | | | |
|---|---|---|---|
| 1.000000 | -1.000000 | 2.000000 | 3.000000 |

Нев'язка :

| | | | |
|---|---|---|---|
| -0.000000 | 0.000000 | -0.000000 | -0.000000 |

## II. Модуль ітераційних методів розв'язання СЛАР Ax=b.

**В модуль *slae_im.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *jacobi(A,b,x, eps=1e-6,maxit=1000)* | розв'язок СЛАР Ax=b ітераційним методом Якобі |
| *jac_rel(A,b,x,w, eps=1e-6,maxit=1000)* | розв'язок СЛАР A*x=b узагальненим методомЯкобі - релаксація |
| *zejdel(A,b,x, eps=1e-6,maxit=1000)* | розв'язок СЛАР A*x=b ітераційним методом Зейделя |
| *relax(A,b,x,w, eps=1e-6,maxit=1000)* | розв'язок СЛАР A*x=b ітераційним методом релаксації |
| *msit(A,b,x,t, eps=1e-6,maxit=1000)* | розв'язок СЛАР A*x=b методом простої ітерації |
| *mmn(A,b,x, eps=1e-6,maxit=1000)* | розв'язок СЛАР A*x=b методом мінімальних нев'язок |
| *mns(A,b,x, eps=1e-6,maxit=1000)* | розв'язок СЛАР A*x=b методом найшвидшого спуску |

**Програмний код Python функцій модуля :**

```
""" Модуль ітераційних методів розв'язання СЛАР Ax=b."""

from copy import deepcopy
import numpy as np


def jacobi(A, b, x, eps=1e-6, maxit=1000):
    '''Розв'язання СЛАР Ax=b ітераційним методом Якобі.

       Вхідні параметри:
       A    - квадратна матриця СЛАР;
       b    - вектор правої частини СЛАР;
       x    - початкове наближення до розв'язку;
       eps  - точність;
       maxit- максимальна кількість ітерацій.
       Вихідні параметри:
       x    - наближений розв'язок СЛАР;
       k    - кількість виконаних ітерацій.
    '''
    n, m = A.shape
    if n != m:
        x = None
        k = None
    else:
        #  Підготовка необхідних масивів для ІП
        C = np.zeros_like(A)
        B = deepcopy(b)
        xx = np.empty(n)
        i = 0
        while i < n:
            a = A[i, i]
            B[i] = B[i] / a
            j = 0
            while j < n:
                if i != j: C[i, j] = A[i, j] / a
```

```
                j += 1
            i += 1
        #  Реалізація ІП
        k = 0
        while k <= maxit:
            m = True
            i = 0
            while i < n:
                a = B[i] - np.vdot(C[i, :], x)
                if m:
                    m = abs(x[i] - a) < eps
                xx[i] = a
                i += 1
            x = deepcopy(xx)
            k += 1
            if m: break
    return x, k


def jac_rel(A, b, x, w, eps=1e-6, maxit=1000):
    '''Розв'язання СЛАР Ax=b узагальненим меодом Якобі - релаксації.

       Вхідні параметри:
       A    - квадратна матриця СЛАР;
       b    - вектор правої частини СЛАР;
       x    - початкове наближення до розв'язку;
       eps  - точність;
       maxit- максимальна кількість ітерацій.
       Вихідні параметри:
       x    - наближений розв'язок СЛАР;
       k    - кількість виконаних ітерацій.
    '''
    n, m = A.shape
    if n != m:
        x = None
        k = None
    else:
        if w == None: w = 2 / 3  # параметр процесу
        #  Підготовка необхідних масивів для ІП
        D = w / np.diag(A)
        #  Реалізація ІП
        k = 0
        F = False
        while not F and k < maxit:
            k += 1
            d = D * (np.dot(A, x) - b)
            F = np.linalg.norm(d, np.inf) < eps
            x = x - d

    return x, k


def zejdel(A, b, x, eps=1e-6, maxit=1000):
    '''Розв'язання СЛАР Ax=b ітераційним методом Зейделя.

       Вхідні параметри:
       A    - квадратна матриця СЛАР;
       b    - вектор правої частини СЛАР;
```

```
        x     - початкове наближення до розв'язку;
        eps   - точність;
        maxit- максимальна кількість ітерацій.
        Вихідні параметри:
        x     - наближений розв'язок СЛАР;
        k     - кількість виконаних ітерацій.
    '''
    n, m = A.shape
    if n != m:
        x = None
        k = None
    else:
        #  Підготовка необхідних масивів для ІП
        C = np.zeros_like(A)
        B = deepcopy(b)
        i = 0
        while i < n:
            a = A[i, i]
            B[i] = B[i] / a
            j = 0
            while j < n:
                if i != j: C[i, j] = A[i, j] / a
                j += 1
            i += 1
        #  Реалізація ІП
        k = 0
        while k <= maxit:
            m = True
            i = 0
            while i < n:
                a = B[i] - np.vdot(C[i, :], x)
                if m:
                    m = abs(x[i] - a) < eps
                x[i] = a
                i += 1
            k += 1
            if m: break
    return x, k


def relax(A, b, x, w, eps=1e-6, maxit=1000):
    '''Розв'язання СЛАР Ax=b ітераційним методом релаксації.

        Вхідні параметри:
        A     - квадратна матриця СЛАР;
        b     - вектор правої частини СЛАР;
        x     - початкове наближення до розв'язку;
        w     - параметр процесу;
        eps   - точність;
        maxit- максимальна кількість ітерацій.
        Вихідні параметри:
        x     - наближений розв'язок СЛАР;
        k     - кількість виконаних ітерацій.
    '''
    n, m = A.shape
    if n != m:
        x = None
        k = None
```

```
    else:
        #  Підготовка необхідних масивів для ІП
        ww = w - 1
        C = np.zeros_like(A)
        B = deepcopy(b)
        i = 0
        while i < n:
            a = w / A[i, i]
            B[i] = B[i] * a
            j = 0
            while j < n:
                C[i, j] = ww if i == j else a * A[i, j]
                j += 1
            i += 1
        #  Реалізація ІП
        k = 0
        while k <= maxit:
            m = True
            i = 0
            while i < n:
                a = B[i] - np.vdot(C[i, :], x)
                if m:
                    m = abs(x[i] - a) < eps
                x[i] = a
                i += 1
            k += 1
            if m: break
    return x, k


def msit(A, b, x, t, eps=1e-6, maxit=1000):
    '''Розв'язання СЛАР Ax=b методом простої ітерації.

       Вхідні параметри:
       A     - квадратна матриця СЛАР;
       b     - вектор правої частини СЛАР;
       x     - початкове наближення до розв'язку;
       t     - параметр (для значення None,
               програмне визначення t<t0, t0=2/‖A‖);
       eps   - точність;
       maxit- максимальна кількість ітерацій.
       Вихідні параметри:
       x     - наближений розв'язок СЛАР;
       k     - кількість виконаних ітерацій.
    '''
    n, m = A.shape
    if n != m:
        x = None
        k = None
    else:
        #  Реалізація ІП
        if t == None:
            t = 2 / np.linalg.norm(A) - eps
        k = 0
        while k <= maxit:
            y = t * (np.dot(A, x) - b)
            x = x - y
            k += 1
```

```
            if np.linalg.norm(y, np.inf) < eps: break
    return x, k


def mmn(A, b, x, eps=1e-6, maxit=1000):
    '''Розв'язання СЛАР Ax=b методом мінімальних нев'язок.

       Вхідні параметри:
       A     - квадратна матриця СЛАР;
       b     - вектор правої частини СЛАР;
       x     - початкове наближення до розв'язку;
       eps   - точність;
       maxit- максимальна кількість ітерацій.
       Вихідні параметри:
       x     - наближений розв'язок СЛАР;
       k     - кількість виконаних ітерацій.
    '''
    n, m = A.shape
    if n != m:
        x = None
        k = None
    else:
        #  Реалізація ІП
        k = 0
        while k <= maxit:
            r = np.dot(A, x) - b
            y = np.dot(A, r)
            t = np.vdot(y, r) / np.vdot(y, y)
            y = t * r
            x = x - y
            k += 1
            if np.linalg.norm(y, np.inf) < eps: break
    return x, k


def mns(A, b, x, eps=1e-6, maxit=1000):
    '''Розв'язання СЛАР Ax=b методом найшвидшого спуску.

       Вхідні параметри:
       A     - квадратна матриця СЛАР;
       b     - вектор правої частини СЛАР;
       x     - початкове наближення до розв'язку;
       eps   - точність;
       maxit- максимальна кількість ітерацій.
       Вихідні параметри:
       x     - наближений розв'язок СЛАР;
       k     - кількість виконаних ітерацій.
    '''
    n, m = A.shape
    if n != m:
        x = None
        k = None
    else:
        #  Реалізація ІП
        k = 0
        while k <= maxit:
            r = np.dot(A, x) - b
            y = np.dot(A, r)
```

```
            t = np.vdot(r, r) / np.vdot(y, r)
            y = t * r
            x = x - y
            k += 1
            if np.linalg.norm(y, np.inf) < eps: break
    return x, k
```

**Програмний код Python головного модуля *main_slae_im* для тестування *slae_im*:**

```
from util_lne import put_A_b, put_answer
from slae_im import *

A = np.array([[10.0, -1, 1, 1, 3],
              [-1, 15, -2, 0, 1],
              [1, -2, 20, -3, 1],
              [1, 0, -3, 25, -4],
              [3, 1, 1, -4, 30]])
b = np.array([1.0, 2, -3, 2, -1])
put_A_b(A, b, " %10.6f")

AA = deepcopy(A)
bb = deepcopy(b)
x = np.linalg.solve(AA, bb)
put_answer("NumPy-солвер для СЛАР", A, b, 0, x, " %11.8f", 6)

eps = 1e-9

x = deepcopy(b)
x, it = jacobi(A, b, x, eps)
if it != None:
    S = " \n    точність=%12.7e ітерацій=%i" % (eps, it)
    put_answer("м.Якобі" + S, A, b, 0, x, " %11.8f", 6)

x = deepcopy(b)
x, it = jac_rel(A, b, x, 0.9818, eps)  # 181818
if it != None:
    S = " \n    точність=%12.7e ітерацій=%i" % (eps, it)
    put_answer("узагальненим м.Якобі - релаксація" + S,
               A, b, 0, x, " %11.8f", 6)

x = deepcopy(b)
x, it = zejdel(A, b, x, eps)
if it != None:
    S = " \n    точність=%12.7e ітерацій=%i" % (eps, it)
    put_answer("м.Зейделя" + S, A, b, 0, x, " %11.8f", 6)

x = deepcopy(b)
x, it = relax(A, b, x, 1.0025, eps)
if it != None:
    S = " \n    точність=%12.7e ітерацій=%i" % (eps, it)
    put_answer("м.верхньої релаксації" + S,
               A, b, 0, x, " %11.8f", 6)

x = deepcopy(b)
x, it = msit(A, b, x, None, eps)
if it != None:
```

```
    S = " \n    точність=%12.7e ітерацій=%i" % (eps, it)
    put_answer("м.простої ітерації" + S,
               A, b, 0, x, " %11.8f", 6)

w = np.linalg.eigvals(A)
t = 2 / (max(w) + min(w))
print("Оптимальний параметр=", t)
x = deepcopy(b)
x, it = msit(A, b, x, t, eps)
if it != None:
    S = " \n    точність=%12.7e ітерацій=%i" % (eps, it)
    put_answer("м.оптимальний лінійний" + S,
               A, b, 0, x, " %11.8f", 6)

x = deepcopy(b)
x, it = mmn(A, b, x, eps)
if it != None:
    S = " \n    точність=%12.7e ітерацій=%i" % (eps, it)
    put_answer("ММН" + S, A, b, 0, x, " %11.8f", 6)

x = deepcopy(b)
x, it = mns(A, b, x, eps)
if it != None:
    S = " \n    точність=%12.7e ітерацій=%i" % (eps, it)
    put_answer("МНС" + S, A, b, 0, x, " %11.8f", 6)
```

### Результат виконання ***main_slae_im***:

```
    СЛАР:
 10.000000 -1.000000  1.000000  1.000000  3.000000  1.000000
 -1.000000 15.000000 -2.000000  0.000000  1.000000  2.000000
  1.000000 -2.000000 20.000000 -3.000000  1.000000 -3.000000
  1.000000  0.000000 -3.000000 25.000000 -4.000000  2.000000
  3.000000  1.000000  1.000000 -4.000000 30.000000 -1.000000
Розв'язок NumPy-солвер для СЛАР :
 0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
 0.00000000 0.00000000 -0.00000000 0.00000000 0.00000000

Розв'язок м.Якобі
   точність=1.0000000e-09 ітерацій=19 :
 0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
 0.00000000 -0.00000000 0.00000000 -0.00000000 0.00000000

Розв'язок узагальненим м.Якобі - релаксація
   точність=1.0000000e-09 ітерацій=18 :
 0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
 0.00000000 0.00000000 -0.00000000 -0.00000000 -0.00000001

Розв'язок м.Зейделя
   точність=1.0000000e-09 ітерацій=11 :
 0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
-0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
Розв'язок м.верхньої релаксації
   точність=1.0000000e-09 ітерацій=10 :
 0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
-0.00000000 -0.00000000 0.00000000 0.00000000 -0.00000000
```

Розв'язок м.простої ітерації
точність=1.0000000e-09 ітерацій=44 :
0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
0.00000001 0.00000000 -0.00000000 -0.00000000 -0.00000000

Оптимальний параметр= 0.047618835368
Розв'язок м.оптимальний лінійний
точність=1.0000000e-09 ітерацій=40 :
0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
0.00000000 0.00000000 -0.00000000 0.00000000 -0.00000001

Розв'язок ММН
точність=1.0000000e-09 ітерацій=37 :
0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
0.00000001 0.00000000 0.00000000 -0.00000000 0.00000000

Розв'язок МНС
точність=1.0000000e-09 ітерацій=37 :
0.13269252 0.12692164 -0.13412880 0.05229448 -0.03938975
Нев'язка :
0.00000001 0.00000000 0.00000000 -0.00000001 0.00000001

## III. Модуль ітераційних методів розв'язання нелінійних рівнянь і систем.

**В модуль *nle_im.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *search_region_root(f,inp=1)* | графічний пошук області існування кореня рівняння f(x)=0 |
| *sit_1(x,f,eps=1e-8,mki=50)* | знаходження кореня рівняння x=f(x) методом простої ітерації |
| *e_stef_1(x,f,eps=1e-8,mki=50)* | знаходження кореня рівняння x=f(x) методом Ейткена-Стеффенсена |
| *njut_1(x,f,df,eps=1e-8,mki=50)* | знаходження кореня рівняння f(x)=0 ітераційним методом Ньютона |
| *tangent_1(x0,x1,f,eps=1e-8,mki=50)* | знаходження кореня рівняння f(x)=0 ітераційним методом січних |
| *kyrchat_1(x0,x1,f,eps=1e-8,mki=50)* | знаходження кореня рівняння f(x)=0 ітераційним методом Курчатова |
| *njut_n(x,f,df,eps=1e-8,mki=50)* | розв'язок системи рівнянь F(X)=0 ітераційним методом Ньютона |
| *grad_n(x,f,df,eps=1e-8,mki=50)* | розв'язок системи рівнянь F(X)=0 ітераційним градієнтним методом |

**Програмний код Python функцій модуля :**

```
""" Модуль ітераційних методів розв'язання нелінійних рівнянь і
систем."""

import matplotlib.pyplot as plt
import numpy as np
from re import sub
from util_lne import movespinesticks


def search_region_root(f, inp=1):
    '''Графічний пошук області існування кореня рівняння f(x)=0.

       Вхідні параметри:
       f    - ім'я векторизованої функції f(x);
       inp  - спосіб завдання початкового наближення до кореня(ів):
              0 - користувач в діалоговому режимі;
              1 - програмне.
       Вихідні параметри:
       x0   - масив наближень до кореня(ів).
    '''
    search = 1
    while search:
        while True:
            s = input('Введіть [a,b] - діапазон пошуку коренів \n' +
                      'і n - кількість точок розбиття (a<b &
n>0)\n=').strip()
            s = sub(r"[\[\],;]+", r" ", s)
            s = s.split()
            a = float(s[0])
            b = float(s[1])
            n = int(s[2])
            if a < b and n > 0: break
```

```
            # побудова графіка в діапазоні x є [a,b]
            x = np.linspace(a, b, n)  # сітка по x
            y1 = f(x)
            y2 = np.zeros(n)
            plt.plot(x, y1, 'b', x, y2, 'k')
            movespinesticks()
            plt.xlabel('x')
            plt.ylabel('y')
            plt.show()
            s = '? продовжимо дослідження областей з коренями (1-так|0-ні) ='
            search = int(input(s))

    if inp == 0:
            s = input('? Наближення до кореня(ів)=').strip()
            s = sub(r"[,;]+", r" ", s)
            s = s.split()
            x0 = [float(a) for a in s]
    else:
            idx = np.argwhere(np.diff(np.sign(y1 - y2))).flatten()
            x0 = x[idx]
    print()
    return x0


def sit_1(x, f, eps=1e-8, mki=50):
    '''Знаходження кореня рівняння x=f(x) методом простої ітерації.

       Вхідні параметри:
       x    - початкове наближення до кореня;
       f    - функція f(x);
       eps  - точність;
       mki  - максимальна кількість ітерацій.
       Вихідні параметри:
       x    - фінішне наближення до кореня;
       err  - код закінчення:
              0 - знайдено корінь з точністю eps;
              1 - eps не належить (0,1);
              2 - за mki ітерацій не знайдено корінь з точністю eps;
       k    - кількість виконаних ітерацій.
    '''
    k = 0
    if eps <= 0 or eps >= 1:
            err = 1
            return x, err, k
    F = True
    while F and k < mki:
            k += 1
            xx = x
            x = f(x)
            F = abs(x - xx) > eps
    err = 0 if k < mki else 2
    return x, err, k


def e_stef_1(x, f, eps=1e-8, mki=50):
    '''Знаходження кореня рівняння x=f(x) методом Ейткена-Стеффенсена.

       Вхідні параметри:
       x    - початкове наближення до кореня;
```

```
        f    - функція f(x);
        eps  - точність;
        mki  - максимальна кількість ітерацій.
        Вихідні параметри:
        x    - фінішне наближення до кореня;
        err  - код закінчення:
               0 - знайдено корінь з точністю eps;
               1 - eps не належить (0,1);
               2 - за mki ітерацій не знайдено корінь з точністю eps;
        k    - кількість виконаних ітерацій.
    '''
    k = 0
    if eps <= 0 or eps >= 1:
        err = 1
        return x, err, k
    F = True
    while F and k < mki:
        k += 1
        x0 = x
        x1 = f(x0)
        x2 = f(x1)
        o = x2 - x1
        o = o * o / (x0 - 2 * x1 + x2)
        x = x2 - o
        F = abs(o) > eps
    err = 0 if k < mki else 2
    return x, err, k


def njut_1(x, f, df, eps=1e-8, mki=50):
    '''Знаходження кореня рівняння f(x)=0 ітераційним методом Ньютона.

        Вхідні параметри:
        x    - початкове наближення до кореня;
        f    - функція f(x);
        df   - функція f'(x);
        eps  - точність;
        mki  - максимальна кількість ітерацій.
        Вихідні параметри:
        x    - фінішне наближення до кореня;
        err  - код закінчення:
               0 - знайдено корінь з точністю eps;
               1 - eps не належить (0,1);
               2 - за mki ітерацій не знайдено корінь з точністю eps;
        k    - кількість виконаних ітерацій.
    '''
    k = 0
    if eps <= 0 or eps >= 1:
        err = 1
        return x, err, k
    F = True
    while F and k < mki:
        k += 1
        o = f(x) / df(x)
        x -= o
        F = abs(o) > eps
    err = 0 if k < mki else 2
    return x, err, k
```

```
def tangent_1(x0, x1, f, eps=1e-8, mki=50):
    '''Знаходження кореня рівняння f(x)=0 ітераційним методом січних.

       Вхідні параметри:
       x0   - початкове наближення до кореня;
       x1   - наступне наближення до кореня;
       f    - функція f(x);
       eps  - точність;
       mki  - максимальна кількість ітерацій.
       Вихідні параметри:
       x    - фінішне наближення до кореня;
       err  - код закінчення:
              0 - знайдено корінь з точністю eps;
              1 - eps не належить (0,1);
              2 - за mki ітерацій не знайдено корінь з точністю eps;
       k    - кількість виконаних ітерацій.
    '''
    k = 0
    if eps <= 0 or eps >= 1:
        err = 1
        return x0, err, k
    F = True
    f1 = f(x0)
    while F and k < mki:
        k += 1
        f0 = f1
        f1 = f(x1)
        o = (x1 - x0) / (f1 - f0) * f1
        x = x1 - o
        F = abs(o) > eps
        x0 = x1
        x1 = x
    err = 0 if k < mki else 2
    return x, err, k


def kyrchat_1(x0, x1, f, eps=1e-8, mki=50):
    '''Знаходження кореня рівняння f(x)=0 ітераційним методом Курчатова.

       Вхідні параметри:
       x0   - початкове наближення до кореня;
       x1   - наступне наближення до кореня;
       f    - функція f(x);
       eps  - точність;
       mki  - максимальна кількість ітерацій.
       Вихідні параметри:
       x    - фінішне наближення до кореня;
       err  - код закінчення:
              0 - знайдено корінь з точністю eps;
              1 - eps не належить (0,1);
              2 - за mki ітерацій не знайдено корінь з точністю eps;
       k    - кількість виконаних ітерацій.
    '''
    k = 0
    if eps <= 0 or eps >= 1:
        err = 1
```

```
            return x, err, k
        F = True
        f1 = f(x0)
        while F and k < mki:
            k += 1
            f0 = f1
            f1 = f(x1)
            o = 2 * (x1 - x0) * f1 / (f(2 * x1 - x0) - f0)
            x = x1 - o
            F = abs(o) > eps
            x0 = x1
            x1 = x
        err = 0 if k < mki else 2
        return x, err, k


def njut_n(x, f, df, eps=1e-8, mki=50):
    '''Розв'язання системи рівнянь F(X)=0 ітераційним методом Ньютона.

       Вхідні параметри:
       x    - початкове наближення до розв'язку;
       f    - вектор-функція F(x);
       df   - матриця Якобі (dFi(x)/dXj), i,j=1,2,...,n;
       eps  - точність;
       mki  - максимальна кількість ітерацій.
       Вихідні параметри:
       x    - фінішне наближення до розв'язку;
       err  - код закінчення:
              0 - знайдено розв'язок з точністю eps;
              1 - eps не належить (0,1);
              2 - за mki ітерацій не знайдено розв'язок з точністю eps;
       k    - кількість виконаних ітерацій.
    '''
    k = 0
    if eps <= 0 or eps >= 1:
        err = 1
        return x, err, k
    F = True
    while F and k < mki:
        k += 1
        o = f(x)
        J = df(x)
        o = np.linalg.solve(J, o)
        x = x - o
        F = np.linalg.norm(o) > eps
    err = 0 if k < mki else 2
    return x, err, k


def grad_n(x, f, df, eps=1e-8, mki=50):
    '''Розв'язання системи рівнянь F(X)=0 ітераційним градієнтним методом.

   Вхідні параметри:
       x    - початкове наближення до розв'язку;
       f    - вектор-функція F(x);
       df   - матриця Якобі (dFi(x)/dXj), i,j=1,2,...,n;
       eps  - точність;
       mki  - максимальна кількість ітерацій.
```

```
        Вихідні параметри:
        x    - фінішне наближення до розв'язку;
        err  - код закінчення:
               0 - знайдено розв'язок з точністю eps;
               1 - eps не належить (0,1);
               2 - за mki ітерацій не знайдено розв'язок з точністю eps;
        k    - кількість виконаних ітерацій.
    '''
    k = 0
    if eps <= 0 or eps >= 1:
        err = 1
        return x, err, k
    F = True
    while F and k < mki:
        k += 1
        z = f(x)
        J = df(x)
        d = np.dot(J.T.conj(), z)
        v = np.dot(J, d)
        m = np.vdot(z, v) / np.vdot(v, v)
        d = m * d
        x = x - d
        F = np.linalg.norm(d) > eps
    err = 0 if k < mki else 2
    return x, err, k
```

**Програмний код Python головного модуля *main_nle_im* для тестування *nle_im*:**

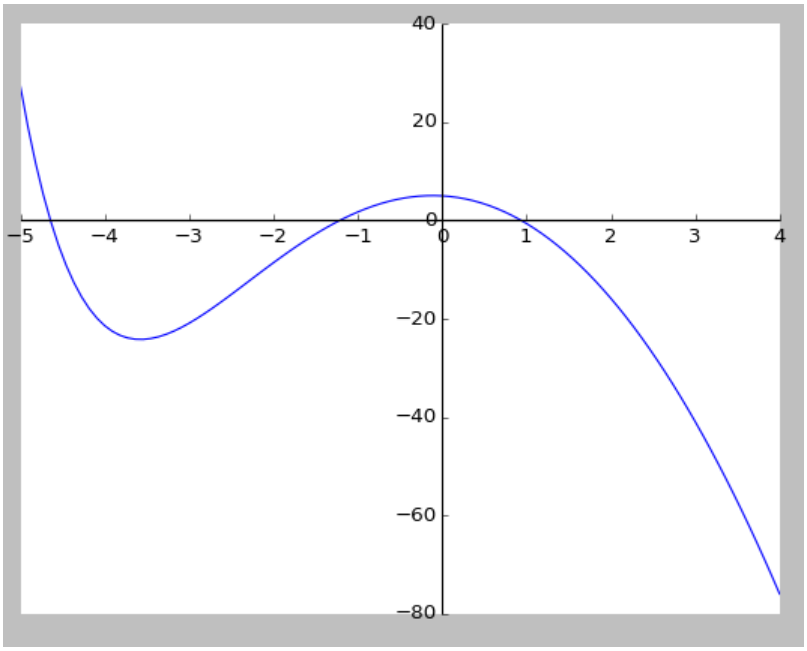
```
from nle_im import *
from util_lne import put_arr
from math import exp, sqrt

def f(x):
    return exp(-x) - 5*x*x + 4

def df(x):
    return -exp(-x) - 10*x

def s(x):
    return sqrt(0.8 + 0.2*exp(-x))

def SF(X):
    F = np.array([X[0]*(1+X[0])-2*X[1]*X[2]-0.1,
                  X[1]*(1-X[1])+3*X[0]*X[2]+0.2,
                  X[2]*(1+X[2])+2*X[0]*X[1]-0.3])
    return F

def dSF(X):
    W = np.array([[1+2*X[0],  -2*X[2],  -2*X[1]],
                  [  3*X[2], 1-2*X[1],   3*X[0]],
                  [  2*X[1],   2*X[0], 1+2*X[2]]])
    return W

F = np.vectorize(f)
eps = 1e-9
```

```
R = search_region_root(F,0)  # введення наближення користувачем
print("Наближення до кореня(ів):\n", R)
for x in R :
    x, err, it = sit_1(x, s, eps)
    if err != 1:
        st = "м.простої ітерації: x=%14.8e eps=%12.7e іт=%i"
        print(st % (x, eps, it))

for x in R :
    x, err, it = e_stef_1(x, s, eps)
    if err != 1:
        st = "м.Ейткена-Стеффенсена: x=%14.8e eps=%12.7e іт=%i"
        print(st % (x, eps, it))

R = search_region_root(F)  # введення наближення програмно
print("Наближення до кореня(ів):\n", R)
print()
for x in R :
    x, err, it = njut_1(x, f, df, eps)
    if err != 1:
        st = "м.Ньютона: x=%14.8e eps=%12.7e іт=%i"
        print(st % (x, eps, it))

delta = 1e-3
print()
for x0 in R :
    x1 = x0 + delta
    x, err, it = tangent_1(x0, x1, f, eps)
    if err != 1:
        st = "м.січних: x=%14.8e eps=%12.7e іт=%i"
        print(st % (x, eps, it))

print()
for x0 in R :
    x1 = x0 + delta
    x, err, it = kyrchat_1(x0, x1, f, eps)
    if err != 1:
        st = "м.Курчатова: x=%14.8e eps=%12.7e іт=%i"
        print(st % (x, eps, it))

X = np.zeros(3)
print("\n  Система нелінійних рівнянь")
X, err, it = njut_n(X, SF, dSF, eps)
if err != 1:
    print("\nм.Ньютона: eps=%12.7e іт=%i" % (eps, it))
    put_arr("  X =", X, "%17.9e", 5)
    put_arr("F(X)=", SF(X), "%17.9e", 5)

X = np.zeros(3)
X, err, it = grad_n(X, SF, dSF, eps)
if err != 1:
    print("\nГрадієнтний метод: eps=%12.7e іт=%i" % (eps, it))
    put_arr("  X =", X, "%17.9e", 5)
    put_arr("F(X)=", SF(X), "%17.9e", 5)
```

**Результат виконання *`main_nle_im`*:**

*Зауваження*. Введення інформації в діалоговому режимі позначено «жирним».

Введіть [a,b] - діапазон пошуку коренів
і n - кількість точок розбиття (a<b & n>0)
**=-5,4,100**




? продовжимо дослідження областей з коренями (1-так|0-ні) =**0**
? Наближення до кореня(ів)=**1**

Наближення до кореня(ів):
[1.0]
м.простої ітерації: x=9.37200402e-01 eps=1.0000000e-09 іт=7
м.Ейткена-Стеффенсена: x=9.37200402e-01 eps=1.0000000e-09 іт=3
Введіть [a,b] - діапазон пошуку коренів
і n - кількість точок розбиття (a<b & n>0)
=**-5 4 100**
? продовжимо дослідження областей з коренями (1-так|0-ні) =**0**

Наближення до кореня(ів):
[-4.72727273 -1.27272727 0.90909091]

м.Ньютона: x=-4.64181118e+00 eps=1.0000000e-09 іт=4
м.Ньютона: x=-1.21376579e+00 eps=1.0000000e-09 іт=4
м.Ньютона: x=9.37200402e-01 eps=1.0000000e-09 іт=4

м.січних: x=-4.64181118e+00 eps=1.0000000e-09 іт=5
м.січних: x=-1.21376579e+00 eps=1.0000000e-09 іт=5
м.січних: x=9.37200402e-01 eps=1.0000000e-09 іт=4

м.Курчатова: x=-4.64181118e+00 eps=1.0000000e-09 іт=5
м.Курчатова: x=-1.21376579e+00 eps=1.0000000e-09 іт=4
м.Курчатова: x=9.37200402e-01 eps=1.0000000e-09 іт=4

Система нелінійних рівнянь

м.Ньютона: eps=1.0000000e-09 іт=6
X =
1.282414583e-02 -1.778006680e-01 2.446880443e-01
F(X)=
0.000000000e+00 2.775557562e-17 0.000000000e+00

Градієнтний метод: eps=1.0000000e-09 іт=16
X =
1.282414596e-02 -1.778006681e-01 2.446880444e-01
F(X)=
2.369952012e-10 -1.155703866e-10 3.645556079e-11

## IV. Модуль чисельних методів інтерполяції.

**В модуль *interp.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *exist_coincidence(X)* | чи є однакові (кратні) елементи сітки X ? |
| *find_Di(X,n,t,ri=0,find=0)* | пошук відрізку D(i)=[X(i),X(i+1)],i=0,...,n-2, що містить t |
| *ipL(X,y)* | побудова інтерполяційного полінома у формі Лагранжа |
| *ipL_val(X,y,T)* | обчислення в точках T інтерполяційного полінома у формі Лагранжа |
| *ipN(X,y)* | побудова інтерполяційного полінома у формі Ньютона |
| *ipN_val(X,y,T)* | обчислення в точках T інтерполяційного полінома у формі Ньютона |
| *spl01_val(X,y,T,find)* | обчислення в точках T сплайна нульового степеня |
| *spl11_val(X,y,T,find)* | обчислення в точках T сплайна першого степеня |
| *spl11(X,y)* | побудова сплайна першого степеня |
| *spline11(X,S11,T,find)* | обчислення в точках T сплайна першого степеня |

**Програмний код Python функцій модуля :**

```
""" Модуль чисельних методів інтерполяції."""

from util_poly import *

def exist_coincidence(X):
    '''Чи є співпадаючі (кратні) елементи сітки X ?

    '''
    P = list(X); P.sort()
    n = len(X); c = False; i = 0
    while not c and  i < n-1 :
        j = i;  i += 1; c = P[j]==P[i]
    return c

def is_order(X):
    '''Чи впорядковані і без кратності елементи сітки X ?

    '''
    n = len(X); c = True; i = 0
    while c and i < n-1 :
        j = i;  i += 1; c = c and X[j] < X[i]
    return c

def find_Di(X, n, t, ri = 0, find = 0):
    '''Пошук відрізку D(i)=[X(i),X(i+1)],i=0,...,n-2, що містить t.

       Вхідні параметри:
       X    - вектор вузлів інтерполяції
              ( попарно різні і у порядку зростання
                a=X(0)<X(1)<...<X(n-2)<X(n-1)=b );
       n    - кількість вузлів інтерполяції;
       t    - значення для пошуку t є [a,b];
       ri   - при 0, перебір відрізків D(i), i=0,...,n-2;
              при 1, перебір відрізків D(i), i=0,...,n-2
                              і додатково перевірка t=X(n-1);
       find - алгоритм пошуку: 0= послідовний, 1= бінарний.
```

```
        Вихідні параметри:
        i     - номер відрізка D(i), на якому знаходиться t (при ri=0),
                або i=n-1 (при ri=1).
    '''
    n1 = n - 1
    if t == X[0] : return 0
    elif ri and t == X[n1] : return n1

    if find : # бінарний
        i = 0; r = n1
        while i < r :
            k = (i+r) // 2
            if X[k] < t :
                i = k
            else :
                r = k
            if i+1 == r : break
    else :    # послідовний
        k = 0; F = False
        while k < n1 and not F :
            i = k; k += 1; F = t < X[k]
    return i

def ipL(X, y):
    '''Побудова інтерполяційного поліному у формі Лагранжа.

        Вхідні параметри:
        X     - вектор вузлів інтерполяції;
        y     - вектор значень функції у вузлах сітки.
        Вихідні параметри:
        err   - код закінчення:
                0 - перевірка вхідних даних пройшла успішно;
                1 - при перевірці вхідних даних знайдена помилка і
                    виконання функції аварійно закінчено;
        L     - вектор, який є представленням інтерп.поліному Lm(x)
    '''
    if exist_coincidence(X) :
        err = 1;
        print('ipL: Є кратні вузли інтерполювання !')
        return err, None

    n = len(X); err = 0
    # завдання Lm(x)=0 і обчислення поліномів Wi(x)= x-X(i), i=0,...,n-1
    W = []; L = np.zeros(n)
    for i in range(n) :
        W.append(np.array([1, -X[i]]))
    # побудова інтерполяційного поліному Lm(t)
    for i in range(n) :
        c = y[i]; e = np.ones(1)
        for j in range(n) :
            if j != i :
                c /= (X[i] - X[j])
                e = conv(e, W[j])
        e = c * e; L = add_sub(L, "+", e)
    return err, L

def ipL_val(X, y, T):
    '''Обчислення в точках T інтерполяційного поліному у формі Лагранжа.
```

```
        Вхідні параметри:
        X     - вектор вузлів інтерполяції;
        y     - вектор значень функції у вузлах сітки;
        T     - значення точок для обчислення.
        Вихідні параметри:
        err   - код закінчення:
                0 - перевірка вхідних даних пройшла успішно;
                1 - при перевірці вхідних даних знайдена помилка і
                    виконання функції аварійно закінчено;
        Lt    - значення інтерполяційного поліному Lm(T).
    '''
    if exist_coincidence(X) :
        err = 1

        print('ipL_val: Є кратні вузли інтерполювання !')
        return err, None

    n = len(X); err = 0
    Lt = np.zeros(len(T)); m = -1
    for t in T :
        e = 0
        for i in range(n) :
            c = y[i]
            for j in range(n) :
                if j != i :
                    c *= (t - X[j]) / (X[i] - X[j])
            e += c
        m += 1; Lt[m] = e
    return err, Lt

def ipN(X, y):
    '''Побудова інтерполяційного поліному у формі Ньютона.

        Вхідні параметри:
        X     - вектор вузлів інтерполяції;
        y     - вектор значень функції у вузлах сітки.
        Вихідні параметри:
        err   - код закінчення:
                0 - перевірка вхідних даних пройшла успішно;
                1 - при перевірці вхідних даних знайдена помилка і
                    виконання функції аварійно закінчено;
        N     - вектор, який є представленням інтерп.поліному Nm(x)
    '''
    if exist_coincidence(X) :
        err = 1

        print('ipN: Є кратні вузли інтерполювання !')
        return err, None

    n = len(X); n1 = n - 1; err = 0
    # обчислення розділених різниць;
    # обчислення поліномів Wi(x)= x-X(i), i=0,...,n-2
    d = deepcopy(y); W = []
    for i in range(n1) :
        for j in range(n1, i, -1) :
            k = j-1; d[j]=(d[k] - d[j]) / (X[k-i] - X[j])
        W.append(np.array([1, -X[i]]))
```

```
    # побудова інтерполяційного поліному Nm(t)
    j = 0; N = np.array([d[n1]])
    for i in range(n1-1, -1, -1) :
        N = conv(N, W[i])

        j += 1;  N[j] = N[j] + d[i]
    return err, N

def ipN_val(X, y, T):
    '''Обчислення в точах T інтерполяційного поліному у формі Ньютона.

       Вхідні параметри:
       X    - вектор вузлів інтерполяції;
       y    - вектор значень функції у вузлах сітки;
       T    - значення точок для обчислення.
       Вихідні параметри:
       err  - код закінчення:
              0 - перевірка вхідних даних пройшла успішно;
              1 - при перевірці вхідних даних знайдена помилка і
                  виконання функції аварійно закінчено;
       Nt   - значення інтерполяційного поліному Nm(T).
    '''
    if exist_coincidence(X) :
        err = 1

        print('ipN_val: Є кратні вузли інтерполювання !')
        return err, None

    n = len(X); n1 = n - 1; err = 0
    # обчислення розділених різниць
    d = deepcopy(y)
    for i in range(n1) :
        for j in range(n1, i, -1) :
            k = j-1; d[j]=(d[k] - d[j]) / (X[k-i] - X[j])
    # обчислення Nm(T)
    Nt = np.zeros(len(T)); m = -1
    for t in T :
        c = d[n1]
        for i in range(n1-1, -1, -1) :
            c = d[i] + c * (t - X[i])
        m += 1; Nt[m] = c
    return err, Nt

def spl01_val(X, y, T, find):
    '''Обчислення в точках T сплайна нульового степеня.

       Вхідні параметри:
       X    - вектор вузлів інтерполяції;
       y    - вектор значень функції у вузлах сітки;
       T    - значення точок для обчислення;
       find - алгоритм пошуку: 0= послідовний, 1= бінарний.
       Вихідні параметри:
       err  - код закінчення:
              0  - перевірка вхідних даних пройшла успішно;
              >0 - при перевірці вхідних даних знайдена помилка і
                   виконання функції аварійно закінчено;
       yt   - значення S01(T;f).
    '''
```

```
    if not is_order(X) :
        err = 1

        print('spl01_val: Вузли сітки не впорядковані або є кратні !')
        return err, None

    n = len(X)
    if X[0] <= np.min(T) and np.max(T) <= X[n-1] :
        err = 0
        yt = np.zeros(len(T))
        m = -1
        for t in T :
            i = find_Di(X, n, t, 1, find)
            m += 1
            yt[m] = y[i]
        return err, yt
    else :
        err = 2

        print('spl01_val: Є точки для обчислень (T), які поза межами
сітки (X) !')
        return err, None

def spl11_val(X, y, T, find):
    '''Обчислення в точках T сплайна першого степеня.

       Вхідні параметри:
       X    - вектор вузлів інтерполяції;
       y    - вектор значень функції у вузлах сітки;
       T    - значення точок для обчислення;
       find - алгоритм пошуку: 0= послідовний, 1= бінарний.
       Вихідні параметри:
       err  - код закінчення:
              0  - перевірка вхідних даних пройшла успішно;
              >0 - при перевірці вхідних даних знайдена помилка і
                   виконання функції аварійно закінчено;
       yt   - значення S11(T;f).
    '''
    if not is_order(X) :
        err = 1

        print('spl11_val: Вузли сітки не впорядковані або є кратні !')
        return err, None

    n = len(X)
    if X[0] <= np.min(T) and np.max(T) <= X[n-1] :
        err = 0
        yt = np.zeros(len(T))
        m = -1
        for t in T :
            i = find_Di(X, n, t, 0, find)
            j = i+1
            v = X[i]
            u = y[i]
            c = (y[j] - u) / (X[j] - v)
            m += 1; yt[m] = c * (t - v) + u
        return err, yt
    else :
```

```
        err = 2

        print('spl11_val: Є точки для обчислень (T), які поза межами
сітки (X) !')
        return err, None

def spl11(X, y):
    '''Побудова сплайна першого степеня.

       Вхідні параметри:
       X    - вектор вузлів інтерполяції;
       y    - вектор значень функції у вузлах сітки;
       Вихідні параметри:
       err  - код закінчення:
              0  - перевірка вхідних даних пройшла успішно;
              >0 - при перевірці вхідних даних знайдена помилка і
                   виконання функції аварійно закінчено;
       S11  - список векторів (кусково-лінійна функція S11(x;f)).
    '''

    if not is_order(X) :
        print('spl11: Вузли сітки не впорядковані або є кратні !')
        err = 1; return err, None

    n = len(X); n1 = n - 1
    err = 0; S11 = []
    for i in range(n1) :
        j = i+1; u = y[i]
        S11.append(np.array([(y[j]-u)/(X[j]-X[i]), u]))
    return err, S11

def spline11(X, S11, T, find):
    '''Обчислення в точках T сплайна першого степеня.

       Вхідні параметри:
       X    - вектор вузлів інтерполяції;
       S11  - список векторів (кусково-лінійна функція S11(x;f));
       T    - значення точок для обчислення;
       find - алгоритм пошуку: 0= послідовний, 1= бінарний.
       Вихідні параметри:
       err  - код закінчення:
              0  - перевірка вхідних даних пройшла успішно;
              >0 - при перевірці вхідних даних знайдена помилка і
                   виконання функції аварійно закінчено;
       yt   - значення S11(T;f).
    '''
    if not is_order(X) :
        print('spline11: Вузли сітки не впорядковані або є кратні !')
        err = 1; return err, None

    n = len(X)
    if X[0] <= np.min(T) and np.max(T) <= X[n-1] :
        err = 0; yt = np.zeros(len(T)); m = -1
        for t in T :
            i = find_Di(X, n, t, 0, find)
            t -= X[i]
            m += 1; yt[m] = S11[i][0]*t + S11[i][1]
```

```
        return err, yt
    else :
        print('spline11: Є точки для обчислень (T), які поза межами сітки
(X) !')
        err = 2; return err, None
```

**Програмний код Python головного модуля *main_interp* для тестування *interp*:**

```
import matplotlib.pyplot as plt
from util_lne import put_arr, movespinesticks
from interp import *


def runge(x):
    '''Приклад К.Рунге.'''
    return 1.0 / (1 + 25 * x * x)


Runge = np.vectorize(runge)

X = np.array([0, 1 / 6, 0.5])
y = np.sin(np.pi * X)
put_arr("       X =", X, "%17.9e", 5)
put_arr("sin(pi*X)=", y, "%17.9e", 5)
err, L = ipL(X, y)
if err != 1:
    print("\nІнтерполяційний поліном (форма Лагранжа):")
    put_poly("L(%i,x)" % (len(L) - 1), L)
    x = np.linspace(X[0] - 0.2, X[2] + 0.2, 91)  # сітка по x
    y1 = np.sin(np.pi * x)
    y2 = np.zeros_like(y1)
    i = 0
    for t in x:
        y2[i] = poly_val(L, t)
        i += 1
    plt.plot(x, y1, 'k')  # точні значення функції
    plt.plot(x[0:20], y2[0:20], 'b--',  # екстраполяція
             x[21:71], y2[21:71], 'b',  # інтерполяція
             x[72:], y2[72:], 'b--')  # екстраполяція
    plt.plot(X, y, 'bo')
    plt.axis('equal')
    # plt.axis([X[0]-0.3, X[2]+0.3, -1.0, 1.2])
    st = ["sin(pi*x)", "L(x) extrap", "L(x) interp"]
    plt.legend(st, loc="lower right")
    plt.grid(True)
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()

err, N = ipN(X, y)
if err != 1:
    print("Інтерполяційний поліном (форма Ньютона):")
    put_poly("N(%i,x)" % (len(N) - 1), N)

print()
```

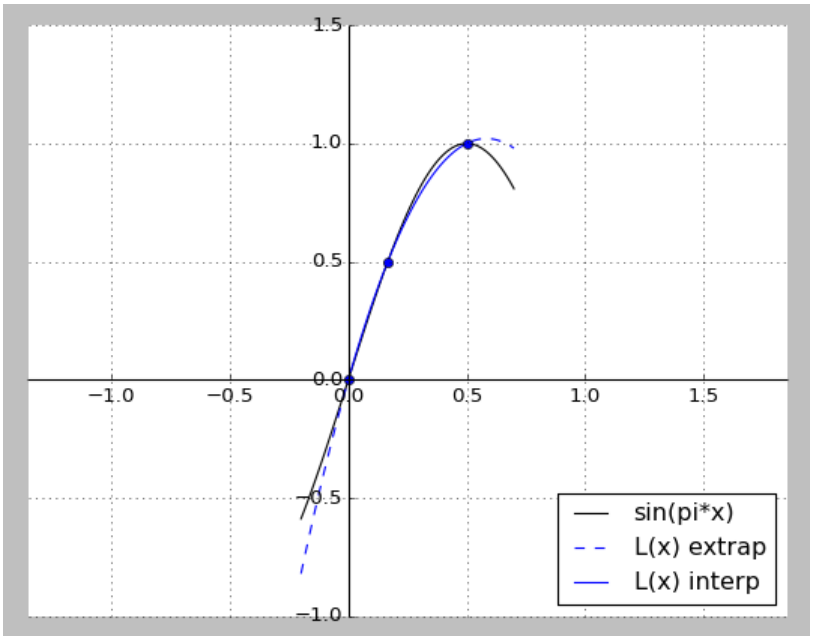
```
X = np.array([0, 1, 2, 3])
y = np.sin(np.pi / 6 * X)
put_arr("         X =", X, "%17.9e", 5)
put_arr("sin(pi/6*X)=", y, "%17.9e", 5)
T = np.array([1.5, 1.8])
t = np.sin(np.pi / 6 * T)
put_arr("  sin(pi/6*T)=", t, "%15.12f", 5)

print()
err, e = ipL_val(X, y, T)
if err == 0:
    d = abs(e - t)
    put_arr("ІП у ф.Лагранжа:=", e, "%19.12f", 5)
    put_arr("        похибка:=", d, "%19.12e", 5)
err, e = ipN_val(X, y, T)
if err == 0:
    d = abs(e - t)
    put_arr("ІП  у ф.Ньютона:=", e, "%19.12f", 5)
    put_arr("        похибка:=", d, "%19.12e", 5)

print("\nІнтерполяція поліномом у ф.Лагранжа на всьому [-1,1]\n" + \
      "і сплайнами S01(x;R), S11(x;R) для прикладу К.Рунге:")
a = -1.0
b = 1.0
X = np.linspace(a, b, 21)
Y = Runge(X)
x = np.linspace(a, b, 201)
y = Runge(x)
err, L = ipL(X, Y)
yL = np.zeros_like(x)
i = 0
for t in x:
    yL[i] = poly_val(L, t)
    i += 1
err, S0 = spl01_val(X, Y, x, 1)
err, S1 = spl11_val(X, Y, x, 1)
print("‖L(x)    -R(x)‖=%13.6e\n" % np.linalg.norm(yL - y, np.inf) + \
      "‖S01(x;R)-R(x)‖=%13.6e\n" % np.linalg.norm(S0 - y, np.inf) + \
      "‖S11(x;R)-R(x)‖=%13.6e" % np.linalg.norm(S1 - y, np.inf))

plt.plot(x, y, 'k')  # точні значення функції
plt.plot(x, yL, 'b')  # інтерполяція по Лагранжу
plt.plot(X, Y, 'bo')  # дані для інтерполяції
plt.axis([a - 0.1, b + 0.1, -1, 4.5])
plt.legend(["Runge(x)", "L(x)", "data"], loc="center")
movespinesticks()
plt.xlabel('x')
plt.ylabel('y')
plt.show()

plt.plot(x, y, 'k')  # точні значення функції
plt.plot(x, S0, 'g--')  # кусково-стала
plt.plot(x, S1, 'm')  # кусково-лінійна
plt.plot(X, Y, 'bo')  # дані для інтерполяції
plt.axis('equal')
# plt.axis([a-0.1, b+0.1, -0.1, 1.1])
plt.legend(["Runge(x)", "S01(x;R)", "S11(x;R)", "data"], loc="best")
movespinesticks()
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.show()

print()
a = 0.0
b = np.pi / 2
X = np.linspace(a, b, 6)
Y = np.cos(X)
put_arr("    X =", X, "%12.8f", 6)
put_arr("cos(X)=", Y, "%12.8f", 6)
err, S11 = spl11(X, Y)
print("Сплайн S11(x;cos):=")
for s in S11:
    print(s)
x = np.linspace(a, b, 21)
y = np.cos(x)
err, S1 = spline11(X, S11, x, 1)
print("‖S11(x;cos)-cos(x)‖=%13.6e" % np.linalg.norm(S1 - y, np.inf))

fig, ax = plt.subplots()
ax.plot(x, y, color='black', label="cos(x)")  # точні значення функції
ax.plot(x, S1, color='m', label="S11(x;cos)")  # кусково-лінійна
ax.plot(X, Y, 'bo', label="data")  # дані для інтерполяції
ax.axis('equal')
# plt.axis([a-0.1, b+0.1, -0.1, 1.1])
plt.legend(loc="best")
movespinesticks()
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
fig.savefig("int4.png")

print("\nВикористання функції np.interp() - \n",
      "одновимірна кусково-лінійна інтерполяція.")
a = -1.0
b = 1.0
X = np.linspace(a, b, 21)
Y = Runge(X)
x = np.linspace(a, b, 201)
y = Runge(x)
s1 = np.interp(x, X, Y)
plt.plot(x, y, 'k')  # точні значення функції
plt.plot(x, s1, 'm')  # кусково-лінійна np.interp()
plt.plot(X, Y, 'bo')  # дані для інтерполяції
plt.axis('equal')
plt.legend(["Runge(x)", "np.interp()", "data"], loc="best")
movespinesticks()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

**Результат виконання *`main_interp`*:**

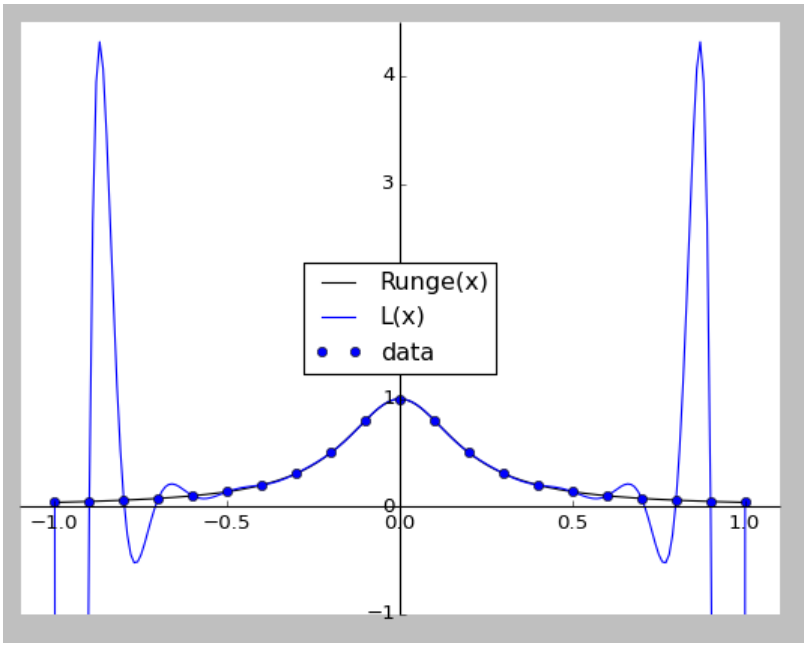
```
    X =
 0.000000000e+00  1.666666667e-01  5.000000000e-01
sin(pi*X)=
 0.000000000e+00  5.000000000e-01  1.000000000e+00
Інтерполяційний поліном (форма Лагранжа):
```

L(2,x):= -3*x^2+3.5*x
Інтерполяційний поліном (форма Ньютона):
N(2,x):= -3*x^2+3.5*x
X =
0.000000000e+00 1.000000000e+00 2.000000000e+00 3.000000000e+00
sin(pi/6*X)=
0.000000000e+00 5.000000000e-01 8.660254038e-01 1.000000000e+00
sin(pi/6*T)=
0.707106781187 0.809016994375

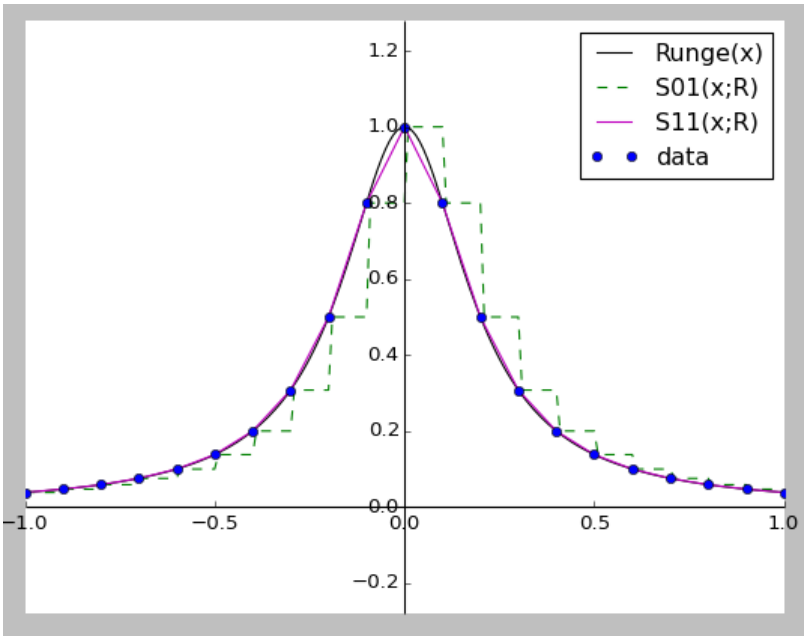



ІП у ф.Лагранжа:=
0.705889289629 0.808245948870
похибка:=
1.217491557801e-03 7.710455051926e-04
ІП у ф.Ньютона:=
0.705889289629 0.808245948870
похибка:=
1.217491557801e-03 7.710455051925e-04

Інтерполяція поліномом у ф.Лагранжа на всьому [-1,1]



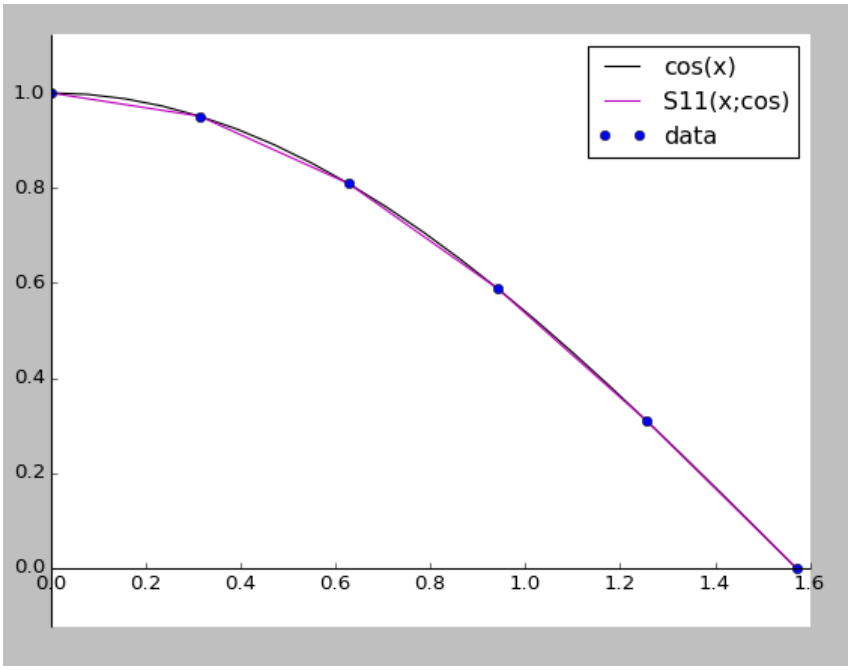

і сплайнами S01(x;R), S11(x;R) для прикладу К.Рунге:
‖L(x)  -R(x)‖= 5.858549e+01
‖S01(x;R)-R(x)‖= 3.000000e-01
‖S11(x;R)-R(x)‖= 4.153846e-02



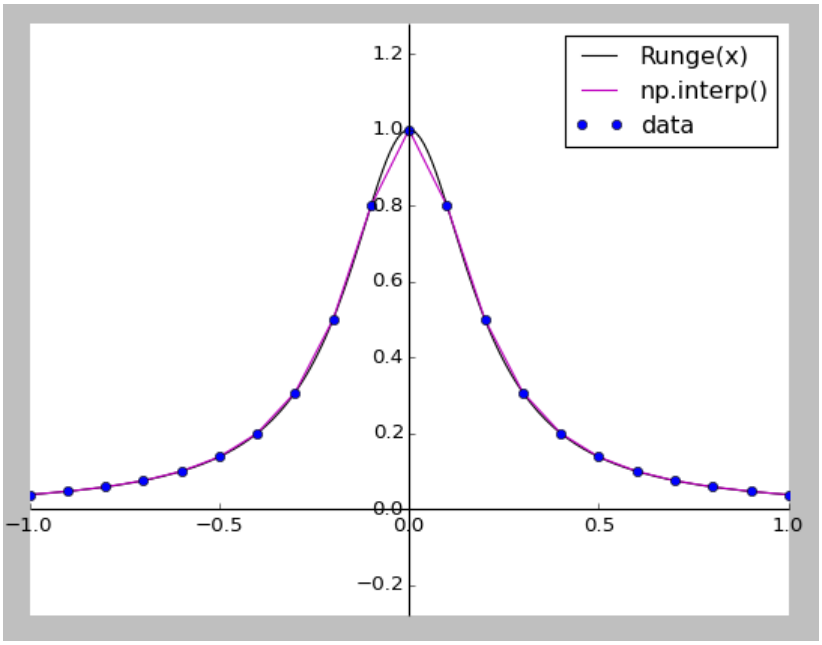

X =
0.00000000  0.31415927  0.62831853  0.94247780  1.25663706  1.57079633

cos(X)=
1.00000000  0.95105652  0.80901699  0.58778525  0.30901699  0.00000000

Сплайн S11(x;cos):=
[-0.15579195  1.        ]
[-0.45212584  0.95105652]
[-0.70420251  0.80901699]
[-0.88734692  0.58778525]
[-0.98363164  0.30901699]
‖S11(x;cos)-cos(x)‖= 1.216008e-02




Використання функції np.interp() -
одновимірна кусково-лінійна інтерполяція.



## V. Модуль числового диференціювання.

**В модуль *num_differ.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *deriv_1x(y,x,num_pnt=3,dif_rel=0)* | числове диференціювання для 1-ї похідної на нерівномірній сітці |
| *deriv_1h(y,h,num_pnt=3,dif_rel=0)* | числове диференціювання для 1-ї похідної на рівномірній сітці |
| *deriv_2x(y,x)* | числове диференціювання для 2-ї похідної на нерівномірній сітці |
| *deriv_2h(y,h,num_pnt=3)* | числове диференціювання для 2-ї похідної на рівномірній сітці |

**Програмний код Python функцій модуля :**

```
""" Модуль числового диференціювання."""

import numpy as np
from interp import is_order


def deriv_1x(y, x, num_pnt=3, dif_rel=0):
    '''Числове диференціювання для 1-ї похідної на нерівномірній сітці.

       Вхідні параметри:
       y       - вектор значень функції у вузлах сітки;
       x       - вектор нерівномірної сітки;
       num_pnt - кількість точок в різницевій формулі, при цьому
                 якщо num_pnt=3, то параметр dif_rel не враховується;
       dif_rel - код різницевого відношення (тільки для num_pnt=2) :
                 -1 = ліва різниця;
                 +1 = права різниця.
       Вихідні параметри:
       err     - код закінчення:
                  0 - перевірка вхідних даних пройшла успішно;
                 >0 - при перевірці вхідних даних знайдена помилка і
                      виконання функції аварійно закінчено;
       dy      - вектор першої похідної, при цьому для num_pnt=2 і :
                 dif_rel=-1 значення першого елемента вектора dy[0]=NaN;
                 dif_rel=+1 значення останього елемента вектора dy[-
1]=NaN.
    '''

    n = len(y)
    if n != len(x):
        print('deriv_1x: Розмірність сітки x ≠ y - розмірності функції
!')
        return 1, None
    if n < num_pnt:
        print('deriv_1x: Розмірність сітки x < значення num_pnt !')
        return 2, None
    if not is_order(x):
        print('deriv_1x: Вузли сітки не впорядковані або є кратні !')
        return 3, None

    n1 = n - 1
```

```
    err = 0
    dy = np.empty(n)
    if num_pnt == 2:  # по 2-х точках
        if dif_rel == 1:  # права різниця
            dy[n1] = np.NaN
            j = -1
            ib = 1
            ie = n
        elif dif_rel == -1:  # ліва різниця
            dy[0] = np.NaN
            j = 0
            ib = 0
            ie = n1
        else:
            print('deriv_1x: Для num_pnt=2 параметр dif_rel '
                  'має належати {-1;1} !')
            return 4, None

        for i in range(ib, ie):
            j += 1
            dy[j] = (y[j] - y[i]) / (x[j] - x[i])

    elif num_pnt == 3:  # по 3-х точках
        hi1 = x[1] - x[0]
        zz = y[1] - y[0]
        hi = x[2] - x[1]
        hh = hi1 + hi
        z = y[2] - y[1]
        dy[0] = ((hi1 + hh) / hi1 * zz - hi1 / hi * z) / hh
        for i in range(1, n1):
            j = i + 1
            hi = hi1
            hi1 = x[j] - x[i]
            hh = hi1 + hi
            z = zz
            zz = y[j] - y[i]
            dy[i] = (hi / hi1 * zz + hi1 / hi * z) / hh
        dy[n1] = ((hi1 + hh) / hi1 * zz - hi1 / hi * z) / hh

    else:
        print('deriv_1x: Параметр num_pnt має належати {2;3} !')
        err = 5
        dy = None

    return err, dy


def deriv_1h(y, h, num_pnt=3, dif_rel=0):
    '''Числове диференціювання для 1-ї похідної на рівномірній сітці.

       Вхідні параметри:
       y       - вектор значень функції у вузлах сітки;
       h       - крок рівномірної сітки;
       num_pnt - кількість точок в різницевій формулі, при цьому
                 якщо num_pnt >2, то параметр dif_rel не враховується;
       dif_rel - код різницевого відношення (тільки для num_pnt=2) :
                 -1 = ліва різниця;
                 +1 = права різниця.
```

```
        Вихідні параметри:
        err     - код закінчення:
                  0 - перевірка вхідних даних пройшла успішно;
                 >0 - при перевірці вхідних даних знайдена помилка і
                      виконання функції аварійно закінчено;
        dy      - вектор першої похідної, при цьому для num_pnt=2 і :
                  dif_rel=-1 значення першого елемента вектора dy[0]=NaN;
                  dif_rel=+1 значення останього елемента вектора dy[-
1]=NaN.
    '''
    n = len(y)
    if n < num_pnt:
        print('deriv_1h: Розмірність функції y < значення num_pnt !')
        return 1, None

    n1 = n - 1
    err = 0
    dy = np.empty(n)
    if num_pnt == 2:  # по 2-х точках
        if dif_rel == 1:  # права різниця
            dy[n1] = np.NaN
            j = -1
            ib = 1
            ie = n
            e = -1 / h
        elif dif_rel == -1:  # ліва різниця
            dy[0] = np.NaN
            j = 0
            ib = 0
            ie = n1
            e = 1 / h
        else:
            print('deriv_1h: Для num_pnt=2 параметр dif_rel '
                  'має належати {-1; 1} !')
            return 2, None

        for i in range(ib, ie):
            j += 1
            dy[j] = (y[j] - y[i]) * e

    elif num_pnt == 3:  # по 3-х точках
        e = 0.5 / h
        u0 = y[0]
        u1 = y[1]
        u2 = y[2]
        dy[0] = (-3 * u0 + 4 * u1 - u2) * e
        dy[1] = (u2 - u0) * e
        for i in range(2, n1):
            u0 = u1
            u1 = u2
            u2 = y[i + 1]
            dy[i] = (u2 - u0) * e
        dy[n1] = (u0 - 4 * u1 + 3 * u2) * e

    elif num_pnt == 4:  # по 4-х точках
        e = 1 / (6 * h)
        n2 = n1 - 1
        u0 = y[0]
```

```
        u1 = y[1]
        u2 = y[2]
        u3 = y[3]
        dy[0] = (-11 * u0 + 18 * u1 - 9 * u2 + 2 * u3) * e
        dy[1] = (-2 * u0 - 3 * u1 + 6 * u2 - u3) * e
        dy[2] = (u0 - 6 * u1 + 3 * u2 + 2 * u3) * e
        for i in range(3, n1):
            u0 = u1
            u1 = u2
            u2 = u3
            u3 = y[i + 1]
            dy[i] = (u0 - 6 * u1 + 3 * u2 + 2 * u3) * e
        dy[n1] = (-2 * u0 + 9 * u1 - 18 * u2 + 11 * u3) * e

    elif num_pnt == 5:  # по 5-ти точках
        e = 1 / (12 * h)
        n2 = n1 - 1
        n3 = n2 - 1
        u0 = y[0]
        u1 = y[1]
        u2 = y[2]
        u3 = y[3]
        u4 = y[4]
        dy[0] = (-25 * u0 + 48 * u1 - 36 * u2 + 16 * u3 - 3 * u4) * e
        dy[1] = (-3 * u0 - 10 * u1 + 18 * u2 - 6 * u3 + u4) * e
        dy[2] = (u0 - 8 * (u1 - u3) - u4) * e
        for i in range(3, n2):
            u0 = u1
            u1 = u2
            u2 = u3
            u3 = u4
            u4 = y[i + 2]
            dy[i] = (u0 - 8 * (u1 - u3) - u4) * e
        dy[n2] = (-u0 + 6 * u1 - 18 * u2 + 10 * u3 + 3 * u4) * e
        dy[n1] = (3 * u0 - 16 * u1 + 36 * u2 - 48 * u3 + 25 * u4) * e

    else:
        print('deriv_1h: Параметр num_pnt має належати {2;3;4;5} !')
        err = 3
        dy = None

    return err, dy


def deriv_2x(y, x):
    '''Числове диференціювання для 2-ї похідної на нерівномірній сітці.

       Вхідні параметри:
       y   - вектор значень функції у вузлах сітки;
       x   - вектор нерівномірної сітки;
       Вихідні параметри:
       err - код закінчення:
              0 - перевірка вхідних даних пройшла успішно;
             >0 - при перевірці вхідних даних знайдена помилка і
                  виконання функції аварійно закінчено;
       dy  - вектор другої похідної, де dy[0]=dy[1], dy[-1]=dy[-2].
    '''
```

```
    num_pnt = 3  # кількість точок сітки в різницевій формулі

    n = len(y)
    if n != len(x):
        print('deriv_2x: Розмірність сітки x ≠ y - розмірності функції
!')
        return 1, None
    if n < num_pnt:
        print('deriv_2x: Розмірність сітки x < 3 !')
        return 2, None
    if not is_order(x):
        print('deriv_2x: Вузли сітки не впорядковані або є кратні !')
        return 3, None

    n1 = n - 1
    err = 0
    dy = np.empty(n)
    hi1 = x[1] - x[0]
    zz = y[1] - y[0]
    for i in range(1, n1):
        j = i + 1
        hi = hi1
        hi1 = x[j] - x[i]
        hh = 0.5 * (hi1 + hi)
        z = zz
        zz = y[j] - y[i]
        dy[i] = (zz / hi1 - z / hi) / hh
    dy[0] = dy[1]
    dy[n1] = dy[n1 - 1]

    return err, dy


def deriv_2h(y, h, num_pnt=3):
    '''Числове диференціювання для 2-ї похідної на рівномірній сітці.

       Вхідні параметри:
       y       - вектор значень функції у вузлах сітки;
       h       - крок рівномірної сітки;
       num_pnt - кількість точок в різницевій формулі (num_pnt >2).
       Вихідні параметри:
       err     - код закінчення:
                 0 - перевірка вхідних даних пройшла успішно;
                >0 - при перевірці вхідних даних знайдена помилка і
                     виконання функції аварійно закінчено;
       dy      - вектор другої похідної.
    '''
    if num_pnt < 3:
        print('deriv_2h: Значення num_pnt < 3 !')
        return 1, None

    n = len(y)
    if n < num_pnt:
        print('deriv_2h: Розмірність функції y < значення num_pnt !')
        return 2, None

    n1 = n - 1
    e = 1 / (h * h)
```

```
err = 0
dy = np.empty(n)
if num_pnt == 3:  # по 3-х точках
    u0 = y[0]
    u1 = y[1]
    u2 = y[2]
    z = (u0 - 2 * u1 + u2) * e
    dy[0] = z
    dy[1] = z
    for i in range(2, n1):
        u0 = u1
        u1 = u2
        u2 = y[i + 1]
        z = (u0 - 2 * u1 + u2) * e
        dy[i] = z
    dy[n1] = z

elif num_pnt == 4:  # по 4-х точках
    n2 = n1 - 1
    u0 = y[0]
    u1 = y[1]
    u2 = y[2]
    u3 = y[3]
    dy[0] = (2 * u0 - 5 * u1 + 4 * u2 - u3) * e
    dy[1] = (u0 - 2 * u1 + u2) * e
    dy[2] = (u1 - 2 * u2 + u3) * e
    for i in range(3, n1):
        u0 = u1
        u1 = u2
        u2 = u3
        u3 = y[i + 1]
        dy[i] = (u1 - 2 * u2 + u3) * e
    dy[n1] = (-u0 + 4 * u1 - 5 * u2 + 2 * u3) * e

elif num_pnt == 5:  # по 5-ти точках
    e /= 12
    n2 = n1 - 1
    n3 = n2 - 1
    u0 = y[0]
    u1 = y[1]
    u2 = y[2]
    u3 = y[3]
    u4 = y[4]
    dy[0] = (35 * u0 - 104 * u1 + 114 * u2 - 56 * u3 + 11 * u4) * e
    dy[1] = (11 * u0 - 20 * u1 + 6 * u2 + 4 * u3 - u4) * e
    dy[2] = (-u0 + 16 * (u1 + u3) - 30 * u2 - u4) * e
    for i in range(3, n2):
        u0 = u1
        u1 = u2
        u2 = u3
        u3 = u4
        u4 = y[i + 2]
        dy[i] = (-u0 + 16 * (u1 + u3) - 30 * u2 - u4) * e
    dy[n2] = (-u0 + 4 * u1 + 6 * u2 - 20 * u3 + 11 * u4) * e
    dy[n1] = (11 * u0 - 56 * u1 + 114 * u2 - 104 * u3 + 35 * u4) * e

else:
    print('deriv_2h: Параметр num_pnt має належати {3;4;5} !')
```

```
        err = 3
        dy = None

    return err, dy
```

**Програмний код Python головного модуля *main_num_differ* для тестування *num_differ*:**

```
from num_differ import *
from util_lne import put_arr

print("ЧИСЛОВЕ ДИФЕРЕНЦІЮВАННЯ ДЛЯ ПОХІДНИХ ПЕРШОГО ПОРЯДКУ")
print()
x = np.array([1, 3, 6, 10, 12.])
y = x ** 3
Dy = 3 * x ** 2
put_arr("Нерівномірна сітка x", x, "%7.1f", 11)
put_arr("функція u(x)=x^3", y, "%7.1f", 11)
put_arr("точна u'(x)=3*x^2", Dy, "%7.1f", 11)
err, d1_2L = deriv_1x(y, x, num_pnt=2, dif_rel=-1)
put_arr("точок=2,  ліва різниця=", d1_2L, "%7.1f", 11)
err, d1_2R = deriv_1x(y, x, num_pnt=2, dif_rel=1)
put_arr("точок=2, права різниця=", d1_2R, "%7.1f", 11)
err, d1_3 = deriv_1x(y, x)
put_arr("точок=3, центральна р.=", d1_3, "%7.1f", 11)

print("")
xh = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10.])
h = xh[1] - xh[0]
yh = xh ** 3
Dy = 3 * xh ** 2
put_arr("  Рівномірна сітка x", xh, "%7.1f", 11)
print("Крок сітки h= %f" % h)
put_arr("функція u(x)=x^3", yh, "%7.1f", 11)
put_arr("точна u'(x)=3*x^2", Dy, "%7.1f", 11)
err, d1_2L = deriv_1h(yh, h, num_pnt=2, dif_rel=-1)
put_arr("точок=2,  ліва різниця=", d1_2L, "%7.1f", 11)
err, d1_2L = deriv_1x(yh, xh, num_pnt=2, dif_rel=-1)
put_arr("точок=2,  ліва різниця (алгор.нерівн.сітки)=",
        d1_2L, "%7.1f", 11)
err, d1_2R = deriv_1h(yh, h, num_pnt=2, dif_rel=1)
put_arr("точок=2, права різниця=", d1_2R, "%7.1f", 11)
err, d1_2R = deriv_1x(yh, xh, num_pnt=2, dif_rel=1)
put_arr("точок=2, права різниця (алгор.нерівн.сітки)=",
        d1_2R, "%7.1f", 11)
err, d1_3 = deriv_1h(yh, h)
put_arr("точок=3, центральна р.=", d1_3, "%7.1f", 11)
err, d1_3 = deriv_1x(yh, xh)
put_arr("точок=3, центральна р. (алгор.нерівн.сітки)=",
        d1_3, "%7.1f", 11)
err, d1_4 = deriv_1h(yh, h, num_pnt=4)
put_arr("точок=4 =", d1_4, "%7.1f", 11)
err, d1_5 = deriv_1h(yh, h, num_pnt=5)
put_arr("точок=5 =", d1_5, "%7.1f", 11)

print("\nЧИСЛОВЕ ДИФЕРЕНЦІЮВАННЯ ДЛЯ ПОХІДНИХ ДРУГОГО ПОРЯДКУ")
DDy = 6 * x
```

```
put_arr("\nНерівномірна сітка x", x, "%7.1f", 11)
put_arr("функція u(x)=x^3", y, "%7.1f", 11)
put_arr('точна u"(x)=6*x', DDy, "%7.1f", 11)
err, d2_3 = deriv_2x(y, x)
put_arr("точок=3, центральна р.=", d2_3, "%7.1f", 11)

print("")
DDy = 6 * xh
put_arr("  Рівномірна сітка x", xh, "%7.1f", 11)
print("Крок сітки h= %f" % h)
put_arr("функція u(x)=x^3", yh, "%7.1f", 11)
put_arr('точна u"(x)=6*x', DDy, "%7.1f", 11)
err, d2_3 = deriv_2h(yh, h)
put_arr("точок=3, центральна р.=", d2_3, "%7.1f", 11)
err, d2_3 = deriv_2x(yh, xh)
put_arr("точок=3, центральна р. (алгор.нерівн.сітки)=",
        d2_3, "%7.1f", 11)
err, d2_4 = deriv_2h(yh, h, num_pnt=4)
put_arr("точок=4 =", d2_4, "%7.1f", 11)
err, d2_5 = deriv_2h(yh, h, num_pnt=5)
put_arr("точок=5 =", d2_5, "%7.1f", 11)
```

### Результат виконання ***main_num_differ***:

```
ЧИСЛОВЕ ДИФЕРЕНЦІЮВАННЯ ДЛЯ ПОХІДНИХ ПЕРШОГО ПОРЯДКУ

Нерівномірна сітка x
   1.0   3.0   6.0  10.0  12.0
функція u(x)=x^3
   1.0  27.0 216.0 1000.0 1728.0
точна u'(x)=3*x^2
   3.0  27.0 108.0 300.0 432.0
точок=2,  ліва різниця=
   nan  13.0  63.0 196.0 364.0
точок=2, права різниця=
  13.0  63.0 196.0 364.0   nan
точок=3, центральна р.=
  -7.0  33.0 120.0 308.0 420.0

  Рівномірна сітка x
   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0  10.0
Крок сітки h= 1.000000
функція u(x)=x^3
   1.0   8.0  27.0  64.0 125.0 216.0 343.0 512.0 729.0 1000.0
точна u'(x)=3*x^2
   3.0  12.0  27.0  48.0  75.0 108.0 147.0 192.0 243.0 300.0
точок=2,  ліва різниця=
   nan   7.0  19.0  37.0  61.0  91.0 127.0 169.0 217.0 271.0
точок=2,  ліва різниця (алгор.нерівн.сітки)=
   nan   7.0  19.0  37.0  61.0  91.0 127.0 169.0 217.0 271.0
точок=2, права різниця=
   7.0  19.0  37.0  61.0  91.0 127.0 169.0 217.0 271.0   nan
точок=2, права різниця (алгор.нерівн.сітки)=
   7.0  19.0  37.0  61.0  91.0 127.0 169.0 217.0 271.0   nan
точок=3, центральна р.=
   1.0  13.0  28.0  49.0  76.0 109.0 148.0 193.0 244.0 298.0
точок=3, центральна р. (алгор.нерівн.сітки)=
   1.0  13.0  28.0  49.0  76.0 109.0 148.0 193.0 244.0 298.0
точок=4 =
   3.0  12.0  27.0  48.0  75.0 108.0 147.0 192.0 243.0 300.0
точок=5 =
   3.0  12.0  27.0  48.0  75.0 108.0 147.0 192.0 243.0 300.0
```

ЧИСЛОВЕ ДИФЕРЕНЦІЮВАННЯ ДЛЯ ПОХІДНИХ ДРУГОГО ПОРЯДКУ

Нерівномірна сітка x
1.0 3.0 6.0 10.0 12.0
функція u(x)=x^3
1.0 27.0 216.0 1000.0 1728.0
точна u"(x)=6*x
6.0 18.0 36.0 60.0 72.0
точок=3, центральна р.=
20.0 20.0 38.0 56.0 56.0

Рівномірна сітка x
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0
Крок сітки h= 1.000000
функція u(x)=x^3
1.0 8.0 27.0 64.0 125.0 216.0 343.0 512.0 729.0 1000.0
точна u"(x)=6*x
6.0 12.0 18.0 24.0 30.0 36.0 42.0 48.0 54.0 60.0
точок=3, центральна р.=
12.0 12.0 18.0 24.0 30.0 36.0 42.0 48.0 54.0 54.0
точок=3, центральна р. (алгор.нерівн.сітки)=
12.0 12.0 18.0 24.0 30.0 36.0 42.0 48.0 54.0 54.0
точок=4 =
6.0 12.0 18.0 24.0 30.0 36.0 42.0 48.0 54.0 60.0
точок=5 =
6.0 12.0 18.0 24.0 30.0 36.0 42.0 48.0 54.0 60.0

## VI. Модуль числового інтегрування.

### В модуль *num_integr.py* входять наступні функції :

| Назва функції | Призначення |
|---|---|
| *i_c_rectangle(x,f)* | знаходження методом центральних прямокутників визначеного інтеграла |
| *i_trap(a,b,f,eps=1e-7,mki=1000)* | знаходження методом трапецій визначеного інтеграла |
| *i_simp(a,b,f,eps=1e-7,mki=1000)* | знаходження методом Сімпсона визначеного інтеграла |
| *i_gauss(a,b,u,n)* | знаходження інтеграла за квадратурною формулою Гаусса |
| *i_g_cheb(a,b,u,n)* | знаходження інтеграла за квадратурною формулою Гаусса-Чебишова |
| *i_laguerre(u,n)* | знаходження інтеграла за квадратурною формулою Лаггера |
| *i_hermite(u,n)* | знаходження інтеграла за квадратурною формулою Ерміта |
| *simpson(F, h_X)* | метод Сімпсона для дискретної функції F |

### Програмний код Python функцій модуля :

```
""" Модуль числового інтегрування."""

import numpy as np
from interp import is_order


def i_c_rectangle(x, f):
    '''Знаходження методом центральних прямокутників визначеного
інтеграла.
                b
          s =∫f(x)dx
                a
        Вхідні параметри:
        x    - квадратурні вузли;
        f    - функція користувача (скалярна), з описом f(x);
        Вихідні параметри:
        err      - код закінчення:
                   0 - перевірка вхідних даних пройшла успішно;
                  >0 - при перевірці вхідних даних знайдена помилка і
                       виконання функції аварійно закінчено;
        s    - значення інтеграла.
    '''
    n = len(x)
    if n < 2:
        print('i_c_rectangle: Кількість квадратурних вузлів x < 2 !')
        return 1, None
    if not is_order(x):
        print('i_c_rectangle: Вузли сітки не впорядковані або є кратні
!')
        return 2, None

    err = 0
    s = 0.0
    xr = x[0]
    for i in range(n - 1):
```

```
        xl = xr
        xr = x[i + 1]
        h = xr - xl
        s += h * f(0.5 * (xl + xr))
    return err, s


def i_trap(a, b, f, eps=1e-7, mki=1000):
    '''Знаходження методом трапецій визначеного інтеграла.
           b
        s =∫f(x)dx
           a
       Вхідні параметри:
       a   - нижня границя інтеграла;
       b   - верхня границя інтеграла;
       f   - функція користувача (скалярна), з описом f(x);
       eps - бажана точність. Якщо не виконується умова 0 <eps <1,
             то покладаємо eps = 1e-7;
       mki - максимальна кількість згущення сітки.
       Вихідні параметри:
       s   - значення інтеграла.
    '''
    if eps <= 0 or eps >= 1: eps = 1e-7
    if a > b:
        A = b
        B = a
        z = -1
    else:
        A = a
        B = b
        z = 1

    # Рекурентний алгоритм
    hn = B - A
    S1 = f(A) + f(B)
    S2 = 0.0
    s = 0.5 * hn * S1
    k = 0
    F = False
    while not F and k < mki:
        k += 1
        ho = hn
        hn *= 0.5

        x = A + hn
        while x < B:
            S2 += f(x)
            x += ho

        ss = 0.5 * hn * (S1 + 2 * S2)
        F = abs(ss - s) / 3 < eps
        s = ss
    return z * s


def i_simp(a, b, f, eps=1e-7, mki=1000):
    '''Знаходження методом Сімпсона визначеного інтеграла.
           b
```

```
        s =∫f(x)dx
           a
       Вхідні параметри:
       a   - нижня границя інтеграла;
       b   - верхня границя інтеграла;
       f   - функція користувача (скалярна), з описом f(x);
       eps - бажана точність. Якщо не виконується умова 0 <eps <1,
             то покладаємо eps = 1e-7;
       mki - максимальна кількість згущення сітки.
       Вихідні параметри:
       s   - значення інтеграла.
    '''
    if eps <= 0 or eps >= 1: eps = 1e-7
    if a > b:
        A = b
        B = a
        z = -1
    else:
        A = a
        B = b
        z = 1

    # Рекурентний алгоритм
    hn = 0.5 * (B - A)
    S1 = f(A) + f(B)
    S2 = 0.0
    S4 = f(0.5 * (A + B))
    s = hn * (S1 + 4 * S4) / 3

    k = 0
    F = False
    while not F and k < mki:
        k += 1
        ho = hn
        hn *= 0.5
        S2 += S4
        S4 = 0.0

        x = A + hn
        while x < B:
            S4 += f(x)
            x += ho

        ss = hn * (S1 + 4 * S4 + 2 * S2) / 3
        F = abs(ss - s) / 15 < eps
        s = ss
    return z * s


def i_gauss(a, b, u, n):
    '''Знаходження за квадратурною формулою Гаусса інтеграла.
           b
        s =∫u(x)dx,   (ρ(x)=1)
           a
       Вхідні параметри:
       a   - нижня границя інтеграла;
       b   - верхня границя інтеграла;
       u   - функція користувача (скалярна), з описом u(x);
```

```
    n    - кількість вузлів (0 < n < 8).
    Вихідні параметри:
    err      - код закінчення:
                0 - перевірка вхідних даних пройшла успішно;
               >0 - при перевірці вхідних даних знайдена помилка і
                    виконання функції аварійно закінчено;
    s    - значення інтеграла.
'''
if n < 1 or n > 7:
    print('i_gauss: Значення n має бути {1,2,3,4,5,6,7} !')
    return 1, None

if a > b:
    A = b
    B = a
    z = -1
else:
    A = a
    B = b
    z = 1

# в залежності від n визначаємо вузли і вагові коефіцієнти :
X = np.zeros(n)
C = np.ones(n)
if n == 1:
    pass
elif n == 2:
    C[1] = 0.5
    X[1] = 0.5773502691896258
elif n == 3:
    C[2] = 5 / 18
    X[2] = 0.7745966692414834
    C[1] = 4 / 9
elif n == 4:
    C[3] = 0.1739274225687284
    X[3] = 0.8611363115940492
    C[2] = 0.3260725774312716
    X[2] = 0.3399810435848646
elif n == 5:
    C[4] = 0.1184634425280945
    X[4] = 0.9061798459386640
    C[3] = 0.2393143352496832
    X[3] = 0.5384693101056830
    C[2] = 64 / 225
elif n == 6:
    C[5] = 0.0856622461895852
    X[5] = 0.9324695142031520
    C[4] = 0.1803807865240693
    X[4] = 0.6612093864662644
    C[3] = 0.2339569672863455
    X[3] = 0.2386191860831970
elif n == 7:
    C[6] = 0.0647424830844348
    X[6] = 0.9491079123427596
    C[5] = 0.1398526957446384
    X[5] = 0.7415311855993944
    C[4] = 0.1909150252525595
    X[4] = 0.4058451513773970
```

```
        C[3] = 256 / 1225
    for i in range(n // 2):
        j = -1 - i
        X[i] = -X[j]
        C[i] = C[j]
    C *= 2.0

    # якщо [A,B]<>[-1,1], то виконуємо лінійне перетворення :
    if A != -1 or B != 1:
        ab = 0.5 * (A + B)
        ba = 0.5 * (B - A)
        for i in range(n):
            X[i] = ab + ba * X[i]
            C[i] = ba * C[i]

    # наближене значення інтеграла :
    err = 0
    s = 0.0
    for i in range(n):
        s += C[i] * u(X[i])
    return err, z * s


def i_g_cheb(a, b, u, n):
    '''Знаходження за квадратурною формулою Гаусса-Чебишова інтеграла.
            b
        s =∫ρ(x)*u(x)dx, ρ(x)=1/sqrt(1-x^2)
            a
       Вхідні параметри:
       a    - нижня границя інтеграла;
       b    - верхня границя інтеграла;
       u    - функція користувача (скалярна), з описом u(x);
       n    - кількість вузлів n > 0.
       Вихідні параметри:
       err      - код закінчення:
                  0 - перевірка вхідних даних пройшла успішно;
                 >0 - при перевірці вхідних даних знайдена помилка і
                      виконання функції аварійно закінчено;
       s    - значення інтеграла.
    '''
    if n < 1:
        print('i_g_cheb: Значення n має бути > 0 !')
        return 1, None

    if a > b:
        A = b
        B = a
        z = -1
    else:
        A = a
        B = b
        z = 1

    # в залежності від n визначаємо вузли і вагові коефіцієнти :
    C = np.pi / n
    X = np.linspace(1, n, n)
    X = np.cos(C * (X - 0.5))
    i = n - n // 2
```

```
    if i * 2 > n: X[i - 1] = 0.0

    # якщо [A,B]<>[-1,1], то виконуємо лінійне перетворення :
    if A != -1 or B != 1:
        ab = 0.5 * (A + B)
        ba = 0.5 * (B - A)
        X = ba * X + ab
        C *= ba

    # наближене значення інтеграла :
    err = 0
    s = 0.0
    for i in range(n):
        s += u(X[i])
    s *= C
    return err, z * s


def i_laguerre(u, n):
    '''Знаходження за квадратурною формулою Лагерра інтеграла.
            ∞
        s =∫ρ(x)*u(x)dx, ρ(x)=x^α*exp(-x), α=0 (в загальному α>-1)
            0
       Вхідні параметри:
       u    - функція користувача (скалярна), з описом u(x);
       n    - кількість вузлів (1 < n < 6).
       Вихідні параметри:
       err      - код закінчення:
                  0 - перевірка вхідних даних пройшла успішно;
                 >0 - при перевірці вхідних даних знайдена помилка і
                      виконання функції аварійно закінчено;
       s    - значення інтегралу.
    '''
    if n < 2 or n > 5:
        print('i_laguerre: Значення n має бути {2,3,4,5} !')
        return 1, None

    # в залежності від n визначаємо вузли і вагові коефіцієнти :
    if n == 2:
        X = np.array([0.585786, 3.414214])
        C = np.array([0.853553, 0.146447])
    elif n == 3:
        X = np.array([0.415775, 2.294280, 6.289945])
        C = np.array([0.711093, 0.278518, 0.0103893])
    elif n == 4:
        X = np.array([0.322548, 1.745761, 4.536620, 9.395071])
        C = np.array([0.603154, 0.357419, 0.0388879, 0.000539295])
    elif n == 5:
        X = np.array([0.263560, 1.413403, 3.596426, 7.085810, 12.640801])
        C = np.array([0.521756, 0.398667, 0.0759424, 0.00361176,
2.33700e-5])

    # наближене значення інтеграла :
    err = 0
    s = 0.0
    for i in range(n):
        s += C[i] * u(X[i])
    return err, s
```

```
def i_hermite(u, n):
    '''Знаходження за квадратурною формулою Ерміта інтеграла.
            ∞
        s =∫ρ(x)*u(x)dx, ρ(x)=exp(-x^2).
           -∞
       Вхідні параметри:
       u    - функція користувача (скалярна), з описом u(x);
       n    - кількість вузлів (1 < n < 6).
       Вихідні параметри:
       err      - код закінчення:
                  0 - перевірка вхідних даних пройшла успішно;
                 >0 - при перевірці вхідних даних знайдена помилка і
                      виконання функції аварійно закінчено;
       s    - значення інтеграла.
    '''
    if n < 2 or n > 5:
        print('i_hermite: Значення n має бути {2,3,4,5} !')
        return 1, None

    # в залежності від n визначаємо вузли і вагові коефіцієнти :
    X = np.zeros(n)
    C = np.ones(n)
    if n == 2:
        C[1] = 0.886227
        X[1] = 0.707107
    elif n == 3:
        C[2] = 0.295409
        X[2] = 1.224745
        C[1] = 1.181636
    elif n == 4:
        C[3] = 0.0813128
        X[3] = 1.650680
        C[2] = 0.804914
        X[2] = 0.524648
    elif n == 5:
        C[4] = 0.0199532
        X[4] = 2.020183
        C[3] = 0.393619
        X[3] = 0.958572
        C[2] = 0.945309
    for i in range(n // 2):
        j = -1 - i
        X[i] = -X[j]
        C[i] = C[j]

    # наближене значення інтеграла :
    err = 0
    s = 0.0
    for i in range(n):
        s += C[i] * u(X[i])
    return err, s


def simpson(F, h_X):
    '''метод Сімпсона для дискретної функції F обчислення визначеного
інтеграла.
```

```
        b
    s =∫f(x)dx.
        a
   Вхідні параметри:
   F   - вектор (дискретна функція);
   h_X - якщо це скалярна величина {float,int},
           то це рівномірний крок сітки;
         якщо це вектор (ndarray з елементами типу float|int),
           то це нерівномірна сітка.
   Вихідні параметри:
   s   - значення інтеграла.
'''
from copy import deepcopy
from util_lne import put_arr
n = len(F)
if n < 3 or n % 2 == 0:
    s = 'Кількість елементів F має бути непарне і > 2'
    print('simpson: ' + s)
    return 1, None
if isinstance(h_X, float) or isinstance(h_X, int):
    n1 = n - 1
    s = F[0] + F[n1] + 4 * sum(F[1:n1:2]) + 2 * sum(F[2:n1 - 1:2])
    return h_X / 3 * s

elif isinstance(h_X, np.ndarray):
    if isinstance(h_X[0], float) or isinstance(h_X[0], int):
        if not is_order(h_X):
            s = 'Вузли сітки не впорядковані або є кратні !'
            print('simpson: ' + s)
            return 2, None

        h = np.diff(h_X)
        o = deepcopy(h[0:-1])
        s = True
        for i, e in enumerate(o):
            o[i] = h[i + 1] / e
            if s: s = 0.5 <= o[i] <= 2
        if not s:
            put_arr("Вектор o", o, "%5.2f", 12)
            s = 'Умова стійкості для сітки не виконується!'
            print('simpson: ' + s)
        s = 0.0
        for i in range(1, n, 2):
            k = i - 1
            z = o[k]
            e = 1 + z
            z1 = 1 / z
            s += h[k] * e * ((2 - z) * F[k] + e * e * z1 * F[i] +
                             (2 - z1) * F[i + 1])
        return s / 6

    else:
        s = 'Елементи сітки h_X мають бути {float,int} !'
        print('simpson: ' + s)
        return 3, None

else:
    s = 'Параметр h_X не належить {float,int,np.ndarray} !'
```

```
        print('simpson: ' + s)
        return 4, None
```

**Програмний код Python головного модуля *main_num_integr* для тестування *num_integr*:**

```
from num_integr import *
from math import sin, cos, log, exp


def f(x):
    return 2 * sin(x) - log(x) + exp(2 * x)


def Ff(x):
    return -2 * cos(x) - x * (log(x) - 1) + 0.5 * exp(2 * x)


F = np.vectorize(f)

a = 0.2
b = 1.0
I = Ff(b) - Ff(a)
print("Значення ∫ на [%7.3f,%7.3f]" % (a, b))
print("Точне      = %16.10e" % I)
eps = 1e-9
print("Числове інтегрування з точністю eps = %16.10e" % eps)
It = i_trap(a, b, f, eps=1e-9)
print("м.Трапецій = %16.10e  |It-I|= %16.10e" % (It, abs(It - I)))
Is = i_simp(a, b, f, eps=1e-9)
print("м.Сімпсона = %16.10e  |Is-I|= %16.10e" % (Is, abs(Is - I)))

print("\nВикористання функції np.trapz() - м.Трапецій :")
x = np.linspace(a, b, 50)
y = F(x)
h = x[1] - x[0]
It_h = np.trapz(y, dx=h)  # рівномірний крок
print("рівномірний крок = %10.6f ∫= %16.10e  |It_h-I|= %16.10e"
      % (h, It_h, abs(It_h - I)))
z = (b + a) / 3
# X - "горизонтальне" об'єднання 3-х масивів з різними кроками
X = np.hstack((np.linspace(a, z, 30),
               np.linspace(z + 0.01, 2 * z, 12),
               np.linspace(2 * z + 0.01, b, 8)))
# print(X)
Y = F(X)
It_x = np.trapz(Y, X)  # нерівномірний крок
st = "для f(x), обчисленої на сітці ∫= %16.10e  |It_x-I|= %16.10e"
print(st % (It_x, abs(It_x - I)))
err, It_x = i_c_rectangle(X, f)
st = "\nм.Центральних прямокутників   ∫= %16.10e  |It_x-I|= %16.10e"
print(st % (It_x, abs(It_x - I)))

print("\n  КВАДРАТУРНІ  ФОРМУЛИ  ГАУССА-КРІСТОФФЕЛЯ")
a = 0.2
b = 1.0
I = Ff(b) - Ff(a)
```

```
print("Значення ∫ на [%7.3f,%7.3f]" % (a, b))
print("Точне       = %16.10e" % I)
for n in range(1, 8):
    err, It = i_gauss(a, b, f, n)
    print("Гаусса n=%i ∫= %16.10e  |It-I|= %16.10e"
          % (n, It, abs(It - I)))

print()


def T3_2(x):  # многочлен Чебишова T(3,x)^2
    t = x * (4 * x * x - 3)  # T(3,x)
    return t * t


a = -1.0
b = 1.0
I = np.pi / 2
print("Значення ∫ на [%7.3f,%7.3f]" % (a, b))
print("Точне                  = %16.10e" % I)
for n in range(1, 11):
    err, It = i_g_cheb(a, b, T3_2, n)
    st = "Гаусса-Чебишова n=%2i ∫= %16.10e  |It-I|= %16.10e"
    print(st % (n, It, abs(It - I)))

print()
del x, y, a, b, X, Y, z, h, eps


def u(t):
    '''для гамма-функції Γ(x)=∫(exp(-t)*u(t)dt), u(t)=t**(x-1)
    '''
    return t ** (x - 1)


x = 4.7
I = 15.43
print("Гамма-функція Γ(x)=∫(0,∞,exp(-t)*u(t)dt), u(t)=t**(x-1)")
print("Таблиця Γ(%7.3f)= %16.10e" % (x, I))
for n in range(2, 6):
    err, It = i_laguerre(u, n)
    print("КФ  Лагерра n=%2i ∫= %16.10e  |It-I|= %16.10e"
          % (n, It, abs(It - I)))

print()


def e(t):
    return 1.0


I = np.sqrt(np.pi)
print("Функція похибки erf(x)=2/sqrt(pi)*∫(0,x,exp(-t**2)dt)")
print("sqrt(pi)*erf(∞) = %16.10e" % I)
for n in range(2, 6):
    err, It = i_hermite(e, n)
    st = "КФ Ерміта n=%2i ∫= %16.10e  |It-I|= %16.10e"
    print(st % (n, It, abs(It - I)))
```

## Результат виконання ***main_num_integr*:**

```
Значення ∫ на [  0.200,  1.000]
Точне      = 4.3062566621e+00
Числове інтегрування з точністю eps = 1.0000000000e-09
м.Трапецій = 4.3062566628e+00  |It-I|= 7.4056494270e-10
м.Сімпсона = 4.3062566623e+00  |Is-I|= 1.5680345911e-10

Використання функції np.trapz() - м.Трапецій :
рівномірний крок =   0.016327 ∫= 4.3065879379e+00  |It_h-I|= 3.3127577076e-04
для f(x), обчисленої на сітці ∫= 4.3071979573e+00  |It_x-I|= 9.4129521276e-04

м.Центральних прямокутників   ∫= 4.3057860593e+00  |It_x-I|= 4.7060283093e-04

  КВАДРАТУРНІ  ФОРМУЛИ  ГАУССА-КРІСТОФФЕЛЯ
Значення ∫ на [  0.200,  1.000]
Точне      = 4.3062566621e+00
Гаусса n=1 ∫= 3.9681819946e+00  |It-I|= 3.3807466747e-01
Гаусса n=2 ∫= 4.2967238632e+00  |It-I|= 9.5327988639e-03
Гаусса n=3 ∫= 4.3057056378e+00  |It-I|= 5.5102431997e-04
Гаусса n=4 ∫= 4.3061980651e+00  |It-I|= 5.8596987163e-05
Гаусса n=5 ∫= 4.3062497807e+00  |It-I|= 6.8814405942e-06
Гаусса n=6 ∫= 4.3062558214e+00  |It-I|= 8.4073233531e-07
Гаусса n=7 ∫= 4.3062565566e+00  |It-I|= 1.0550885499e-07

Значення ∫ на [ -1.000,  1.000]
Точне              = 1.5707963268e+00
Гаусса-Чебишова n= 1 ∫= 0.0000000000e+00  |It-I|= 1.5707963268e+00
Гаусса-Чебишова n= 2 ∫= 1.5707963268e+00  |It-I|= 0.0000000000e+00
Гаусса-Чебишова n= 3 ∫= 1.5489247653e-30  |It-I|= 1.5707963268e+00
Гаусса-Чебишова n= 4 ∫= 1.5707963268e+00  |It-I|= 4.4408920985e-16
Гаусса-Чебишова n= 5 ∫= 1.5707963268e+00  |It-I|= 2.2204460493e-16
Гаусса-Чебишова n= 6 ∫= 1.5707963268e+00  |It-I|= 0.0000000000e+00
Гаусса-Чебишова n= 7 ∫= 1.5707963268e+00  |It-I|= 4.4408920985e-16
Гаусса-Чебишова n= 8 ∫= 1.5707963268e+00  |It-I|= 0.0000000000e+00
Гаусса-Чебишова n= 9 ∫= 1.5707963268e+00  |It-I|= 4.4408920985e-16
Гаусса-Чебишова n=10 ∫= 1.5707963268e+00  |It-I|= 2.2204460493e-16

Гамма-функція Г(x)=∫(0,∞,exp(-t)*u(t)dt), u(t)=t**(x-1)
Таблиця Г(  4.700)= 1.5430000000e+01
КФ  Лагерра n= 2 ∫= 1.3885531070e+01  |It-I|= 1.5444689295e+00
КФ  Лагерра n= 3 ∫= 1.5409329804e+01  |It-I|= 2.0670196461e-02
КФ  Лагерра n= 4 ∫= 1.5428251538e+01  |It-I|= 1.7484615485e-03
КФ  Лагерра n= 5 ∫= 1.5430585860e+01  |It-I|= 5.8586011091e-04

Функція похибки erf(x)=2/sqrt(pi)*∫(0,x,exp(-t**2)dt)
sqrt(pi)*erf(∞) = 1.7724538509e+00
КФ Ерміта n= 2 ∫= 1.7724540000e+00  |It-I|= 1.4909448409e-07
КФ Ерміта n= 3 ∫= 1.7724540000e+00  |It-I|= 1.4909448409e-07
КФ Ерміта n= 4 ∫= 1.7724536000e+00  |It-I|= 2.5090551592e-07
КФ Ерміта n= 5 ∫= 1.7724534000e+00  |It-I|= 4.5090551604e-07
```

## VII. Модуль числових методів розв'язування задачі Коші для ЗДР.

**В модуль *num_odePrC.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *euler(F,Xotr,Y0,h)* | явний метод Ейлера O(*h*) розв'язування ЗК для системи ЗДР (крок *h* – постійний) |
| *symmetry(f,Xotr,Y0,h,df, eps=1e-8,mki=50)* | симетрична неявна схема O(*h*^2) розв'язування ЗК для 1-го ЗДР (крок *h* – постійний) |
| *r_kutta2(F,Xotr,Y0,h,gam=0.5)* | 2-х стадійний метод Рунге-Кутта O(*h*^2) розв'язування ЗК для системи ЗДР (крок *h* – постійний) |
| *k_mersona(F,Xotr,Y0,h,old, eps=1e-8,kdiv=20)* | метод Кутта-Мерсона O(*h*^4) розв'язування ЗК для системи ЗДР (автоматичний вибір кроку) |

**Програмний код Python функцій модуля :**

```
""" Модуль числових методів розв'язування задачі Коші для ЗДР :

          ┌ U'(x) = F(x,U(x)), X0 < x <= Xend;
          └ U(X0) = U0.
"""

import numpy as np
from nle_im import njut_1


def euler(F, Xotr, Y0, h):
    '''Явний метод Ейлера O(h) розв'язування ЗК для системи ЗДР.

       Вхідні параметри:
       F    - ім'я функції користувача з описом F(x,U);
       Xotr - вектор, виду:
              [X0, Xend] - відрізок, на якому шукаємо розв'язок;
              [X0, X1,...,Xend] - точки в яких отримуємо розв'язок;
       Y0   - вектор початкових значень задачі Коші для ЗДР;
       h    - крок знаходження розв'язку.
       Вихідні параметри:
       X    - сітка розв'язку;
       Y    - матриця розв'язку (стовпці Y[:,i]=Ui(X)).
    '''
    N = len(Xotr)
    if N == 2:
        X = np.arange(Xotr[0], Xotr[1] + 0.5 * h, h)
    else:
        X = Xotr[:]

    nx = len(X)
    ny = len(Y0)
    Y = np.zeros((nx, ny))

    Y[0, :] = Y0[:]
    xe = X[0]
    ho = h
    for i in range(1, nx):
        x = xe
        xe = X[i]
```

```
            h = ho
            y = Y[i - 1, :]
            while x < xe:
                xm = x
                x += h
                if x > xe:
                    h = xe - xm
                    x = xe
                y = y + h * F(xm, y)
            Y[i, :] = y

    return X, Y


def symmetry(f, Xotr, Y0, h, df, eps=1e-8, mki=50):
    '''Симетрична неявна схема O(h^2) розв'язування ЗК для 1-го ЗДР.

       Вхідні параметри:
       f    - ім'я (скалярної) функції користувача з описом f(x,u);
       Xotr - вектор, виду:
              [X0, Xend] - відрізок, на якому шукаємо розв'язок;
              [X0, X1,...,Xend] - точки в яких отримуємо розв'язок;
       Y0   - початкове значення задачі Коші для ЗДР;
       h    - крок знаходження розв'язку;
       df   - ім'я (скалярної) функції користувача з описом f'(x,u);
       eps  - точність (0<e<1);
       mki  - максимальна кількість ітерацій.
       Вихідні параметри:
       X    - сітка розв'язку;
       Y    - вектор розв'язку (Y=u(X)).
    '''

    def fi(t):
        '''Опис функції fi(t)=t-0.5hf(x(n+1),t)-(y(n)+0.5hf(x(n),y(n))).

        '''
        z = t - h2 * f(x, t) - e
        return z

    def dfi(t):
        '''Опис функції dfi(t)=1-0.5hf'(x(n+1),t)

        '''
        z = 1 - h2 * df(x, t)
        return z

    if eps <= 0 or eps >= 1: eps = 1e-8
    if mki <= 1: mki = 50

    N = len(Xotr)
    if N == 2:
        X = np.arange(Xotr[0], Xotr[1] + 0.5 * h, h)
    else:
        X = Xotr[:]
    nx = len(X)
    Y = np.zeros(nx)

    Y[0] = Y0
```

```
    xe = X[0]
    ho = h
    for i in range(1, nx):
        x = xe
        xe = X[i]
        h = ho
        y = Y[i - 1]
        while x < xe:
            xm = x
            x += h
            if x > xe:
                h = xe - xm
                x = xe
            h2 = 0.5 * h
            e = y + h2 * f(xm, y)
            y, err, k = njut_1(y, fi, dfi, eps, mki)
        Y[i] = y

    return X, Y


def r_kutta2(F, Xotr, Y0, h, gam=0.5):
    '''2-х стадійний метод Рунге-Кутта O(h^2) розв'язування ЗК для
системи ЗДР.

       Вхідні параметри:
       F    - ім'я функції користувача з описом F(x,U)
       Xotr - вектор, виду:
              [X0, Xend] - відрізок, на якому шукаємо розв'язок;
              [X0, X1,...,Xend] - точки в яких отримуємо розв'язок;
       Y0   - вектор початкових значень задачі Коші для ЗДР;
       h    - крок знаходження розв'язку;
       gam  - параметр сімейства 2-х стадійний метод Рунге-Кутта.
       Вихідні параметри:
       X    - сітка розв'язку;
       Y    - матриця розв'язку (стовпці Y[:,i]=Ui(X)).
    '''
    N = len(Xotr)
    if N == 2:
        X = np.arange(Xotr[0], Xotr[1] + 0.5 * h, h)
    else:
        X = Xotr[:]

    nx = len(X)
    ny = len(Y0)
    Y = np.zeros((nx, ny))
    if gam <= 0 or gam > 1:
        print("r_kutta2: Параметр gam має належати (0, 1] !",
              "\nРозрахунок виконано з gam=0.5")
        gam = 0.5

    A1 = 1 - gam
    A2 = gam
    te = 0.5 / gam

    Y[0, :] = Y0[:]
    xe = X[0]
    ho = h
```

```
    for i in range(1, nx):
        x = xe
        xe = X[i]
        h = ho
        y = Y[i - 1, :]
        while x < xe:
            xm = x
            x += h
            if x > xe:
                h = xe - xm
                x = xe
            K1 = F(xm, y)
            tet = te * h
            K2 = F(xm + tet, y + tet * K1)
            y = y + h * (A1 * K1 + A2 * K2)
        Y[i, :] = y

    return X, Y


def k_mersona(F, Xotr, Y0, hold, eps=1e-8, kdiv=20):
    '''метод Кутта-Мерсона O(h^4) розв'язування ЗК для системи ЗДР.

       Метод Кутта-Мерсона 4-го порядку з автоматичним вибором
       кроку розв'язування задачі Коші для системи ЗДР.

       Вхідні параметри:
       F    - ім'я функції користувача з описом F(x,U);
       Xotr - вектор, виду:
              [X0, Xend] - відрізок, на якому шукаємо розв'язок;
              [X0, X1,...,Xend] - точки в яких отримуємо розв'язок;
       Y0   - вектор початкових значень задачі Коші для ЗДР;
       hold - початковий крок інтегрування;
       eps  - бажана точність знаходження розв'язку (0<eps<1);
       kdiv - максимальна кількість поділів кроку при автоматичному
                 виборі кроку.
       Вихідні параметри:
       err  - код закінчення алгоритму :
              0 - нормальне завершення;
              1 - кількість поділів кроку при автоматичному виборі
                  кроку перевищила задану кількість kdiv.
       X    - сітка розв'язку;
       Y    - матриця розв'язку (стовпці Y[:,i]=Ui(X)).
    '''

    if eps <= 0 or eps >= 1: eps = 1e-6

    N = len(Xotr)
    if N == 2:
        X = np.arange(Xotr[0], Xotr[1] + 0.1 * hold, hold)
    else:
        X = Xotr[:]

    nx = len(X)
    ny = len(Y0)
    Y = np.zeros((nx, ny))

    Y[0, :] = Y0[:]
```

```
xe = X[0]
ho = hold
err = 0
for i in range(1, nx):
    x = xe
    xe = X[i]
    h = ho
    y = Y[i - 1, :]
    h3 = h / 3
    while x < xe:
        xm = x
        x += h
        if x > xe:
            h = xe - xm
            x = xe
            h3 = h / 3
        kd = 0  # кількість поділів кроку
        while True:
            K1 = h3 * F(xm, y)
            z = xm + h3
            K2 = h3 * F(z, y + K1)
            K3 = h3 * F(z, y + 0.5 * (K1 + K2))
            K4 = h3 * F(xm + 0.5 * h, y + 0.375 * (K1 + 3 * K3))
            w = y + 1.5 * K1 - 4.5 * K3 + 6 * K4
            K5 = h3 * F(x, w)
            u = y + 0.5 * (K1 + 4 * K4 + K5)
            fl = False
            fdub = True
            for k in range(ny):
                d = 0.2 * abs(w[k] - u[k])  # похибка обчислень
                z = abs(u[k])
                g = eps if z <= 1 else eps * z
                if fdub:
                    fdub = 32 * d <= g
                if d > g:
                    fl = True
                    fdub = False
                    break
            if fl:
                # точність не досягнута, зменшення кроку
                h *= 0.5
                h3 = h / 3
                kd += 1
                if kd > kdiv:
                    err = 1
                    break
            else:
                y = u
                break  # в x знайденно розв'язок
        if err == 1: break
        if fdub:  # можливо удвоїть крок ?
            z = 2 * ho
            if hold >= z:  # але не більше заданого !
                ho = z
        Y[i, :] = y
    if err == 1: break

return err, X, Y
```

**Програмний код Python головного модуля *main_num_odePrC* для тестування *num_odePrC*:**

```
import matplotlib.pyplot as plt
from util_lne import put_arr, movespinesticks
from num_odePrC import *


def F1(x, U):
    dU = np.array([0.5 * x * U[0]])
    return dU


def F2(x, U):
    dU = np.array([(x - 2) * U[1],
                   U[0] - x * (x * x - 12) / 3 + 1])
    return dU


def f1(x, u):
    du = 0.5 * x * u
    return du


def df1(x, u):
    du = 0.5 * x
    return du


Xotr = np.array([0, 1, 1.304, 1.5, 2])
Y0 = np.array([1])
h = 1e-3

X, Y = euler(F1, Xotr, Y0, h)
put_arr("  X =", X, "%6.3f", 5)
print("Явний м.Ейлера O(h) для 1-го рівняння")
for i in range(len(Y0)):
    put_arr("Y%i=" % i, Y[:, i], "%9.5f", 5)

Y0 = 1.0
X, Y = symmetry(f1, Xotr, Y0, h, df1)
print("Симметрична неявна схема O(h^2) для 1-го рівняння")
put_arr("  Y =", Y, "%9.5f", 5)

Y = np.exp(0.25 * X ** 2)
put_arr("Yточ=", Y, "%9.5f", 5)

Xotr = np.array([0, .3, .6, .8, 1, 1.4, 2, 2.5, 2.8, 3])
Y0 = np.array([0, 2])
h = 1e-4

put_arr("  X =", X, "%6.3f", 5)
print("Явний м.Ейлера O(h) для 2-х рівнянь")
X, Y = euler(F2, Xotr, Y0, h)
for i in range(len(Y0)):
    put_arr("Y%i=" % i, Y[:, i], "%9.5f", 5)
```

```
gam = 0.5
st = "2-х стадійний gam=%.3f м.Рунге-Кутта O(h^2) для 2-х рівнянь"
print(st % gam)
X, Y = r_kutta2(F2, Xotr, Y0, h, gam)
for i in range(len(Y0)):
    put_arr("Y%i=" % i, Y[:, i], "%9.5f", 5)

print("м.Кутта-Мерсона O(h^4) розв'язування ЗК для 2-х рівнянь")
err, X, Y = k_mersona(F2, Xotr, Y0, h)
if err == 0:
    for i in range(len(Y0)):
        put_arr("Y%i=" % i, Y[:, i], "%9.5f", 5)


def LV(x, U):  # рівняння Лотки-Вольтерри
    dU = np.array([U[0] * (1 - alf * U[1]),
                   U[1] * (bet * U[0] - 1)])
    return dU


# Модель "хижак-жертва"
alf = 1
bet = 2.5
Xotr = np.array([0, 25])
Y0 = np.array([1, 1])
h = 1e-2
err, X, Y = k_mersona(LV, Xotr, Y0, h)
if err == 0:
    plt.plot(X, Y[:, 0], X, Y[:, 1])
    plt.axis('equal')
    plt.grid(True)
    plt.title("alf=1, bet=2.5")
    plt.legend(["u(x)", "v(x)"], loc="best")
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("u,v")
    plt.show()

    # Фазові траекторії
    plt.plot(Y[:, 0], Y[:, 1], "r")
    alf = 2.5
    bet = 1
    err, X, Y = k_mersona(LV, Xotr, Y0, h)
    if err == 0:
        plt.plot(Y[:, 0], Y[:, 1], "b")
        plt.axis('equal')
        plt.grid(True)
        plt.legend(["alf=1, bet=2.5", "alf=2.5, bet=1"], loc="best")
        movespinesticks()
        plt.xlabel("u")
        plt.ylabel("v")
    plt.show()
```

**Результат виконання *main_num_odePrC*:**

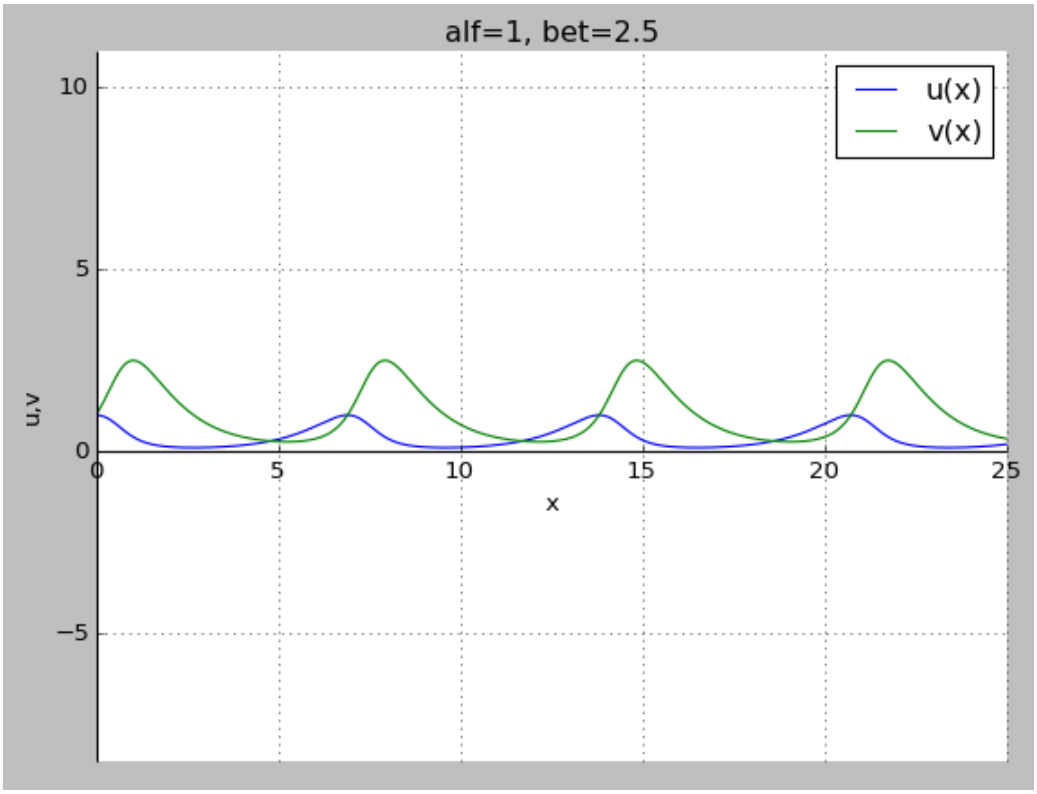
```
X =
0.000 1.000 1.304 1.500 2.000
```

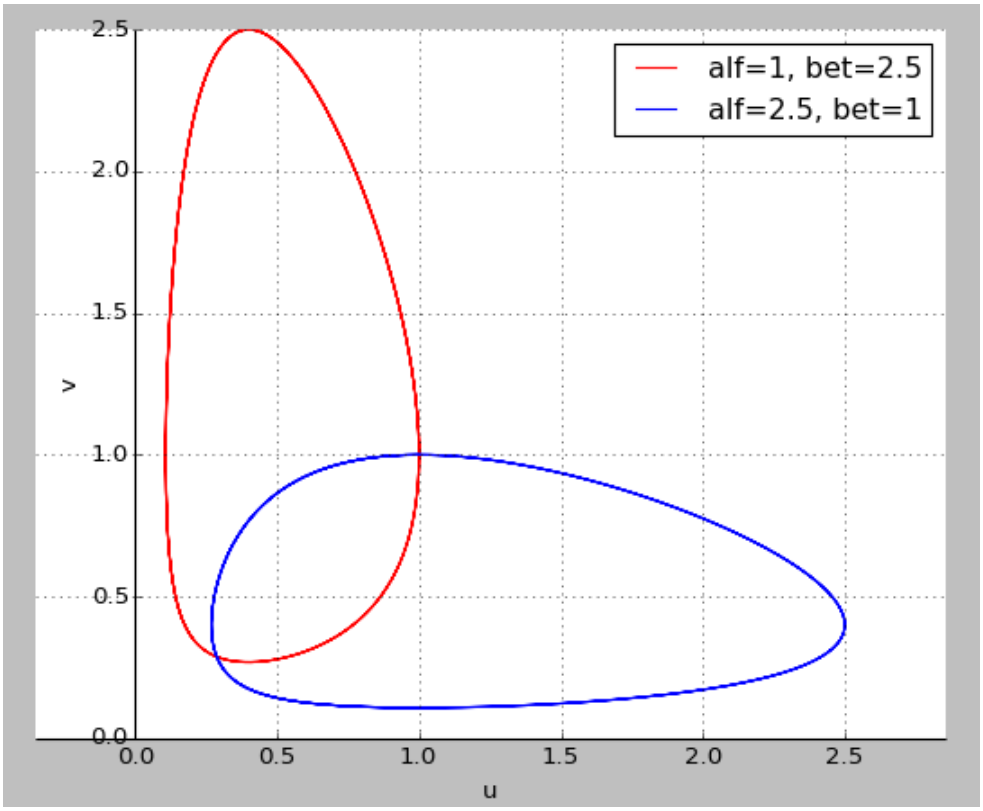
Явний м.Ейлера O(h) для 1-го рівняння
Y0=
 1.00000  1.28365  1.52911  1.75415  2.71602
Симметрична неявна схема O(h^2) для 1-го рівняння
 Y =
 1.00000  1.28403  1.52975  1.75505  2.71828
Yточ=
 1.00000  1.28403  1.52975  1.75505  2.71828

 X =
 0.000 1.000 1.304 1.500 2.000
Явний м.Ейлера O(h) для 2-х рівнянь
Y0=
 0.00000 -1.19100 -2.32802 -3.02936 -3.66671
-4.68542 -5.33351 -4.79198 -3.88312 -3.00058
Y1=
 2.00000  2.30000  2.60000  2.79999  2.99998
 3.39996  3.99988  4.49976  4.79965  4.99955

2-х стадійний gam=0.500 м.Рунге-Кутта O(h^2) для 2-х рівнянь
Y0=
 0.00000 -1.19100 -2.32800 -3.02933 -3.66667
-4.68533 -5.33333 -4.79167 -3.88267 -3.00000
Y1=
 2.00000  2.30000  2.60000  2.80000  3.00000
 3.40000  4.00000  4.50000  4.80000  5.00000

м.Кутта-Мерсона O(h^4) розв'язування ЗК для 2-х рівнянь
Y0=
 0.00000 -1.19100 -2.32800 -3.02933 -3.66667
-4.68533 -5.33333 -4.79167 -3.88267 -3.00000
Y1=
 2.00000  2.30000  2.60000  2.80000  3.00000
 3.40000  4.00000  4.50000  4.80000  5.00000




alf=1, bet=2.5
alf=2.5, bet=1
2.5
2.0
1.5
1.0
0.5
0.0
v
0.0
0.5
1.0
1.5
2.0
2.5
u

## VIII. Модуль різницевих методів розв'язування лінійних крайових задач для ЗДР другого порядку.

**В модуль *num_odeBvP.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *m_grid2(p,f,a,b,D,n)* | метод сіток O($h$^2) розв'язування ЛКЗ для ЗДР 2-го порядку |
| *m_grid4(p,f,a,b,D,n,d1f,d2f)* | метод сіток O($h$^4) розв'язування ЛКЗ для ЗДР 2-го порядку |

**Програмний код Python функцій модуля :**

```
"""Модуль різницевих методів розв'язування ЛКЗ для ЗДР 2 порядку:

    Розв'язати лінійне диференціальне рівняння
      u"(x) - p(x)*u(x) = -f(x),
         p(x) >= 0,  x є (a,b),
    для якого виконуються крайові умови
      -D(1)*u'(a) + D(2)*u(a) = D(3),
       D(4)*u'(b) + D(5)*u(b) = D(6).
    різницевим методом 2-го порядку точності.
    Для параметрів D повинні виконуватись умови :
     1) abs(D(1))+abs(D(2))≠0 & abs(D(4))+abs(D(5))≠0
     2) D(2)=D(5) & p(x)≠0.
"""

import numpy as np
from slae_em import progm


def m_grid2(p, f, a, b, D, n):
    '''Метод сіток O(h^2) розв'язування ЛКЗ для ЗДР 2-го порядку.

       Вхідні параметри:
       p  - ім'я (векторизованої) ф-ї користувача з описом p(x);
       f  - ім'я (векторизованої) ф-ї користувача з описом f(x);
       a  - ліва границя відрізку [a,b];
       b  - права границя відрізку [a,b];
       D  - масив зі значеннями коефіцієнтів крайових умов;
       n  - число точок сітки різницевої схеми.
       Вихідні параметри:
       err- код закінчення:
            0 - перевірка вхідних даних пройшла успішно;
           >0 - при перевірці вхідних даних знайдена помилка і
                виконання функції аварійно закінчено;
       X  - сітка розв'язку;
       Y  - вектор розв'язку (Y=u(X)).
    '''

    if a >= b:
        print("m_grid2: Неправильно задані значення a, b")
        return 1, None, None
    if len(D) < 6:
        print("m_grid2: Кількість значень параметрів D < 6")
        return 2, None, None
    if abs(D[0]) + abs(D[1]) == 0 or abs(D[3]) + abs(D[4]) == 0:
        print("m_grid2: Для параметрів D не виконується умова 1)")
        return 3, None, None
```

```
    if n < 2:
        print("m_grid2: Помилкова кількість вузлів n < 2")
        return 4, None, None

    X = np.linspace(a, b, n)  # формування сітки

    C = p(X)
    if not np.all(C >= 0):
        print("m_grid2: Не виконується умова p(X) >= 0 !")
        return 5, None, None
    if D[1] == D[4] and D[4] == 0 and not np.all(C == 0):
        print("m_grid2: Для параметрів D не виконується умова 2)")
        return 6, None, None

    h = X[1] - X[0]
    n1 = n - 1
    hh = h * h
    h2 = 2 * h

    C = hh * C + 2  # формування коефіцієнтів СЛАР
    C[0] = D[0] * C[0] + h2 * D[1]
    C[n1] = D[3] * C[n1] + h2 * D[4]
    A = np.ones(n)
    A[0] = 0
    A[n1] = 2 * D[3]
    B = np.ones(n)
    B[0] = 2 * D[0]
    B[n1] = 0
    F = hh * f(X)
    F[0] = D[0] * F[0] + h2 * D[2]
    F[n1] = D[3] * F[n1] + h2 * D[5]

    Y = progm(A, C, B, F)  # розв'язок СЛАР

    return 0, X, Y


def m_grid4(p, f, a, b, D, n, d1f, d2f):
    '''Метод сіток O(h^4) розв’язування ЛКЗ для ЗДР 2-го порядку.

       Вхідні параметри:
       p  - коефіцієнт в рівнянні;
       f  - ім'я (векторизованої) ф-ї користувача з описом f(x);
       a  - ліва границя відрізку [a,b];
       b  - права границя відрізку [a,b];
       D  - масив зі значеннями коефіцієнтів крайових умов;
       n  - число точок сітки різницевої схеми;
       d1f- ім'я (скалярної) ф-ї користувача з описом f'(x);
       d2f- ім'я (векторизованої) ф-ї користувача з описом f"(x).
       Вихідні параметри:
       err- код закінчення:
            0 - перевірка вхідних даних пройшла успішно;
           >0 - при перевірці вхідних даних знайдена помилка і
                виконання функції аварійно закінчено;
       X  - сітка розв'язку;
       Y  - вектор розв'язку (Y=u(X)).
    '''
```

```
if a >= b:
    print("m_grid4: Неправильно задані значення a, b")
    return 1, None, None
if len(D) < 6:
    print("m_grid4: Кількість значень параметрів D < 6")
    return 2, None, None
if abs(D[0]) + abs(D[1]) == 0 or abs(D[3]) + abs(D[4]) == 0:
    print("m_grid4: Для параметрів D не виконується умова 1)")
    return 3, None, None
if n < 2:
    print("m_grid4: Помилкова кількість вузлів n < 2")
    return 4, None, None
if p < 0:
    print("m_grid4: Не виконується умова p(X) >= 0")
    return 5, None, None
if D[1] == D[4] and D[4] == 0 and p == 0:
    print("m_grid4: Для параметрів D не виконується умова 2)")
    return 6, None, None

X = np.linspace(a, b, n)  # формування сітки
#  допоміжні змінні
h = X[1] - X[0]
n1 = n - 1
hh = h * h
h2 = 0.5 * h
hh12 = hh / 12
hh6 = hh / 6
h4 = 0.25 * h
pt = 1 + p * hh12
hp = h2 * pt
z = 2 + p * pt * hh
ptt = 1 + p * hh6
php = p * hp

# формування коефіцієнтів
F = f(X)
fa = F[0]
fb = F[n1]
C = d2f(X)
F = hh * (pt * F + hh12 * C)
F[0] = D[0] * (hp * fa + hh6 * (h4 * C[0] + d1f(a))) + D[2] * ptt
F[n1] = D[3] * (hp * fb + hh6 * (h4 * C[n1] - d1f(b))) + D[5] * ptt
A = np.ones(n)
A[0] = 0
A[n1] = D[3] / h
B = np.ones(n)
B[0] = D[0] / h
B[n1] = 0
C = z * np.ones(n)
C[0] = B[0] + D[0] * php + D[1] * ptt
C[n1] = A[n1] + D[3] * php + D[4] * ptt

Y = progm(A, C, B, F)  # розв'язок СЛАР

return 0, X, Y
```

**Програмний код Python головного модуля *main_num_odeBvP* для тестування *num_odeBvP*:**

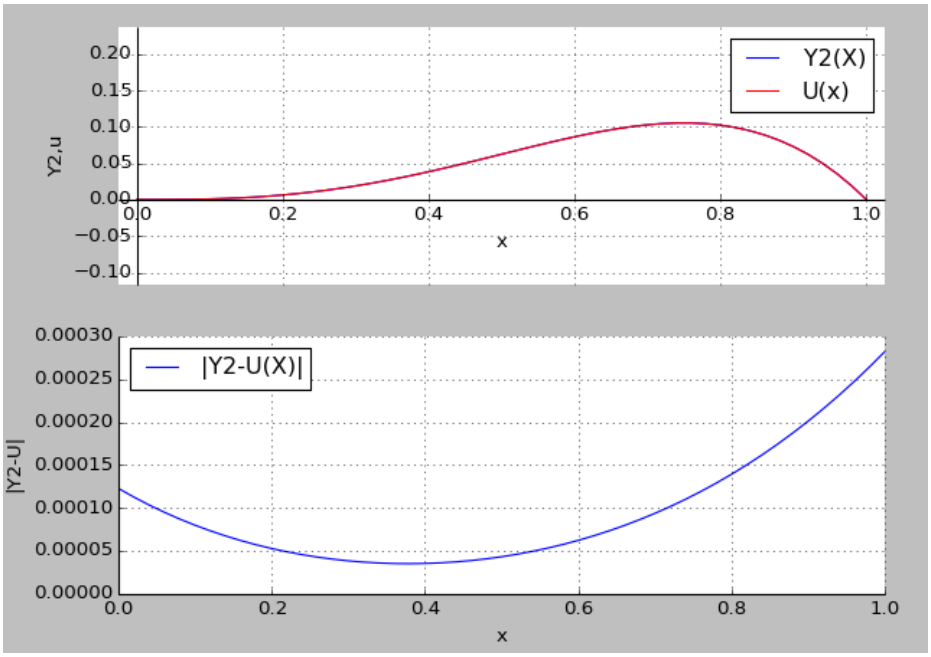
```
import matplotlib.pyplot as plt
from util_lne import movespinesticks
from num_odeBvP import *

print("Розв'язання модельної задачі\n",
      '    U"(x) - p(x)*U(x) = -f(x),\n',
      '      p(x) >= 0,  x Є (0,1);\n',
      "      U'(0) + U(0) =  0,\n",
      "      U'(1) + U(1) = -1.\n",
      "Для p(x)=5(2-x) і f(x)=x(6(2x-1)+5(1-x)(2-x)x^2)\n",
      "розв'язок буде U(x)=(1-x)x^3.")


def pf(x):
    return 5 * (2 - x)


def ff(x):
    return x * (6 * (2 * x - 1) + 5 * (1 - x) * (2 - x) * x * x)


def u(x):
    """Модельний розв'язок."""
    t = x * x
    return (1 - x) * x * t


Pv = np.vectorize(pf)
Fv = np.vectorize(ff)
Uv = np.vectorize(u)

a = 0.0
b = 1.0
D = np.array([-1, 1, 0.0, 1, 1, -1])
n = 51

err, X, Y2 = m_grid2(Pv, Fv, a, b, D, n)
if err == 0:  # Якщо знайшли наближений розв'язок
    U = Uv(X)  # Модельний розв'язок
    plt.subplot(2, 1, 1)
    plt.plot(X, Y2, 'b', X, U, 'r')
    plt.axis('equal')
    plt.grid(True)
    plt.legend(["Y2(X)", "U(x)"], loc="best")
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("Y2,u")
    plt.subplot(2, 1, 2)
    Y2 = abs(Y2 - U)
    print("Крок сітки=", X[1] - X[0])
    print('Похибка для РС O(h^2) в нормі C=', np.max(Y2))
    plt.plot(X, Y2)
    plt.grid(True)
    plt.legend(["|Y2-U(X)|"], loc="best")
```

```
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("|Y2-U|")
    plt.show()

print("\nРозв'язання модельної задачі\n",
      '    U"(x) - p*U(x) = -f(x),\n',
      '      p >= 0,  x Є (0,1);\n',
      "      U'(0) + U(0) =  0,\n",
      "      U'(1) + U(1) = -1.\n",
      "Для p=5 і f(x)=x(6(2x-1)+5(1-x)x^2)\n",
      "розв'язок буде U(x)=(1-x)x^3.")


def pf(x):
    return 5.0


def ff(x):
    return x * (6 * (2 * x - 1) + 5 * (1 - x) * x * x)


def d1f(x):
    return -6 + x * (24 + x * (15 - 20 * x))


def d2f(x):
    return 24 + 30 * x * (1 - 2 * x)


Pv = np.vectorize(pf)
Fv = np.vectorize(ff)
D2fv = np.vectorize(d2f)

err, X, Y2 = m_grid2(Pv, Fv, a, b, D, n)
if err == 0:  # Якщо знайшли наближений розв'язок
    Y2 = abs(Y2 - U)
    print("Крок сітки=", X[1] - X[0])
    print('Похибка для РС O(h^2) в нормі C=', np.max(Y2))

p = 5.0
er4, X, Y4 = m_grid4(p, Fv, a, b, D, n, d1f, D2fv)
if err == 0 and er4 == 0:  # Якщо знайшли наближений розв'язок
    plt.subplot(2, 1, 1)
    plt.plot(X, Y4, 'b', X, U, 'r')
    plt.axis('equal')
    plt.grid(True)
    plt.legend(["Y4(X)", "U(x)"], loc="best")
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("Y4,u")
    plt.subplot(2, 1, 2)
    Y4 = abs(Y4 - U)
    print('Похибка для РС O(h^4) в нормі C=', np.max(Y4))
    plt.plot(X, Y2, X, 1e10 * Y4)
    plt.grid(True)
    plt.legend(["|Y2-U(X)|", "1e10*|Y4-U(X)|"], loc="best")
    movespinesticks()
```

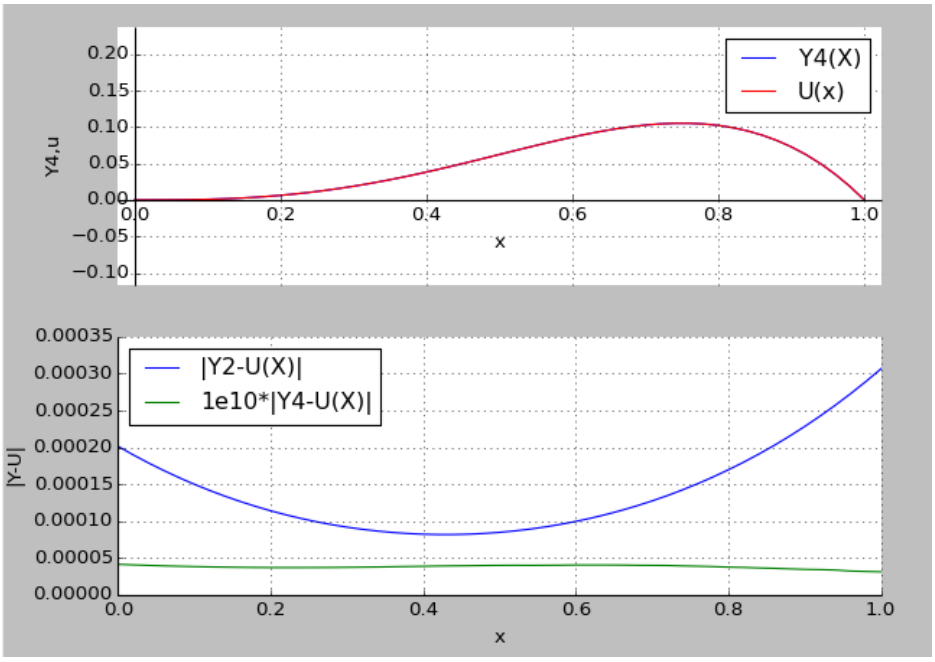
```
    plt.xlabel("x")
    plt.ylabel("|Y-U|")
    plt.show()
```

### Результат виконання ***main_num_odeBvP*** :

Розв'язання модельної задачі
U''(x) - p(x)*U(x) = -f(x),
p(x) >= 0, x Є (0,1);
U'(0) + U(0) = 0,
U'(1) + U(1) = -1.
Для p(x)=5(2-x) і f(x)=x(6(2x-1)+5(1-x)(2-x)x^2)
розв'язок буде U(x)=(1-x)x^3.
Крок сітки= 0.02
Похибка для РС O(h^2) в нормі C= 0.000282632333362




Розв'язання модельної задачі
U''(x) - p*U(x) = -f(x),
p >= 0, x Є (0,1);
U'(0) + U(0) = 0,
U'(1) + U(1) = -1.
Для p=5 і f(x)=x(6(2x-1)+5(1-x)x^2)
розв'язок буде U(x)=(1-x)x^3.
Крок сітки= 0.02
Похибка для РС O(h^2) в нормі C= 0.000306764457708
Похибка для РС O(h^4) в нормі C= 4.13179452945e-15


Y4(X)
U(x)
Y4,u
x
|Y2-U(X)|
1e10*|Y4-U(X)|
|Y-U|
x

## IX. Модуль різницевих методів розв'язування лінійних крайових задач для рівнянь гіперболічного типу першого порядку.

**В модуль *num_pdeH1.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *fdm_H1Ccp(c,f,a,b,te,ma,U0,n,m,Nm,S=3)* | метод сіток розв'язування КЗ для рівняння переносу з *c=Const*>0. *S*=1 – явна РС, *S* =2 – неявна, *S*=3 – симетрична неявна |
| *fdm_H1Cvp(c,f,X,T,ma,U0,Nm)* | явно-неявна схема розв'язування КЗ для рівняння переносу зі змінною *c(x,t)>0* і на нерівномірних сітках *X*, *T* |
| *fdm_H1Cxt(c,f,a,b,te,ma,mb,U0,n,m,Nm)* | метод сіток розв'язування КЗ для рівняння переносу зі знакозмінною *c(x,t)* : при *x=a  c(x,t)>0* і для *x=a  c(x,t)<0* |

**Програмний код Python функцій модуля :**

```
"""Модуль РМ розв'язування ЛКЗ для р-нь гіперболічного типу 1 порядку.

"""

import numpy as np
from slae_em import progn
from interp import is_order


def check_(Nm, m):
    '''Чи всі елементи Nm[j] є [0,m] і впорядковані за зростанням ?'''
    yes = len(Nm) < 1  # відсутні елементи в масиві
    if not yes:
        fl = True
        for j in Nm:
            yes = j < 0 or j > m
            if yes: break  # не є [0,m]
            if fl:
                fl = False
                i = j
            else:
                yes = i >= j
                if yes: break  # не впорядковані за зростанням
    return yes


def fdm_H1Ccp(c, f, a, b, te, ma, U0, n, m, Nm, S=3):
    '''Метод сіток розв'язування ЛКЗ для рівняння переносу.

          dU(x,t)/dt + c*dU(x,t)/dx = f(x,t),
             c > 0, (x,t) Є (a,b)*(0,T];
             U(a,t) = Ma(t), t Є [0,T];
             U(x,0) = U0(x), x Є [a,b].
       Вхідні параметри:
       c  - коефіцієнт в рівнянні (c>0);
       f  - ім'я (скалярної) ф-ї користувача з описом f(x,t);
       a  - ліва границя відрізку [a,b] (a<b);
       b  - права границя відрізку [a,b];
```

```
    te - граничне значення по часу (te>0);
    ma - ім'я (скалярної) ф-ї користувача з описом Ma(t) -
             крайова умова для x=a;
    U0 - ім'я (векторизованої) ф-ї користувача з описом U0(x) -
             початкова умова для t=0;
    n  - число точок сітки різницевої схеми по x;
    m  - число m+1 точки сітки різницевої схеми по t;
    Nm - цілочисельний np.array з номерами шарів сітки по t
         для запису (виведення) наближеного розв'язку;
    S  - код типу сіткового шаблона для схеми :
         1 - явної ┘=S{(k-1,j),(k,j),(k,j+1)},
         2 - неявної ┐=S{(k-1,j+1),(k,j+1),(k,j)},
         3 - симетричної □=S{(k-1,j+1),(k,j+1),(k,j),(k-1,j)}.
    Вихідні параметри:
    err- код закінчення:
         0 - перевірка вхідних даних пройшла успішно;
        >0 - при перевірці вхідних даних знайдена помилка і
                  виконання функції аварійно закінчено;
    X  - сітка по x;
    Tp - сітка зі значеннями = Nm*τ;
    Y  - матриця розв'язку (стовпці Y[:,Nm]=U(X,Tp)).
'''

if c <= 0:
    print("fdm_H1Ccp: Коефіцієнт c <= 0, a має бути >0")
    return 1, None, None, None
if a >= b:
    print("fdm_H1Ccp: Неправильно задані значення a, b")
    return 2, None, None, None
if te <= 0:
    print("fdm_H1Ccp: Граничне значення по t має бути >0")
    return 3, None, None, None
if n < 2:
    print("fdm_H1Ccp: Помилкова кількість вузлів n < 2")
    return 4, None, None, None
if m < 1:
    print("fdm_H1Ccp: Помилкова кількість вузлів m < 1")
    return 5, None, None, None
if check_(Nm, m):
    print("fdm_H1Ccp: len(Nm)<1 або елементи Nm[:] не є [0,m]",
          "\n           або не впорядковані по зростанню")
    return 6, None, None, None
if S < 1 or S > 3:
    print("fdm_H1Ccp: Код шаблону має бути S={1,2,3}")
    return 7, None, None, None

X = np.linspace(a, b, n)  # сітка по x
j = len(Nm)
Tp = np.zeros(j)  # сітка виведення по t
Y = np.zeros((n, j))  # матриця під результати

h = X[1] - X[0]
h2 = 0.5 * h
tau = te / m
tau2 = 0.5 * tau
gam = c * tau / h
if S == 1:
    A = gam
```

```
        B = 1 - gam
        C = tau
    else:
        B = 1 / (1 + gam)
        if S == 2:
            A = gam * B
            C = tau * B
        else:
            A = (1 - gam) * B
            C = 2 * tau * B

    t = 0
    y = U0(X)
    j = 0
    jt = 0
    if Nm[0] == j:
        Tp[jt] = t
        Y[:, jt] = y
    while j < m:  # по шарах сітки t
        j += 1
        t = j * tau
        if S != 2: yr = y[0]
        y[0] = ma(t)  # значення на границі x=a
        if S == 2: yr = y[0]
        for k in range(1, n):  # по точках сітки x
            cf = C * f(X[k] - h2, t - tau2)
            yl = yr
            yr = y[k]
            if S == 3:
                y[k] = A * (yr - y[k - 1]) + yl + cf
            else:
                y[k] = A * yl + B * yr + cf
                if S == 2: yr = y[k]
        # запам'ятовуємо розв'язок у точках виведення по t :
        k = jt + 1
        if Nm[k] == j:
            jt = k
            Tp[jt] = t
            Y[:, jt] = y
    return 0, X, Tp, Y


def fdm_H1Cvp(c, f, X, T, ma, U0, Nm):
    '''Метод сіток розв'язування ЛКЗ для рівняння переносу.

          dU(x,t)/dt + c(x,t)*dU(x,t)/dx = f(x,t),
             c(x,t) > 0, (x,t) Є (a,b)*(0,T];
             U(a,t) = Ma(t), t Є [0,T];
             U(x,0) = U0(x), x Є [a,b].
       Вхідні параметри:
       c  - ім'я (скалярної) ф-ї користувача з описом c(x,t);
       f  - ім'я (скалярної) ф-ї користувача з описом f(x,t);
       X  - сітка по x;
       T  - сітка по t;
       ma - ім'я (скалярної) ф-ї користувача з описом Ma(t) -
            крайова умова для x=a;
       U0 - ім'я (векторизованої) ф-ї користувача з описом U0(x) -
            початкова умова для t=0;
```

```
    Nm - цілочисельний np.array з номерами шарів сітки по t
         для запису (виведення) наближеного розв'язку;
    Вихідні параметри:
    err- код закінчення:
         0 - перевірка вхідних даних пройшла успішно;
        >0 - при перевірці вхідних даних знайдена помилка і
                  виконання функції аварійно закінчено;
    Tp - сітка зі значеннями = Nm*т;
    Y  - матриця розв'язку (стовпці Y[:,Nm]=U(X,Tp)).
'''

if not is_order(X):
    print("fdm_H1Cvp: Вузли сітки X не впорядковані або є кратні")
    return 1, None, None
if not is_order(T):
    print("fdm_H1Cvp: Вузли сітки T не впорядковані або є кратні")
    return 2, None, None
n = len(X)
if n < 2:
    print("fdm_H1Cvp: Кількість вузлів сітки X < 2")
    return 3, None, None
m = len(T) - 1
if m < 1:
    print("fdm_H1Cvp: Кількість вузлів сітки T < 1")
    return 4, None, None
if check_(Nm, m):
    print("fdm_H1Cvp: len(Nm)<1 або елементи Nm[:] не є [0,m]",
          "\n           або не впорядковані по зростанню")
    return 5, None, None

j = len(Nm)
Tp = np.zeros(j)  # сітка виведення по t
Y = np.zeros((n, j))  # матриця під результати

t = T[0]
y = U0(X)
j = 0
jt = 0
if Nm[0] == j:
    Tp[jt] = t
    Y[:, jt] = y
while j < m:  # по шарах сітки t
    tau = t
    j += 1
    t = T[j]
    tt = 0.5 * (tau + t)
    tau = t - tau

    yr = y[0]
    x = X[0]
    y[0] = ma(t)  # значення на границі x=a
    for k in range(1, n):  # по точках сітки x
        z = x
        x = X[k]
        h = x - z
        w = f(0.5 * (z + x), tt)
        v = c(x, t)
        yl = yr
```

```
            yr = y[k]
            z = tau * v
            if z <= h:  # явна
                gam = z / h
                z = 1 - gam
                y[k] = gam * yl + z * yr + tau * w
            else:  # неявна
                gam = h / z
                z = 1 - gam
                y[k] = gam * yl + z * y[k - 1] + h / v * w

        # запам'ятовуємо розв'язок в точках виведення по t :
        k = jt + 1
        if Nm[k] == j:
            jt = k
            Tp[jt] = t
            Y[:, jt] = y
    return 0, Tp, Y


def fdm_H1Cxt(c, f, a, b, te, ma, mb, U0, n, m, Nm):
    '''Метод сіток розв’язування ЛКЗ для рівняння переносу.

          dU(x,t)/dt + c(x,t)*dU(x,t)/dx = f(x,t),
            c(x,t),  (x,t) Є (a,b)*(0,T];
             U(a,t) = Ma(t),
             U(b,t) = Mb(t), t Є [0,T];
             U(x,0) = U0(x), x Є [a,b].
       Вхідні параметри:
       c  - ім'я (скалярної) ф-ї користувача з описом c(x,t);
       f  - ім'я (скалярної) ф-ї користувача з описом f(x,t);
       a  - ліва границя відрізку [a,b] (a<b);
       b  - права границя відрізку [a,b];
       te - граничне значення по часу (te>0);
       ma - ім'я (скалярної) ф-ї користувача з описом Ma(t) -
            крайова умова для x=a;
       mb - ім'я (скалярної) ф-ї користувача з описом Mb(t) -
            крайова умова для x=b;
       U0 - ім'я (векторизованої) ф-ї користувача з описом U0(x) -
            початкова умова для t=0;
       n  - число точок сітки різницевої схеми по x;
       m  - число m+1 точки сітки різницевої схеми по t;
       Nm - цілочисельний масив з номерами шарів сітки по t
            для запису (виведення) наближеного розв'язку.
       Вихідні параметри:
       err- код закінчення:
            0 - перевірка вхідних даних пройшла успішно;
           >0 - при перевірці вхідних даних знайдена помилка і
                виконання функції аварійно закінчено;
       X  - сітка по x;
       Tp - сітка зі значеннями = Nm*τ;
       Y  - матриця розв'язку (стовпці Y[:,Nm]=U(X,T)).
    '''

    if a >= b:
        print("fdm_H1Cxt: Неправильно задані значення a, b")
        return 1, None, None, None
```

```
if te <= 0:
    print("fdm_H1Cxt: Граничне значення по t має бути >0")
    return 2, None, None, None
if n < 2:
    print("fdm_H1Cxt: Помилкова кількість вузлів n < 2")
    return 3, None, None, None
if m < 1:
    print("fdm_H1Cxt: Помилкова кількість вузлів m < 1")
    return 4, None, None, None
if check_(Nm, m):
    print("fdm_H1Cxt: len(Nm)<1 або елементи Nm[:] не є [0,m]",
          "\n           або не впорядковані по зростанню")
    return 5, None, None, None

X = np.linspace(a, b, n)  # сітка по x
j = len(Nm)
Tp = np.zeros(j)  # сітка виведення по t
Y = np.zeros((n, j))  # матриця під результати

h = X[1] - X[0]
tau = te / m
th = 0.5 * tau / h

n1 = n - 1
C = np.ones(n)
A = np.zeros(n)
F = np.zeros(n)

t = 0
y = U0(X)
j = 0
jt = 0
if Nm[0] == j:
    Tp[jt] = t
    Y[:, jt] = y
while j < m:
    j += 1
    t = j * tau
    # розрахунок елементів СЛАР :
    for k in range(1, n1):
        x = X[k]
        A[k] = th * c(x, t)
        F[k] = tau * f(x, t) + y[k]
    F[0] = ma(t)
    F[n1] = mb(t)
    y = progn(A, C, -A, F)  # розв'язок СЛАР на шарі j
    # запам'ятовуємо розв'язок в точках виведення по t :
    k = jt + 1
    if Nm[k] == j:
        jt = k
        Tp[jt] = t
        Y[:, jt] = y
return 0, X, Tp, Y
```

**Програмний код Python головного модуля *main_num_pdeH1* для тестування *num_pdeH1* :**

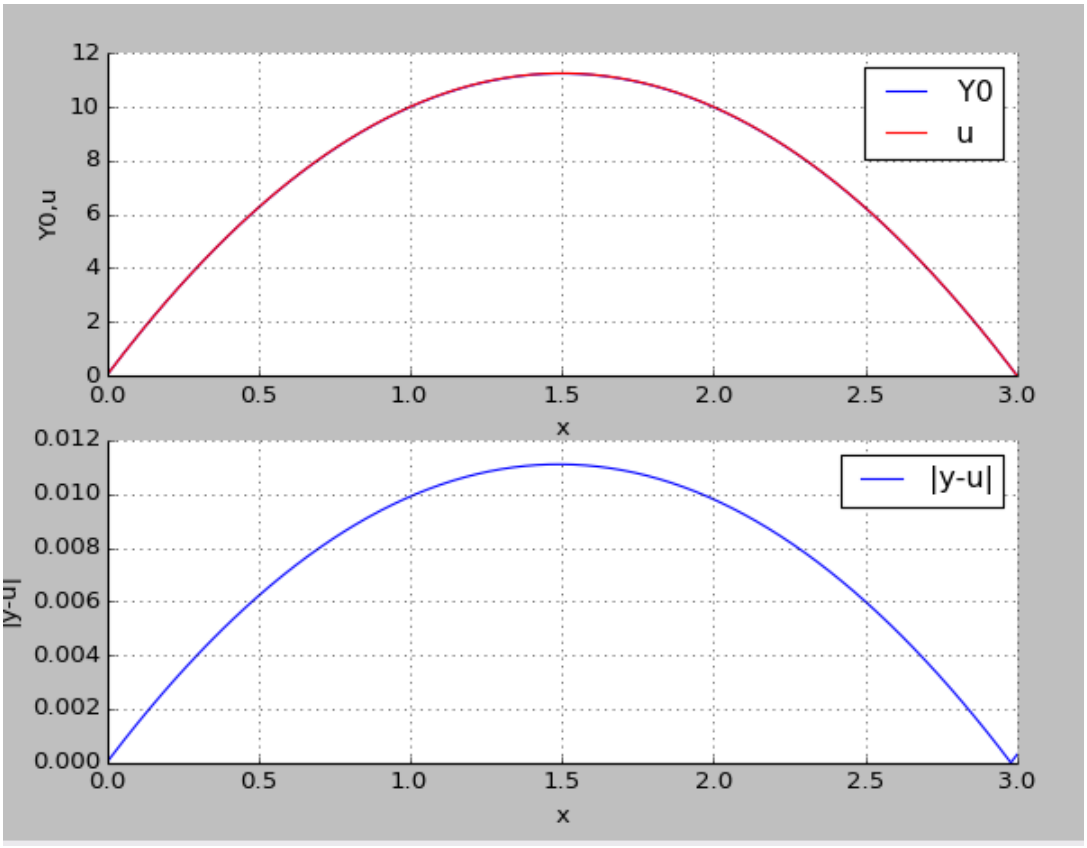
```
import matplotlib.pyplot as plt
from matplotlib import __version__
from util_lne import movespinesticks
from num_pdeH1 import *

MATPLOTLIB_VERSION = float(__version__[:3])

def f(x, t):
    z = 1 + t
    return (x + z) * (3 - x) - x * z


def U0(x):
    return x * (3 - x)


def ma(t):
    return 0.0


def u(x, t):
    """Модельний розв'язок."""
    return (t + 1) * x * (3 - x)


U0v = np.vectorize(U0)

c = 1.0
a = 0.0
b = 3.0
tend = 4.0
n = 151
m = 400
S = 1
Nm = np.array([0, m], 'int')

err, X, Tp, Y = fdm_H1Ccp(c, f, a, b, tend, ma, U0v, n, m, Nm, S)
if err == 0:  # Якщо знайшли наближений розв'язок
    U = np.zeros(n)  # Модельний розв'язок
    y = np.zeros(n)
    for k, x in enumerate(X):
        U[k] = u(x, tend)
        y[k] = Y[k, -1]

    plt.subplot(2, 1, 1)
    plt.plot(X, y, 'b', X, U, 'r')
    plt.grid(True)
    plt.legend(["Y0", "u"], loc="best")
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("Y0,u")
    plt.subplot(2, 1, 2)
    y = abs(y - U)
    print("Сітка:  h=%7.4f  τ=%7.4f" % (X[1] - X[0], tend / m))
```

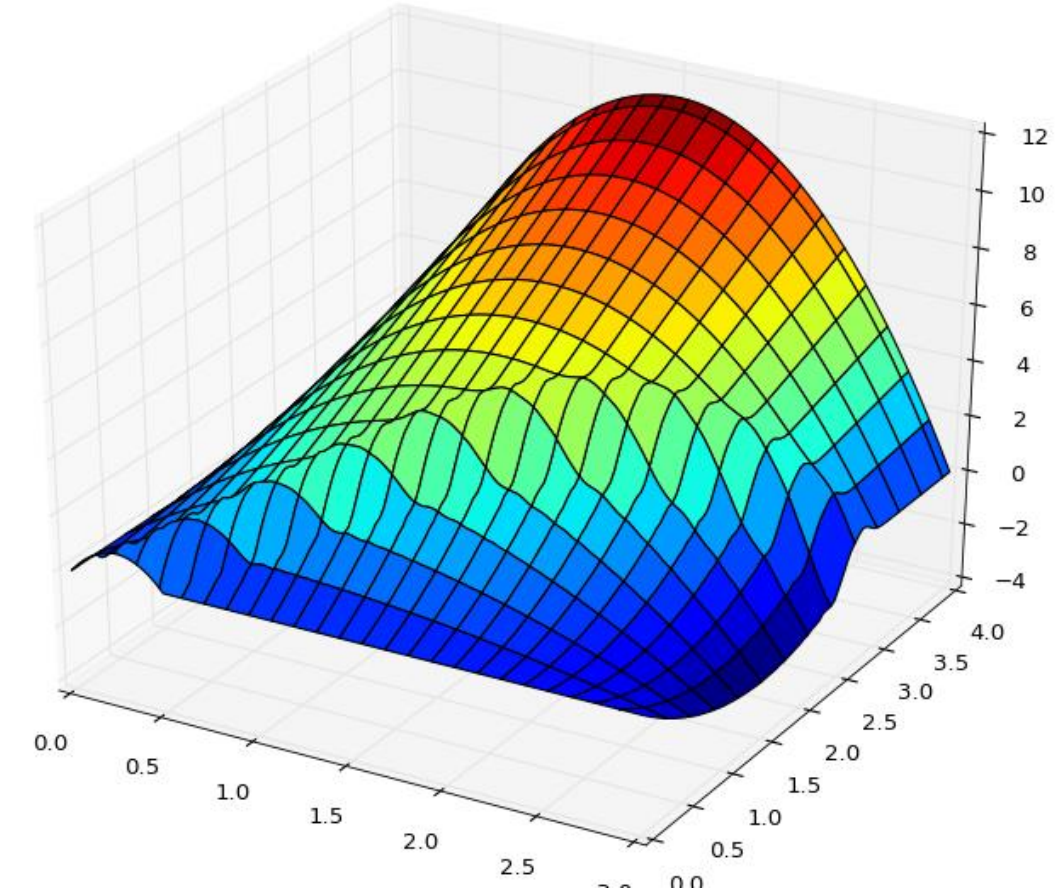
```
    if S == 1:
        print("Схема явна O(h+τ)")
    elif S == 2:
        print("Схема неявна O(h+τ)")
    elif S == 3:
        print("Схема симетрична O(h^2+τ^2)")
    print('Похибка для РС в нормі C=', np.max(y))
    plt.plot(X, y)
    plt.grid(True)
    plt.legend(["|y-u|"], loc="best")
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("|y-u|")
    plt.show()

from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import LinearSegmentedColormap
from matplotlib import cm


def U0(x):
    z = x
    if x < 0.5:
        z = z * (0.5 - z) / 0.0625
    else:
        z = 0.0
    return z


U0v = np.vectorize(U0)
Nm = np.arange(0, m + 1, 5)
err, X, Tp, Y = fdm_H1Ccp(c, f, a, b, tend, ma, U0v, n, m, Nm)
if err == 0:  # Якщо знайшли наближений розв'язок
    XX, TT = np.meshgrid(X, Tp)
    YY = np.transpose(Y)
    fig = plt.figure()
    # щоб працювати з різними версіями matplotlib
    if MATPLOTLIB_VERSION < 3.5:
        ax = Axes3D(fig)
    else:
        ax = Axes3D(fig, auto_add_to_figure=False)
        fig.add_axes(ax)
    ax.plot_surface(XX, TT, YY, rstride=6, cstride=6, cmap=cm.jet)
    plt.show()


def cf(x, t):
    return t * (1 + x)


def f(x, t):
    return x * x + 2 * t


def ma(t):
    return 0.0
```

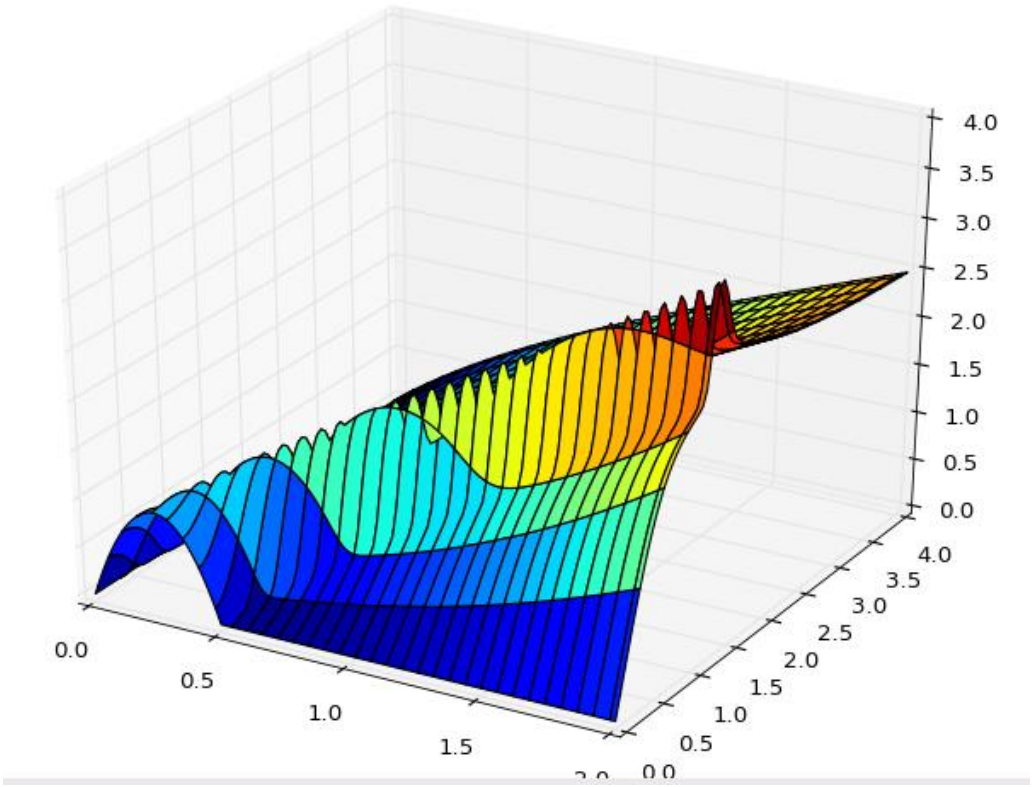
```
def U0(x):
    z = x
    if x < 0.5:
        z = z * (0.5 - z) / 0.0625
    else:
        z = 0.0
    return z


U0v = np.vectorize(U0)
a = 0.0
b = 2.0
tend = 4.0
n = 201
m = 400
X = np.linspace(a, b, n)
T = np.linspace(0, tend, m + 1)
Nm = np.arange(0, m + 1, 5)

err, Tp, Y = fdm_H1Cvp(cf, f, X, T, ma, U0v, Nm)
if err == 0:  # Якщо знайшли наближений розв'язок
    XX, TT = np.meshgrid(X, Tp)
    YY = np.transpose(Y)
    fig = plt.figure()
    # щоб працювати з різними версіями matplotlib
    if MATPLOTLIB_VERSION < 3.5:
        ax = Axes3D(fig)
    else:
        ax = Axes3D(fig, auto_add_to_figure=False)
        fig.add_axes(ax)
    ax.plot_surface(XX, TT, YY, rstride=6, cstride=6, cmap=cm.jet)
    plt.show()


def cf(x, t):
    return t * (0.5 - x) / (1 + x)


def mb(t):
    return 0.0


err, X, Tp, Y = fdm_H1Cxt(cf, f, a, b, tend, ma, mb, U0v, n, m, Nm)
if err == 0:  # Якщо знайшли наближений розв'язок
    XX, TT = np.meshgrid(X, Tp)
    YY = np.transpose(Y)
    fig = plt.figure()
    # щоб працювати з різними версіями matplotlib
    if MATPLOTLIB_VERSION < 3.5:
        ax = Axes3D(fig)
    else:
        ax = Axes3D(fig, auto_add_to_figure=False)
        fig.add_axes(ax)
    ax.plot_surface(XX, TT, YY, rstride=6, cstride=6, cmap=cm.jet)
    plt.show()
```

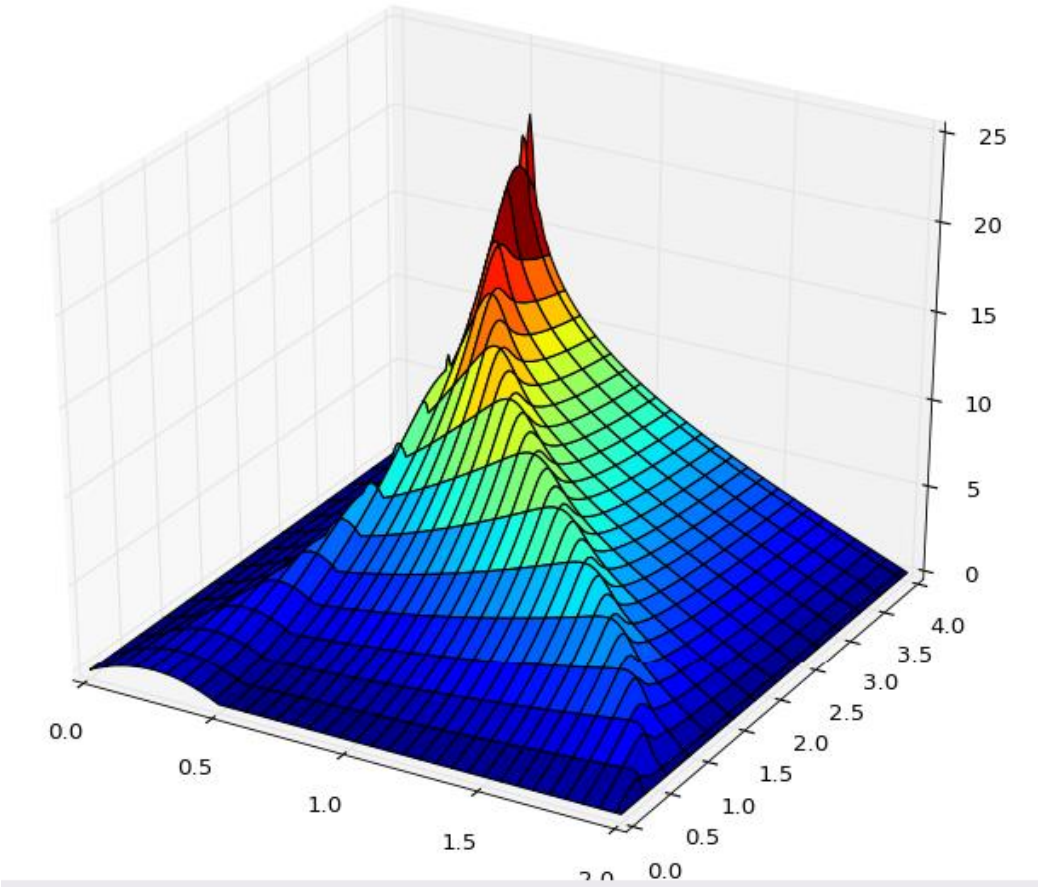
## Результат виконання ***main_num_ pdeH1*** :

```
Сітка:  h= 0.0200  т= 0.0100
Схема симетрична O(h^2+т^2)
Похибка для РС в нормі C= 0.000300000000065
>>>
Сітка:  h= 0.0200  т= 0.0100
Схема неявна O(h+т)
Похибка для РС в нормі C= 0.0336000000001
>>>

Сітка:  h= 0.0200  т= 0.0100
Схема явна O(h+т)
Похибка для РС в нормі C= 0.0111000000001
```




12
10
8
6
4
2
0
−2
−4
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5


25
20
15
10
5
0
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5

## X. Модуль різницевих методів розв'язування лінійних крайових задач для одновимірних рівнянь параболічного типу.

**В модуль *num_pdeP.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *fdm_P1(K,p,f,a,b,te, ma,mb,U0,n,m,Nm,sigm=0.5)* | метод сіток розв'язування ЛКЗ для одновимірного рівняння параболічного типу (схема з вагою *sigm* $\in[0,1]$) |

**Програмний код Python функцій модуля :**

```
"""Модуль РМ розв'язування ЛКЗ для 1-вимірного р-ня параболічного типу.

"""

import numpy as np
from slae_em import progm
from num_pdeH1 import check_


def fdm_P1(K, p, f, a, b, te, ma, mb, U0, n, m, Nm, sigm=0.5):
    '''Метод сіток розв'язування ЛКЗ для р-ня параболічного типу.

         dU/dt = K*d2U/dx2 - p(x,t)U(x,t) + f(x,t),
              K > 0, p>=0, (x,t) Є (a,b)*(0,T];
              U(a,t) = Ma(t), U(b,t) = Mb(t), t Є [0,T];
              U(x,0) = U0(x), x Є [a,b].
       Вхідні параметри:
       K    - коефіцієнт в рівнянні (K>0);
       p    - ім'я (скалярної) ф-ї користувача з описом p(x,t);
       f    - ім'я (скалярної) ф-ї користувача з описом f(x,t);
       a    - ліва границя відрізку [a,b] (a<b);
       b    - права границя відрізку [a,b];
       te   - граничне значення по часу (te>0);
       ma   - ім'я (скалярної) ф-ї користувача з описом
              Ma(t) - крайова умова для x=a;
       mb   - ім'я (скалярної) ф-ї користувача з описом
              Mb(t) - крайова умова для x=b;
       U0   - ім'я (векторизованої) ф-ї користувача з описом
              U0(x) - початкова умова для t=0;
       n    - число точок сітки різницевої схеми по x;
       m    - число m+1 точки сітки різницевої схеми по t;
       Nm   - цілочисельний np.array з номерами шарів сітки по t
              для запису (виведення) наближеного розв'язку;
       sigm - параметр схеми (σ Є [0, 1]).
              σ = 0.5 - схема Кранка-Ніколсон.
       Вихідні параметри:
       err  - код закінчення:
              0 - перевірка вхідних даних пройшла успішно;
             >0 - при перевірці вхідних даних знайдена помилка і
                  виконання функції аварійно закінчено;
       X  - сітка по x;
       Tp - сітка зі значеннями = Nm*τ;
       Y  - матриця розв'язку (стовпці Y[:,Nm]=U(X,Tp)).
    '''
```

```
if K <= 0:
    print("fdm_P1: Коефіцієнт K <= 0, a має бути >0")
    return 1, None, None, None
if a >= b:
    print("fdm_P1: Неправильно задані значення a, b")
    return 2, None, None, None
if te <= 0:
    print("fdm_P1: Граничне значення по t має бути >0")
    return 3, None, None, None
if n < 2:
    print("fdm_P1: Помилкова кількість вузлів n < 2")
    return 4, None, None, None
if m < 1:
    print("fdm_P1: Помилкова кількість вузлів m < 1")
    return 5, None, None, None
if check_(Nm, m):
    print("fdm_P1: len(Nm)<1 або елементи Nm[:] не є [0,m]",
          "\n         або не впорядковані по зростанню")
    return 6, None, None, None
if sigm < 0 or sigm > 1:
    print("fdm_P1: Параметр σ має бути з діапазону [0,1]")
    return 7, None, None, None

X = np.linspace(a, b, n)  # сітка по x
j = len(Nm)
Tp = np.zeros(j)  # сітка виведення по t
Y = np.zeros((n, j))  # матриця під результати

h = X[1] - X[0]
hh = h * h
n1 = n - 1
tau = te / m
tau2 = 0.5 * tau
t = tau / hh
gam = K * t
g2 = 1 - 2 * gam
if sigm == 0:
    if 0.5 / t < 1:
        c = "для σ=0 не виконується умова стійкості"
        print("fdm_P1: ПОПЕРЕДЖЕННЯ", c)
else:
    sigm1 = 1 - sigm
    nju = sigm1 * gam
    gam *= sigm
    g2 = 1 + 2 * gam
    nj2 = 1 - 2 * nju
    ts = tau * sigm
    ts1 = tau * sigm1
    A = gam * np.ones(n)
    A[0] = 0
    A[n1] = 0
    C = np.ones(n)
    F = np.zeros(n)

t = 0.0
y = U0(X)
j = 0
```

```
    jt = 0
    if Nm[0] == j:
        Tp[jt] = t
        Y[:, jt] = y
    while j < m:  # по шарам сітки t
        j += 1
        t = j * tau
        tt = t - tau2
        yr = y[0]
        if sigm == 0:  # алгоритм для явної схеми
            y[0] = ma(t)  # значення на границі x=a
            for k in range(1, n1):  # по внутрішніх точках сітки x
                x = X[k]
                z = g2 - tau * p(x, tt)
                fi = tau * f(x, tt)
                yl = yr
                yr = y[k]
                y[k] = gam * (yl + y[k + 1]) + z * yr + fi
            y[n1] = mb(t)  # значення на границі x=b
        else:  # алгоритм для неявної схеми
            # розрахунок елементів СЛАР :
            for k in range(1, n1):
                x = X[k]
                z = p(x, tt)
                C[k] = g2 + ts * z
                if sigm == 1:  # чисто неявна схема
                    w = y[k]
                else:
                    yl = yr
                    yr = y[k]
                    w = nju * (yl + y[k + 1]) + (nj2 - ts1 * z) * yr
                F[k] = w + tau * f(x, tt)
            F[0] = ma(t)
            F[n1] = mb(t)
            y = progm(A, C, A, F)  # розв'язок СЛАР на шарі j
        # запам'ятовуємо розв'язок в точках виведення по t :
        k = jt + 1
        if Nm[k] == j:
            jt = k
            Tp[jt] = t
            Y[:, jt] = y
    return 0, X, Tp, Y
```

**Програмний код Python головного модуля *main_num_pdeP* для тестування *num_pdeP*:**

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm, __version__
from util_lne import put_arr, movespinesticks
from num_pdeP import *
from math import exp


MATPLOTLIB_VERSION = float(__version__[:3])

def p(x, t):
```

```
    return x * t


def f(x, t):
    return (1 + (x * t - 1.0) * (1.0 + x * (2.0 - x))) * exp(-t)


def U0(x):
    return 1.0 + x * (2.0 - x)


def ma(t):
    return exp(-t)


def mb(t):
    return 2.0 * exp(-t)


def u(x, t):
    """Модельний розв'язок."""
    return (1.0 + x * (2.0 - x)) * exp(-t)


def put_s():
    global U, y
    print("Сітка:  h=%7.4f  τ=%7.4f" % (X[1] - X[0], te / m))
    if sigm == 0:
        print("Схема явна O(h^2+τ)")
    elif 0 < sigm < 1 and sigm != 0.5:
        print("Схема неявна O(h^2+τ) σ=%6.3f" % sigm)
    elif sigm == 0.5:
        print("Схема Кранка-Ніколсон O(h^2+τ^2)")
    elif sigm == 1:
        print("Схема чисто неявна O(h^2+τ)")
    for k, x in enumerate(X):
        U[k] = u(x, te)  # Модельний розв'язок
        y[k] = Y[k, -1]
    put_arr("Розв'язок для t=%5.2f\n" % te, y, "%12.8f", 5)
    y = abs(y - U)
    print('Похибка для РС (останній шар) в нормі C=', np.max(y))


U0v = np.vectorize(U0)

K = 0.5
a = 0.0
b = 1.0
te = 4.0
n = 21
m = 8000
sigm = 0
Nm = np.arange(0, m + 1, 100)
global U, y
err, X, Tp, Y = fdm_P1(K, p, f, a, b, te, ma, mb, U0v, n, m, Nm, sigm)
if err == 0:  # Якщо знайшли наближений розв'язок
    U = np.zeros(n)
    y = np.zeros(n)
```

```
    put_s()

m = 1600
Nm = np.arange(0, m + 1, 20)
for sigm in [0.75, 1]:
    err, X, Tp, Y = fdm_P1(K, p, f, a, b, te, ma, mb, U0v, n, m, Nm,
sigm)
    if err == 0:  # Якщо знайшли наближений розв'язок
        U = np.zeros(n)
        y = np.zeros(n)
        put_s()

sigm = 0.5
err, X, Tp, Y = fdm_P1(K, p, f, a, b, te, ma, mb, U0v, n, m, Nm, sigm)
if err == 0:  # Якщо знайшли наближений розв'язок
    U = np.zeros(n)
    y = np.zeros(n)
    put_s()
    plt.subplot(2, 1, 1)
    plt.plot(X, Y[:, -1], 'b', X, U, 'r')
    plt.grid(True)
    plt.legend(["Y", "u"], loc="best")
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("Y,u")
    plt.subplot(2, 1, 2)
    plt.plot(X, y)
    plt.grid(True)
    plt.legend(["|y-u|"], loc="best")
    movespinesticks()
    plt.xlabel("x")
    plt.ylabel("|y-u|")
    plt.show()

    XX, TT = np.meshgrid(X, Tp)
    YY = np.transpose(Y)
    fig = plt.figure()
    # щоб працювати з різними версіями matplotlib
    if MATPLOTLIB_VERSION < 3.5:
        ax = Axes3D(fig)
    else:
        ax = Axes3D(fig, auto_add_to_figure=False)
        fig.add_axes(ax)
    ax.plot_surface(XX, TT, YY, rstride=6, cstride=6, cmap=cm.jet)
    plt.show()
```

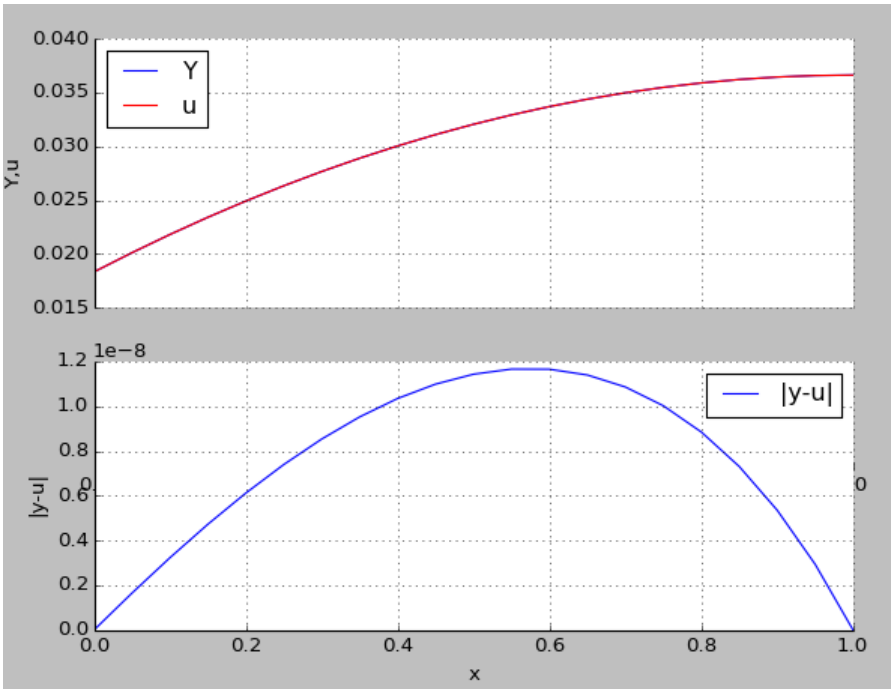
### Результат виконання ***main_num_ pdeP*** :

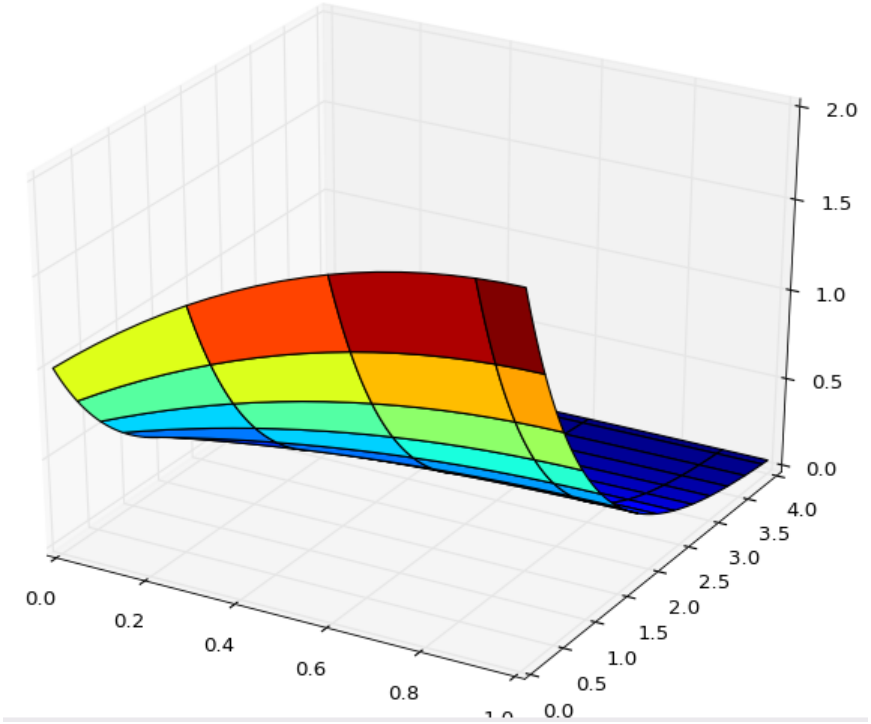
```
Сітка:  h= 0.0500  τ= 0.0005
Схема явна O(h^2+τ)
Розв'язок для t= 4.00
 0.01831564  0.02010079  0.02179438  0.02339644  0.02490696
 0.02632596  0.02765343  0.02888938  0.03003381  0.03108674
 0.03204817  0.03291809  0.03369652  0.03438347  0.03497894
 0.03548293  0.03589547  0.03621655  0.03644621  0.03658444
 0.03663128
Похибка для PC (останній шар) в нормі C= 4.27048516245e-06

Сітка:  h= 0.0500  τ= 0.0025
Схема неявна O(h^2+τ) σ= 0.750
```

Розв'язок для t= 4.00
0.01831564 0.02010298 0.02179867 0.02340269 0.02491503
0.02633566 0.02766458 0.02890178 0.03004723 0.03110093
0.03206287 0.03293303 0.03371140 0.03439797 0.03499270
0.03549559 0.03590661 0.03622573 0.03645291 0.03658811
0.03663128
Похибка для РС (останній шар) в нормі C= 1.06728086474e-05

Сітка: h= 0.0500 т= 0.0025
Схема чисто неявна O(h^2+т)
Розв'язок для t= 4.00
0.01831564 0.02010456 0.02180174 0.02340717 0.02492080
0.02634261 0.02767257 0.02891065 0.03005684 0.03111109
0.03207340 0.03294373 0.03372206 0.03440835 0.03500256
0.03550466 0.03591459 0.03623230 0.03645771 0.03659073
0.03663128
Похибка для РС (останній шар) в нормі C= 2.13707563016e-05

Сітка: h= 0.0500 т= 0.0025
Схема Кранка-Ніколсон O(h^2+т^2)
Розв'язок для t= 4.00
0.01831564 0.02010141 0.02179561 0.02339822 0.02490926
0.02632872 0.02765661 0.02889291 0.03003764 0.03109079
0.03205236 0.03292235 0.03370076 0.03438760 0.03498286
0.03548654 0.03589864 0.03621917 0.03644812 0.03658549
0.03663128
Похибка для РС (останній шар) в нормі C= 1.16541917783e-08




2.0
1.5
1.0
0.5
0.0
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8

## XI. Модуль різницевих методів розв'язування лінійних крайових задач для рівнянь еліптичного типу.

**В модуль *num_pdeE.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *fdm_E_DDDD(f,X1,X2,m1,m2,m3,m4, w,eps=1e-8)* | метод сіток розв'язування ЛКЗ з крайовими умовами Діріхле для рівняння Пуасона в прямокутній області [0,a]*[0,b] |
| *fdm_E_DDNN(f,X1,X2,m1,m2,m3,m4, w,eps=1e-8)* | метод сіток розв'язування ЛКЗ для рівняння Пуасона в прямокутнику D= [0,a]*[0,b] з крайовими умовами: на правій і верхній стороні D – умови Діріхле, на лівій і нижній стороні D – умови Неймана |

**Програмний код Python функцій модуля :**

```
"""Модуль РМ розв'язування ЛКЗ для 2-вимірних р-нь еліптичного типу.

"""

import numpy as np
from interp import is_order


def fdm_E_DDDD(f, X1, X2, m1, m2, m3, m4, w, eps=1e-8):
    '''Метод сіток розв'язування ЛКЗ для рівняння Пуасона.

                ΔU(x1,x2) = -f(x1,x2),
                (x1,x2) Є (0,a)*(0,b);
               U(a,x2) = m1(x2), x2 Є [0,b];
               U(x1,b) = m2(x1), x1 Є [0,a];
               U(0,x2) = m3(x2), x2 Є [0,b];
               U(x1,0) = m4(x1), x1 Є [0,a].
       Вхідні параметри:
       f  - ім'я (скалярної) ф-ї користувача з описом f(x1,x2);
       X1 - сітка по x1;
       X2 - сітка по x2;
       m1 - ім'я (векторизованої) ф-ї користувача з описом m1(x2) -
                крайова умова для x1=a;
       m2 - ім'я (векторизованої) ф-ї користувача з описом m2(x1) -
                крайова умова для x1=a;
       m3 - ім'я (векторизованої) ф-ї користувача з описом m3(x2) -
                крайова умова для x1=0;
       m4 - ім'я (векторизованої) ф-ї користувача з описом m4(x1) -
                крайова умова для x2=0;
       w  - ітераційний параметр ω Є (0,2);
       eps- бажана точність ітераційного процесу.
       Вихідні параметри:
       err- код закінчення:
            0 - перевірка вхідних даних пройшла успішно;
           >0 - при перевірці вхідних даних знайдена помилка і
                    виконання функції аварійно закінчено;
       Y  - матриця розв'язку Y=U(X1,X2).
    '''

    n1 = len(X1)
    if n1 < 2:
```

```
        print("fdm_E_DDDD: Помилкова кількість вузлів по X1")
        return 1, None, None
    if not is_order(X1):
        print("fdm_E_DDDD: Вузли сітки X1 не впорядковані або є кратні")
        return 2, None, None
    n2 = len(X2)
    if n2 < 2:
        print("fdm_E_DDDD: Помилкова кількість вузлів по X2")
        return 3, None, None
    if not is_order(X2):
        print("fdm_E_DDDD: Вузли сітки X2 не впорядковані або є кратні")
        return 4, None, None
    if w <= 0 or w >= 2:
        print("fdm_E_DDDD: Релаксаційний параметр має Є (0,2)")
        return 5, None, None
    if eps <= 0 or eps >= 1:
        print("fdm_E_DDDD: Точність має Є (0,1)")
        return 6, None, None

    N1 = n1 - 1
    N2 = n2 - 1
    h1 = X1[1] - X1[0]
    h2 = X2[1] - X2[0]
    e1 = 1 / h1
    e2 = 1 / h2
    a = e1 * e1
    b = e2 * e2
    h = 2 * (a + b)
    c = w / h
    a *= c
    b *= c
    d = 1 - w

    Y = np.zeros((n1, n2))  # матриця під результати
    Y[N1, :] = m1(X2)  # КУ в x1=a
    Y[:, N2] = m2(X1)  # КУ в x2=b
    Y[0, :] = m3(X2)  # КУ в x1=0
    Y[:, 0] = m4(X1)  # КУ в x2=0
    F = np.zeros((n1, n2))  # допоміжна матриця
    for i in range(n1):
        for j in range(n2):
            F[i, j] = c * f(X1[i], X2[j])

    S = 0
    fl = False
    while not fl:  # ІП
        S += 1
        fl = True
        for j in range(1, N2):
            for i in range(1, N1):
                y = Y[i, j]
                Y[i, j] = a * (Y[i - 1, j] + Y[i + 1, j]) + \
                          b * (Y[i, j - 1] + Y[i, j + 1]) + F[i, j] + d *
Y
                if fl: fl = abs(Y[i, j] - y) <= eps
    return 0, S, Y
```

```
def fdm_E_DDNN(f, X1, X2, m1, m2, m3, m4, w, eps=1e-8):
    '''Метод сіток розв'язування ЛКЗ для рівняння Пуасона.

              ΔU(x1,x2) = -f(x1,x2),
              (x1,x2) Є (0,a)*(0,b);
             U(a,x2) = m1(x2), x2 Є [0,b];
             U(x1,b) = m2(x1), x1 Є [0,a];
             dU(0,x2)/dx1 = m3(x2), x2 Є [0,b);
             dU(x1,0)/dx2 = m4(x1), x1 Є [0,a).
       Вхідні параметри:
       f  - ім'я (скалярної) ф-ї користувача з описом f(x1,x2);
       X1 - сітка по x1;
       X2 - сітка по x2;
       m1 - ім'я (векторизованої) ф-ї користувача з описом m1(x2) -
            крайова умова для x1=a;
       m2 - ім'я (векторизованої) ф-ї користувача з описом m2(x1) -
            крайова умова для x1=a;
       m3 - ім'я (векторизованої) ф-ї користувача з описом m3(x2) -
            крайова умова для x1=0;
       m4 - ім'я (векторизованої) ф-ї користувача з описом m4(x1) -
            крайова умова для x2=0;
       w  - ітераційний параметр ω Є (0,2);
       eps- бажана точність ітераційного процесу.
       Вихідні параметри:
       err- код закінчення:
            0 - перевірка вхідних даних пройшла успішно;
           >0 - при перевірці вхідних даних знайдена помилка і
                виконання функції аварійно закінчено;
       Y  - матриця розв'язку Y=U(X1,X2).
    '''

    n1 = len(X1)
    if n1 < 2:
        print("fdm_E_DDNN: Помилкова кількість вузлів по X1")
        return 1, None, None
    if not is_order(X1):
        print("fdm_E_DDNN: Вузли сітки X1 не впорядковані або є кратні")
        return 2, None, None
    n2 = len(X2)
    if n2 < 2:
        print("fdm_E_DDNN: Помилкова кількість вузлів по X2")
        return 3, None, None
    if not is_order(X2):
        print("fdm_E_DDNN: Вузли сітки X2 не впорядковані або є кратні")
        return 4, None, None
    if w <= 0 or w >= 2:
        print("fdm_E_DDNN: Релаксаційний параметр має Є (0,2)")
        return 5, None, None
    if eps <= 0 or eps >= 1:
        print("fdm_E_DDNN: Точність має Є (0,1)")
        return 6, None, None

    N1 = n1 - 1
    N2 = n2 - 1
    h1 = X1[1] - X1[0]
    h2 = X2[1] - X2[0]
    e1 = 1 / h1
    e2 = 1 / h2
```

```
    a = e1 * e1
    b = e2 * e2
    h = 2 * (a + b)
    c = w / h
    a *= c
    b *= c
    e1 *= 2
    e2 *= 2
    d = 1 - w
    a2 = 2 * a
    b2 = 2 * b

    Y = np.zeros((n1, n2))  # матриця під результати
    Y[N1, :] = m1(X2)  # КУ в x1=a
    Y[:, N2] = m2(X1)  # КУ в x2=b
    F = np.zeros((n1, n2))  # допоміжна матриця
    for i in range(n1):
        for j in range(n2):
            F[i, j] = c * f(X1[i], X2[j])
    F[0, :] = F[0, :] - c * e1 * m3(X2)
    F[:, 0] = F[:, 0] - c * e2 * m4(X1)

    S = 0
    fl = False
    while not fl:  # ІП
        S += 1
        fl = True
        y = Y[0, 0]
        Y[0, 0] = a2 * Y[1, 0] + b2 * Y[0, 1] + F[0, 0] + d * y
        if fl: fl = abs(Y[0, 0] - y) <= eps
        for i in range(1, N1):
            y = Y[i, 0]
            Y[i, 0] = a * (Y[i - 1, 0] + Y[i + 1, 0]) \
                      + b2 * Y[i, 1] + F[i, 0] + d * y
            if fl: fl = abs(Y[i, 0] - y) <= eps
        for j in range(1, N2):
            y = Y[0, j]
            Y[0, j] = a2 * Y[1, j] + b * (Y[0, j - 1] + Y[0, j + 1]) \
                      + F[0, j] + d * y
            if fl: fl = abs(Y[0, j] - y) <= eps
            for i in range(1, N1):
                y = Y[i, j]
                Y[i, j] = a * (Y[i - 1, j] + Y[i + 1, j]) + \
                          b * (Y[i, j - 1] + Y[i, j + 1]) + F[i, j] + d *
y
                if fl: fl = abs(Y[i, j] - y) <= eps
    return 0, S, Y
```

**Програмний код Python головного модуля *main_num_pdeE* для тестування *num_pdeE*:**

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm, __version__
from num_pdeE import *

MATPLOTLIB_VERSION = float(__version__[:3])
```

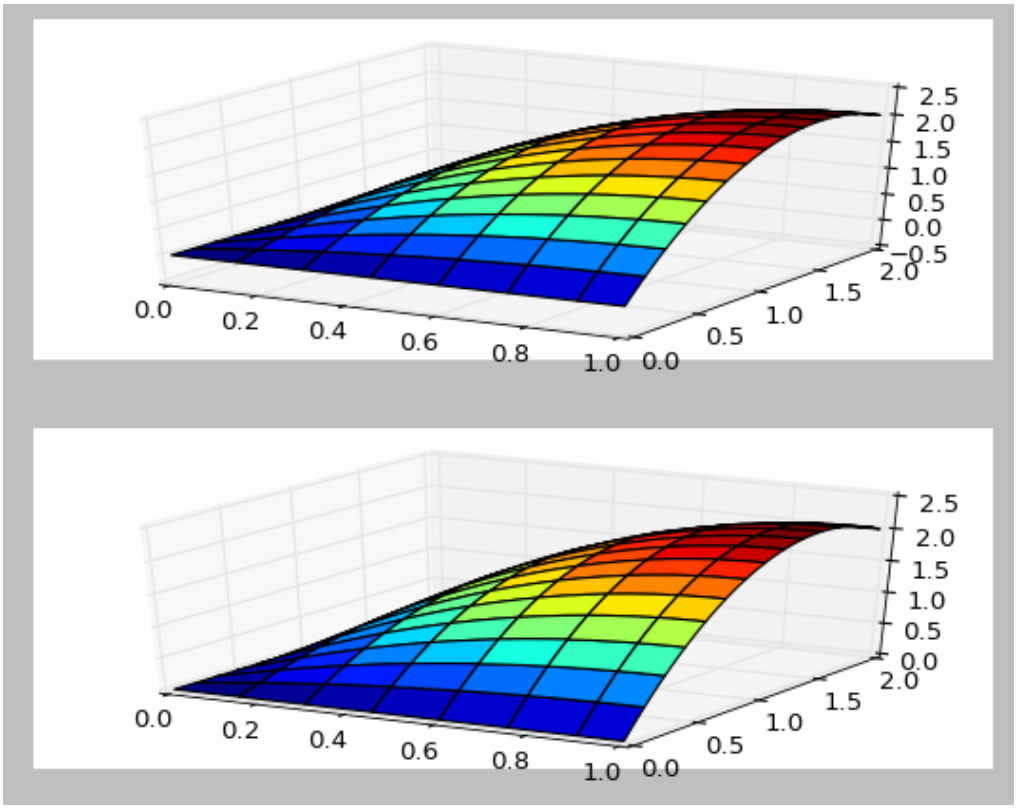
```
print("Розв'язання модельної задачі\n",
      '    ΔU(x,y) = -f(x,y),\n',
      '    (x,y) Є (0,1)*(0,2);\n',
      "    U(a,y) = m1(y),\n",
      "    U(x,b) = m2(x),\n",
      "    U(a,y) = m3(y),\n",
      "    U(x,b) = m4(x).\n",
      "Для f(x,y)=2(5-x^2-y^2),\n",
      "m1(y)=0,  m2(x)=0,\n",
      "m3(y)=4-y^2, m4(x)=4(1-x^2)\n",
      "розв'язок буде U(x,y)=(1-x^2)(4-y^2).")


def f(x, y):
    return -2.0 * (x * x + y * y - 5)


def m1(y):
    return 0.0


def m2(x):
    return 0.0


def m3(y):
    return -y * y + 4.0


def m4(x):
    return -4.0 * (x * x - 1)


def u(x, y):
    """Модельний розв'язок."""
    return (x * x - 1) * (y * y - 4)


m1v = np.vectorize(m1)
m2v = np.vectorize(m2)
m3v = np.vectorize(m3)
m4v = np.vectorize(m4)
uv = np.vectorize(u)

X1 = np.linspace(0, 1.0, 21)
X2 = np.linspace(0, 2.0, 41)
w = 1 / (0.5 + np.sin(0.5 * np.pi * 0.5 * 0.05 / 2))
err, it, Y = fdm_E_DDDD(f, X1, X2, m1v, m2v, m3v, m4v, w, 1e-9)
if err == 0:  # Якщо знайшли наближений розв'язок
    XX1, XX2 = np.meshgrid(X1, X2)
    YY = np.transpose(Y)
    fig, ax = plt.subplots(2, 1, subplot_kw=dict(projection='3d'))
    ax[0].plot_surface(XX1, XX2, YY, rstride=3, cstride=3, cmap=cm.jet)
    U = uv(XX1, XX2)  # Модельний розв'язок
    ax[1].plot_surface(XX1, XX2, U, rstride=3, cstride=3, cmap=cm.jet)
    plt.show()
```

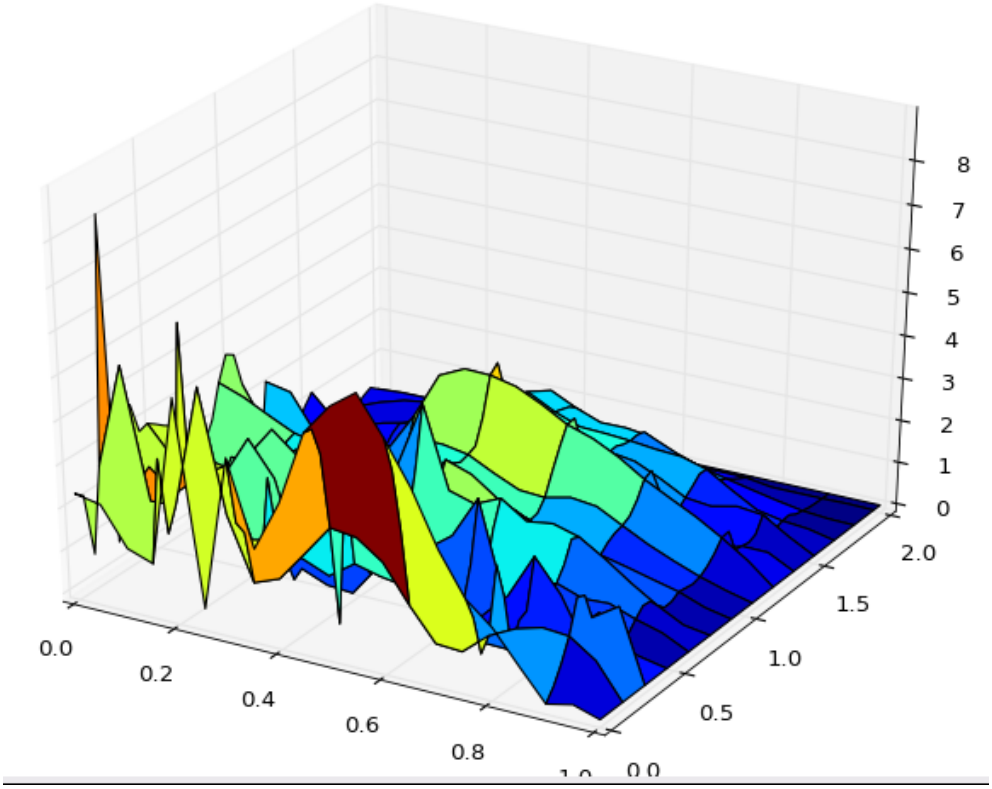
```
    YY = abs(YY - U)
    print("Кроки сітки=", X1[1] - X1[0], X2[1] - X2[0])
    print("Релаксаційний параметр=", w)
    print("Кількість ітерацій=", it)
    print('Похибка для РС в нормі C=', np.max(YY))
    fig = plt.figure()
    # щоб працювати з різними версіями matplotlib
    if MATPLOTLIB_VERSION < 3.5:
        ax = Axes3D(fig)
    else:
        ax = Axes3D(fig, auto_add_to_figure=False)
        fig.add_axes(ax)
    ax.plot_surface(XX1, XX2, YY, rstride=3, cstride=3, cmap=cm.jet)
    plt.show()

print("Розв'язання модельної задачі\n",
      '    ΔU(x,y) = -f(x,y),\n',
      '    (x,y) Є (0,1)*(0,2);\n',
      "    U(a,y) = m1(y),\n",
      "    U(x,b) = m2(x),\n",
      "    U'x(a,y) = m3(y),\n",
      "    U'y(x,b) = m4(x).\n",
      "Для f(x,y)=2x(2-x)+2y(3-y),\n",
      "m1(y)=y(3-y),  m2(x)=2x(2-x),\n",
      "m3(y)=2y(3-y), m4(x)=3x(2-x)\n",
      "розв'язок буде U(x,y)=x(2-x)y(3-y).")


def f(x, y):
    return 2 * (x * (2 - x) + y * (3 - y))


def m1(y):
    return -y * (y - 3)


def m2(x):
    return -2 * x * (x - 2)


def m3(y):
    return -2 * y * (y - 3)


def m4(x):
    return -3 * x * (x - 2)


def u(x, y):
    """Модельний розв'язок."""
    return x * (x - 2) * y * (y - 3)


m1v = np.vectorize(m1)
m2v = np.vectorize(m2)
m3v = np.vectorize(m3)
m4v = np.vectorize(m4)
uv = np.vectorize(u)
```

```
X1 = np.linspace(0, 1.0, 21)
X2 = np.linspace(0, 2.0, 41)
w = 1 / (0.5 + np.sin(0.5 * np.pi * 0.5 * 0.05 / 2))
err, it, Y = fdm_E_DDNN(f, X1, X2, m1v, m2v, m3v, m4v, w, 1e-9)
if err == 0:  # Якщо знайшли наближений розв'язок
    XX1, XX2 = np.meshgrid(X1, X2)
    YY = np.transpose(Y)
    fig, ax = plt.subplots(2, 1, subplot_kw=dict(projection='3d'))
    ax[0].plot_surface(XX1, XX2, YY, rstride=3, cstride=3, cmap=cm.jet)
    U = uv(XX1, XX2)  # Модельний розв'язок
    ax[1].plot_surface(XX1, XX2, U, rstride=3, cstride=3, cmap=cm.jet)
    plt.show()

    YY = abs(YY - U)
    print("Кроки сітки=", X1[1] - X1[0], X2[1] - X2[0])
    print("Релаксаційний параметр=", w)
    print("Кількість ітерацій=", it)
    print('Похибка для РС в нормі C=', np.max(YY))
    fig = plt.figure()
    # щоб працювати з різними версіями matplotlib
    if MATPLOTLIB_VERSION < 3.5:
        ax = Axes3D(fig)
    else:
        ax = Axes3D(fig, auto_add_to_figure=False)
        fig.add_axes(ax)
    ax.plot_surface(XX1, XX2, YY, rstride=3, cstride=3, cmap=cm.jet)
    plt.show()
```

**Результат виконання *main_num_pdeE*:**

```
Розв'язання модельної задачі
   ΔU(x,y) = -f(x,y),
   (x,y) Є (0,1)*(0,2);
   U(a,y) = m1(y),
   U(x,b) = m2(x),
   U(a,y) = m3(y),
   U(x,b) = m4(x).
 Для f(x,y)=2(5-x^2-y^2),
 m1(y)=0,  m2(x)=0,
 m3(y)=4-y^2, m4(x)=4(1-x^2)
 розв'язок буде U(x,y)=(1-x^2)(4-y^2).
Кроки сітки= 0.05 0.05
Релаксаційний параметр= 1.92443256569
Кількість ітерацій= 275
Похибка для РС в нормі C= 1.28958621559e-09

Розв'язання модельної задачі
   ΔU(x,y) = -f(x,y),
   (x,y) Є (0,1)*(0,2);
   U(a,y) = m1(y),
   U(x,b) = m2(x),
   U'x(a,y) = m3(y),
   U'y(x,b) = m4(x).
 Для f(x,y)=2x(2-x)+2y(3-y),
 m1(y)=y(3-y),  m2(x)=2x(2-x),
 m3(y)=2y(3-y), m4(x)=3x(2-x)
 розв'язок буде U(x,y)=x(2-x)y(3-y).
Кроки сітки= 0.05 0.05
Релаксаційний параметр= 1.92443256569
Кількість ітерацій= 295
Похибка для РС в нормі C= 8.18395061771e-10
```


2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
2.0


8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
0.0
0.5
1.0
1.5
2.0

## XII. Модуль різницевих методів розв'язування лінійних крайових задач для рівнянь гіперболічного типу другого порядку.

**В модуль *num_pdeH2.py* входять наступні функції :**

| Назва функції | Призначення |
|---|---|
| *fdm_H2(K,p,f,a,b,te,ma,mb,U0,U1, n,m,Nm,sigm=0.25)* | метод сіток розв'язування ЛКЗ для одновимірних рівнянь гіперболічного типу 2-го порядку |

**Програмний код Python функцій модуля :**

```
"""Модуль РМ розв'язування ЛКЗ для р-нь гіперболічного типу 2 порядку.

"""

import numpy as np
from slae_em import progm
from num_pdeH1 import check_


def fdm_H2(K, p, f, a, b, te, ma, mb, U0, U1, n, m, Nm, sigm=0.25):
    '''Метод сіток розв'язування ЛКЗ для гіперболічного рівняння.

        d2U/dt2 = K*d2U/dx2 - p(x,t)*U(x,t) + f(x,t),
             K > 0, p(x,t)>=0, (x,t) Є (a,b)*(0,T];
          U(a,t) = Ma(t), U(b,t) = Mb(t), t Є [0,T];
         U(x,0) = U0(x), dU(x,0)/dt = U1(x), x Є [a,b].
       Вхідні параметри:
       K    - коефіцієнт в рівнянні (K>0);
       p    - ім'я (скалярної) ф-ї користувача з описом p(x,t);
       f    - ім'я (скалярної) ф-ї користувача з описом f(x,t);
       a    - ліва границя відрізку [a,b] (a<b);
       b    - права границя відрізку [a,b];
       te   - граничне значення по часу (te>0);
       ma   - ім'я (скалярної) ф-ї користувача з описом
              Ma(t) - крайова умова для x=a;
       mb   - ім'я (скалярної) ф-ї користувача з описом
              Mb(t) - крайова умова для x=b;
       U0   - ім'я (векторизованої) ф-ї користувача з описом
              U0(x) - початкова умова для t=0;
       U1   - ім'я (векторизованої) ф-ї користувача з описом
              U1(x) - початкова умова для t=0;
       n    - число точок сітки різницевої схеми по x;
       m    - число m+1 точки сітки різницевої схеми по t;
       Nm   - цілочисельний np.array з номерами шарів сітки по t
              для запису (виведення) наближеного розв'язку;
       sigm - параметр схеми (σ Є [0, 0.5]).
       Вихідні параметри:
       err  - код закінчення:
              0 - перевірка вхідних даних пройшла успішно;
             >0 - при перевірці вхідних даних знайдена помилка і
                  виконання функції аварійно закінчено;
       X  - сітка по x;
       Tp - сітка зі значеннями = Nm*τ;
       Y  - матриця розв'язку (стовпці Y[:,Nm]=U(X,Tp)).
    '''
```

```
if K <= 0:
    print("fdm_H2: Коефіцієнт K <= 0, a має бути >0")
    return 1, None, None, None
if a >= b:
    print("fdm_H2: Неправильно задані значення a, b")
    return 2, None, None, None
if te <= 0:
    print("fdm_H2: Граничне значення по t має бути >0")
    return 3, None, None, None
if n < 2:
    print("fdm_H2: Помилкова кількість вузлів n < 2")
    return 4, None, None, None
if m < 1:
    print("fdm_H2: Помилкова кількість вузлів m < 1")
    return 5, None, None, None
if check_(Nm, m):
    print("fdm_H2: len(Nm)<1 або елементи Nm[:] не є [0,m]",
          "\n         або не впорядковані по зростанню")
    return 6, None, None, None
if sigm < 0 or sigm > 0.5:
    print("fdm_H2: Параметр σ має бути з діапазону [0, 0.5]")
    return 7, None, None, None

X = np.linspace(a, b, n)  # сітка по x
j = len(Nm)
Tp = np.zeros(j)  # сітка виведення по t
Y = np.zeros((n, j))  # матриця під результати

h = X[1] - X[0]
n1 = n - 1
tau = te / m
gam = tau / h
pk = True
if sigm == 0:
    pk = False
    if gam > 1:
        c = "для σ=0 не виконується умова стійкості"
        print("fdm_H2: ПОПЕРЕДЖЕННЯ", c)
elif 0 < sigm < 0.25:
    if gam > (1 - 4 * sigm) ** 0.5:
        c = "для σ Є (0,0.25) не виконується умова стійкості"
        print("fdm_H2: ПОПЕРЕДЖЕННЯ", c)

t = 0.0
j = 0
jt = 0
w = U0(X)  # шар j = 0
if Nm[0] == j:
    Tp[jt] = t
    Y[:, jt] = w

g2K = gam * gam * K
g2K1 = 1 - g2K
z = 0.5 * g2K
tt = tau * tau
tt5 = 0.5 * tt
v = U1(X)
```

```
    j += 1  # шар j = 1
    for k in range(1, n1):
        x = X[k]
        v[k] = z * (w[k - 1] + w[k + 1]) + (g2K1 - tt5 * p(x, t)) * w[k]
\
               + tau * v[k] + tt5 * f(x, t)
    t += tau
    v[0] = ma(t)
    v[n1] = mb(t)
    k = jt + 1
    if Nm[k] == j:
        jt = k
        Tp[jt] = t
        Y[:, jt] = v

    if pk:  # допом.змінні і "заготовка" векторів при σ≠0
        kp = 1 - 2 * sigm
        n2K = kp * g2K
        z = 2 * (1 - n2K)
        nt5 = kp * tt
        g2K *= sigm
        g2K1 = 1 + 2 * g2K
        tt5 = sigm * tt

        A = g2K * np.ones(n)
        A[0] = 0
        A[n1] = 0
        C = np.ones(n)
        F = np.zeros(n)
    else:
        z = 2 * g2K1
    y = np.zeros(n)

    while j < m:  # по шарам сітки t
        j += 1
        tj = t
        t = j * tau
        if sigm == 0:  # алгоритм для явної схеми
            y[0] = ma(t)  # значення на границі x=a
            for k in range(1, n1):  # по внутрішніх точках сітки x
                x = X[k]
                y[k] = g2K * (v[k - 1] + v[k + 1]) + \
                       (z - tt * p(x, tj)) * v[k] - w[k] + tt * f(x, tj)
            y[n1] = mb(t)  # значення на границі x=b
        else:  # алгоритм для неявної схеми
            # розрахунок елементів СЛАР :
            for k in range(1, n1):
                x = X[k]
                pk = p(x, tj)
                km = k - 1
                kp = k + 1
                C[k] = g2K1 + tt5 * pk
                wk = g2K * (w[km] + w[kp]) - (g2K1 + tt5 * pk) * w[k]
                vk = n2K * (v[km] + v[kp]) + (z - nt5 * pk) * v[k]
                F[k] = wk + vk + tt * f(x, tj)
            F[0] = ma(t)
            F[n1] = mb(t)
            y = progm(A, C, A, F)  # розв'язок СЛАР на шарі j
```

```
        for k in range(n):  # для нового кроку по j
            w[k] = v[k]
            v[k] = y[k]

        # запам'ятовуємо розв'язок у точках виведення по t :
        k = jt + 1
        if Nm[k] == j:
            jt = k
            Tp[jt] = t
            Y[:, jt] = y
    return 0, X, Tp, Y
```

**Програмний код Python головного модуля *main_num_pdeH2* для тестування *num_pdeH2* :**

```
"""Модуль РМ розв’язування ЛКЗ для р-нь гіперболічного типу 2 порядку.

"""

import numpy as np
from slae_em import progm
from num_pdeH1 import check_


def fdm_H2(K, p, f, a, b, te, ma, mb, U0, U1, n, m, Nm, sigm=0.25):
    '''М.сіток розв’язування ЛКЗ для гіперболічного рівняння.

        d2U/dt2 = K*d2U/dx2 - p(x,t)*U(x,t) + f(x,t),
             K > 0, p(x,t)>=0, (x,t) Є (a,b)*(0,T];
          U(a,t) = Ma(t), U(b,t) = Mb(t), t Є [0,T];
         U(x,0) = U0(x), dU(x,0)/dt = U1(x), x Є [a,b].
       Вхідні параметри:
       K    - коефіцієнт в рівнянні (K>0);
       p    - ім'я (скалярної) ф-ї користувача з описом p(x,t);
       f    - ім'я (скалярної) ф-ї користувача з описом f(x,t);
       a    - ліва границя відрізку [a,b] (a<b);
       b    - права границя відрізку [a,b];
       te   - граничне значення по часу (te>0);
       ma   - ім'я (скалярної) ф-ї користувача з описом
              Ma(t) - крайова умова для x=a;
       mb   - ім'я (скалярної) ф-ї користувача з описом
              Mb(t) - крайова умова для x=b;
       U0   - ім'я (векторизованої) ф-ї користувача з описом
              U0(x) - початкова умова для t=0;
       U1   - ім'я (векторизованої) ф-ї користувача з описом
              U1(x) - початкова умова для t=0;
       n    - число точок сітки різницевої схеми по x;
       m    - число m+1 точки сітки різницевої схеми по t;
       Nm   - цілочисельний np.array з номерами шарів сітки по t
              для запису (виведення) наближеного розв'язку;
       sigm - параметр схеми (σ Є [0, 0.5]).
       Вихідні параметри:
       err  - код закінчення:
              0 - перевірка вхідних даних пройшла успішно;
             >0 - при перевірці вхідних даних знайдена помилка і
                  виконання функції аварійно закінчено;
```

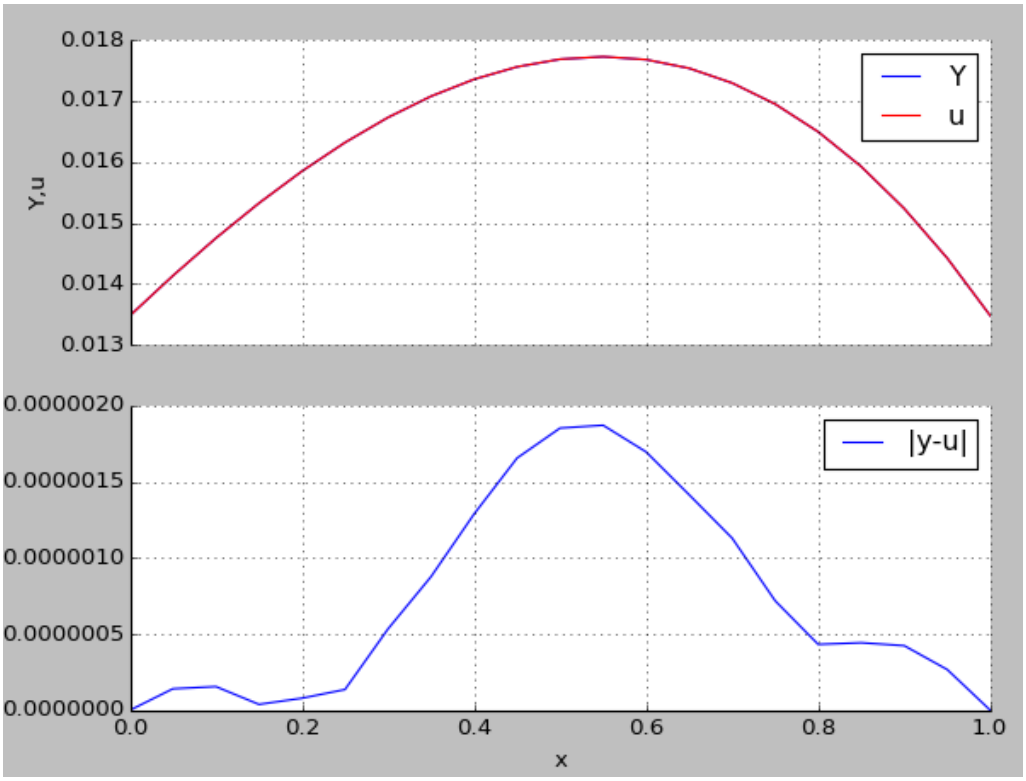
```
    X  - сітка по x;
    Tp - сітка зі значеннями = Nm*τ;
    Y  - матриця розв'язку (стовпці Y[:,Nm]=U(X,Tp)).
'''

if K <= 0:
    print("fdm_H2: Коефіцієнт K <= 0, а має бути >0")
    return 1, None, None, None
if a >= b:
    print("fdm_H2: Неправильно задані значення a, b")
    return 2, None, None, None
if te <= 0:
    print("fdm_H2: Граничне значення по t має бути >0")
    return 3, None, None, None
if n < 2:
    print("fdm_H2: Помилкова кількість вузлів n < 2")
    return 4, None, None, None
if m < 1:
    print("fdm_H2: Помилкова кількість вузлів m < 1")
    return 5, None, None, None
if check_(Nm, m):
    print("fdm_H2: len(Nm)<1 або елементи Nm[:] не є [0,m]",
          "\n        або не впорядковані по зростанню")
    return 6, None, None, None
if sigm < 0 or sigm > 0.5:
    print("fdm_H2: Параметр σ має бути з діапазону [0, 0.5]")
    return 7, None, None, None

X = np.linspace(a, b, n)  # сітка по x
j = len(Nm)
Tp = np.zeros(j)  # сітка виведення по t
Y = np.zeros((n, j))  # матриця під результати

h = X[1] - X[0]
n1 = n - 1
tau = te / m
gam = tau / h
pk = True
if sigm == 0:
    pk = False
    if gam > 1:
        c = "для σ=0 не виконується умова стійкості"
        print("fdm_H2: ПОПЕРЕДЖЕННЯ", c)
elif 0 < sigm < 0.25:
    if gam > (1 - 4 * sigm) ** 0.5:
        c = "для σ Є (0,0.25) не виконується умова стійкості"
        print("fdm_H2: ПОПЕРЕДЖЕННЯ", c)

t = 0.0
j = 0
jt = 0
w = U0(X)  # шар j = 0
if Nm[0] == j:
    Tp[jt] = t
    Y[:, jt] = w

g2K = gam * gam * K
g2K1 = 1 - g2K
```

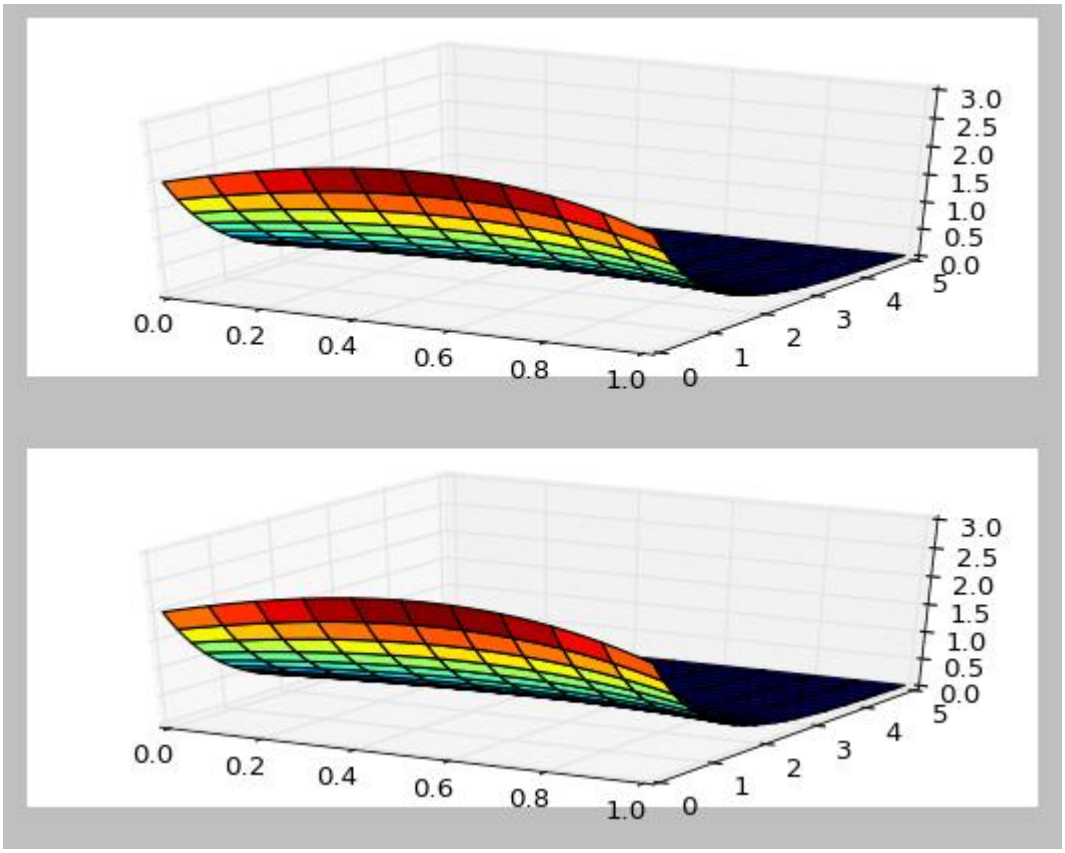
```
    z = 0.5 * g2K
    tt = tau * tau
    tt5 = 0.5 * tt
    v = U1(X)
    j += 1  # шар j = 1
    for k in range(1, n1):
        x = X[k]
        v[k] = z * (w[k - 1] + w[k + 1]) + (g2K1 - tt5 * p(x, t)) * w[k]
\
               + tau * v[k] + tt5 * f(x, t)
    t += tau
    v[0] = ma(t)
    v[n1] = mb(t)
    k = jt + 1
    if Nm[k] == j:
        jt = k
        Tp[jt] = t
        Y[:, jt] = v

    if pk:  # допом.змінні і "заготовка" векторів при σ≠0
        kp = 1 - 2 * sigm
        n2K = kp * g2K
        z = 2 * (1 - n2K)
        nt5 = kp * tt
        g2K *= sigm
        g2K1 = 1 + 2 * g2K
        tt5 = sigm * tt

        A = g2K * np.ones(n)
        A[0] = 0
        A[n1] = 0
        C = np.ones(n)
        F = np.zeros(n)
    else:
        z = 2 * g2K1
    y = np.zeros(n)

    while j < m:  # по шарам сітки t
        j += 1
        tj = t
        t = j * tau
        if sigm == 0:  # алгоритм для явної схеми
            y[0] = ma(t)  # значення на границі x=a
            for k in range(1, n1):  # по внутрішніх точках сітки x
                x = X[k]
                y[k] = g2K * (v[k - 1] + v[k + 1]) + \
                       (z - tt * p(x, tj)) * v[k] - w[k] + tt * f(x, tj)
            y[n1] = mb(t)  # значення на границі x=b
        else:  # алгоритм для неявної схеми
            # розрахунок елементів СЛАР :
            for k in range(1, n1):
                x = X[k]
                pk = p(x, tj)
                km = k - 1
                kp = k + 1
                C[k] = g2K1 + tt5 * pk
                wk = g2K * (w[km] + w[kp]) - (g2K1 + tt5 * pk) * w[k]
                vk = n2K * (v[km] + v[kp]) + (z - nt5 * pk) * v[k]
```

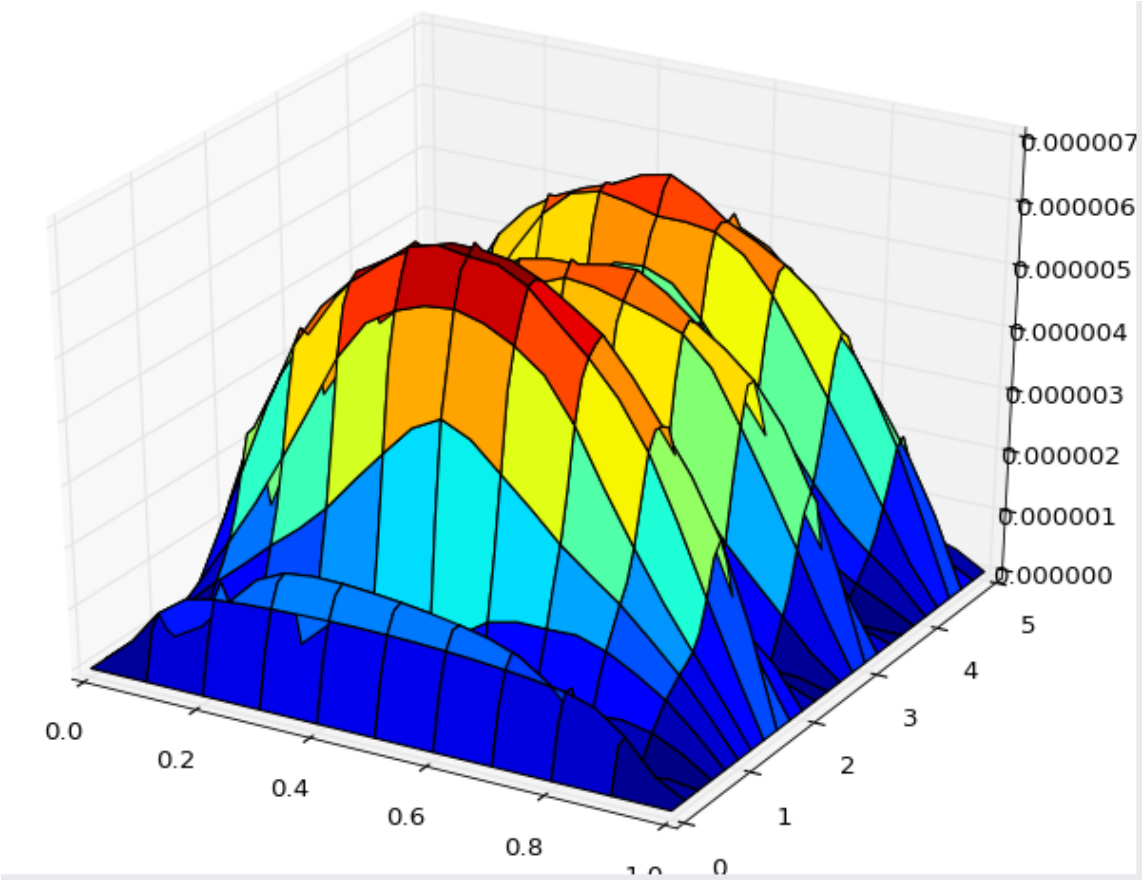
```
                F[k] = wk + vk + tt * f(x, tj)
            F[0] = ma(t)
            F[n1] = mb(t)
            y = progm(A, C, A, F)   # розв'язок СЛАР на шарі j

        for k in range(n):   # для нового кроку по j
            w[k] = v[k]
            v[k] = y[k]

        # запам'ятовуємо розв'язок у точках виведення по t :
        k = jt + 1
        if Nm[k] == j:
            jt = k
            Tp[jt] = t
            Y[:, jt] = y
    return 0, X, Tp, Y
```

### Результат виконання ***main_num_pdeH2*** :

```
Сітка:  h= 0.0500  τ= 0.0400
Схема явна O(h^2+τ^2)= 4.10000000000e-03
Розв'язок для t= 5.00

 0.01347589  0.01408423  0.01466460  0.01520475  0.01569142
 0.01612276  0.01649160  0.01679373  0.01702802  0.01720410
 0.01732435  0.01737037  0.01735044  0.01725215  0.01705585
 0.01675260  0.01633653  0.01580469  0.01514876  0.01437179
 0.01347589
Похибка для PC (останній шар) в нормі C= 3.62760164464e-04

Сітка:  h= 0.0500  τ= 0.0050
Схема неявна σ= 0.200 O(h^2+τ^2)= 2.52500000000e-03
Розв'язок для t= 5.00
 0.01347589  0.01413174  0.01474897  0.01532253  0.01584725
 0.01631775  0.01672912  0.01707656  0.01735490  0.01755919
 0.01768450  0.01772578  0.01767799  0.01753599  0.01729463
 0.01694896  0.01649368  0.01592357  0.01523389  0.01441969
 0.01347589
Похибка для PC (останній шар) в нормі C= 2.61230644366e-06
...

Сітка:  h= 0.0500  τ= 0.0050
Схема неявна σ= 0.250 O(h^2+τ^2)= 2.52500000000e-03
Розв'язок для t= 5.00
 0.01347589  0.01413186  0.01474921  0.01532290  0.01584773
 0.01631833  0.01672979  0.01707729  0.01735566  0.01755996
 0.01768526  0.01772651  0.01767868  0.01753662  0.01729518
 0.01694943  0.01649406  0.01592385  0.01523407  0.01441978
 0.01347589
Похибка для PC (останній шар) в нормі C= 1.87121379875e-06
```


0.018
0.017
0.016
0.015
0.014
0.013
Y,u
Y
u
0.0000020
0.0000015
0.0000010
0.0000005
0.0000000
|y-u|
0.0
0.2
0.4
0.6
0.8
1.0
x


3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5


0.000007
0.000006
0.000005
0.000004
0.000003
0.000002
0.000001
0.000000
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5

# СПИСОК ЛІТЕРАТУРИ


1. *Вакал Є. С.* Класифікація рівянь з частинними похідними з використанням системи MATLAB / Є.С.Вакал Ю.Є.Вакал. – К.: Основа, 2017. – 104 с.
2. *Вакал Є. С.* Методи математичної фізики в прикладах і задачах / Є.С.Вакал, А.В.Ловейкін. – К.: Видавець Кравченко Я.О., 2020. – 188 с.
3. *Вержбицкий В. М.* Основы численных методов: Учебник для вузов / В. М. Вержбицкий. – М.: Высш. шк., 2002. – 840 с.
4. Використання математичного пакета MATLAB для розв'язування прикладних задач: навч. посіб. / Б. П. Довгий, Є. С. Вакал, Ю. Є. Вакал, А. В. Попов. – К.: Фітосоціоцентр, 2012. – 78 с.
5. *Довгий Б. П.* Інформаційні систем та технології: навч. посіб. / Б.П.Довгий, Є.С.Вакал. – К.: Видавець Кравченко Я.О., 2021. – 111 с.
6. *Довгий Б. П.* Методи обчислень. Лекції для студентів механіко-математичного факультету: Навч. посіб. / Б.П.Довгий. – К.: [Електронний ресурс]. – Режим доступу: http://www.matfiz.univ.kiev.ua/userfiles/files/Calc_Methods_Lect.pdf , 2021. – 81 с.
7. *Калиткин Н. Н.* Численные методы / Н. Н. Калиткин. – М.: Наука, 1978. – 512 с.
8. *Ляшко И. И.* Методы вычислений / И. И. Ляшко, В. Л. Макаров, А. А. Скоробогатько. – К.: Вища шк., 1977. – 408 с.
9. Методи обчислень: методичні вказівки до лабораторних робіт із використанням пакета MATLAB для розв'язування прикладних задач: навч. посіб. / Б. П. Довгий, Є. С. Вакал, Ю. Є. Вакал. – К.: ВПЦ «Київський університет», 2017. – 60 с.
10. Методичні вказівки та учбові завдання до самостійних робіт над курсом “Методи обчислень” для студентів механіко-математичного факультету: навч. видання / А. А. Глущенко, Б. П. Довгий та ін. – К.: ВПЦ "Київський університет", 1992. – 124 с.
11. Методичні вказівки і учбові завдання для аудиторної та самостійної роботи з навчальної дисципліни “Методи обчислень”: навч. посіб. / Б. П. Довгий, А. В. Ловейкін. – К.: ВПЦ «Київський університет», 2020. – 115 с.
12 *Обвінцев О. В.* Інформатика та програмування. Курс на основі Python. Матеріали лекцій: навч. посіб. / О.В.Обвінцев. – К.: Основа., 2017. – 248 с.
13. *Обвінцев О. В.* Об’єктно-орієнтоване програмування. Курс на основі Python. Матеріали лекцій: навч. посіб. / О.В.Обвінцев. – К.: Основа., 2017. – 326 с.
14. *Попов В. В.* Методи обчислень / В. В. Попов. – К.: ВПЦ "Київський університет", 2012. – 303 с.
15. Программное обеспечение ЭВМ МИР-1 и МИР-2. т.2 Программы / В. М. Глушков, И. Н. Молчанов и др. – К.: Наукова думка, 1976. – 371 с.
16. *Самарский А. А.* Методы решения сеточных уравнений / А. А. Самарский, Е. С. Николаев. – М.: Наука, 1978. – 592 с.
17. *Самарский А. А.* Теория разностных схем / А. А. Самарский. – М.: Наука, 1977. – 656 с.
18. *Самарский А. А.* Численные методы / А. А. Самарский, А. В. Гулин. – М.: Наука, 1989. – 439 с.
19. Сплайн-функції та їхнє застосування: навч. посіб. / Б. П. Довгий, А. В. Ловейкін, Є. С. Вакал, Ю. Є. Вакал. – К.: ВПЦ «Київський університет», 2017. – 120 с.

ЗМІСТ

**Навчальне видання**

**ДОВГИЙ Борис Павлович**
**ВАКАЛ Євген Сергійович**
**ГАП'ЯК Ігор Васильович**
**ОБВІНЦЕВ Олександр Вальдемарович**

# НАВЧАЛЬНИЙ ПОСІБНИК ІЗ ДИСЦИПЛІНИ "МЕТОДИ ОБЧИСЛЕНЬ"

**для студентів механіко-математичного факультету заочної форми навчання**

**Підписано до друку 28.10.2022**
**Формат 60x84$^{1/8}$. Папір офсетний.**
**Гарнітура: Times**