

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

**до лабораторних робіт
з дисципліни**

«Алгоритми та методи обчислень»

для студентів спеціальності 123 «Комп'ютерна інженерія»
всіх форм навчання

2017

Методичні вказівки до лабораторних робіт з дисципліни
"Алгоритми та методи обчислень" для студентів спеціальності 123
"Комп'ютерна інженерія" всіх форм навчання / Укл.: В.В. Шкарупило,
Т.В. Голуб, Н.В. Щербак. – Запоріжжя: ЗНТУ, 2017. – 62 с.

Укладачі:

В.В. Шкарупило, доцент, к.т.н.
Т.В. Голуб, асистент
Н.В. Щербак, асистент

Рецензент:

Р.К. Кудерметов, доцент, к.т.н.

Відповідальний
за випуск:

Т.В. Голуб, асистент

Затверджено:
на засіданні кафедри

"Комп'ютерні системи та мережі"
Протокол № 8 від 27.04.2017 р.

Рекомендовано до видання:

НМК факультету КНТ
Протокол № 10 від 29.05.2017 р.

ЗМІСТ

ЛАБОРАТОРНА РОБОТА №1

1 МЕТОДИ НАБЛИЖЕННЯ ФУНКЦІЙ.....	5
1.1 Теоретичні відомості.....	5
1.1.1 Сортуння даних.....	7
1.1.2 Вирішення задач інтерполяції та екстраполяції.....	9
1.1.3 Вирішення задачі апроксимації.....	12
1.2 Завдання до лабораторної роботи.....	14
1.3 Зміст звіту.....	17
1.4 Контрольні питання.....	17

ЛАБОРАТОРНА РОБОТА №2

2 МЕТОДИ РОЗВ'ЯЗАННЯ СЛАР.....	18
2.1 Теоретичні відомості.....	18
2.1.1 Прямі методи.....	18
2.1.2 Ітераційні методи.....	23
2.2 Завдання до лабораторної роботи.....	28
2.3 Зміст звіту.....	29
2.4 Контрольні питання.....	29

ЛАБОРАТОРНА РОБОТА №3

3 БАЗОВІ АЛГОРИТМИ.....	31
3.1 Теоретичні відомості.....	31
3.1.1 Алгоритми пошуку.....	31
3.1.2 Алгоритми сортування.....	33
3.2 Завдання до лабораторної роботи.....	36
3.3 Зміст звіту.....	40
3.4 Контрольні питання.....	40

ЛАБОРАТОРНА РОБОТА №4

4 МЕТОДИ РОЗВ'ЯЗАННЯ ДИФЕРЕНЦІЙНИХ РІВНЯНЬ.....	42
4.1 Теоретичні відомості.....	42
4.1.1 Чисельне розв'язання ЗДР.....	43
4.1.2 Чисельне розв'язання ЗДР у середовищі SciLab.....	44
4.2 Завдання до лабораторної роботи.....	48
4.3 Зміст звіту.....	49
4.4 Контрольні питання.....	49
5 РОЗРАХУНКОВО-ГРАФІЧНЕ ЗАВДАННЯ.....	50

5.1 Теоретичні відомості.....	50
5.2 Завдання до виконання	53
5.3 Зміст звіту.....	55
5.4 Контрольні питання.....	56
ЛІТЕРАТУРА	57
ДОДАТОК А Числа Фібоначчі.....	58
ДОДАТОК Б Програмні реалізації методів.....	59

ЛАБОРАТОРНА РОБОТА №1

1 МЕТОДИ НАБЛИЖЕННЯ ФУНКЦІЙ

Мета роботи – набути навичок чисельного вирішення задач інтерполяції, екстраполяції та апроксимації.

1.1 Теоретичні відомості

Вирішення задач інтерполяції та апроксимації тісно пов'язане із науково-виробничою діяльністю, зокрема із обробкою експериментальних даних та дослідженням певних процесів. При цьому поняття інтерполяції та апроксимації, хоч і є близькими за своєю суттю, мають суттєві відмінності.

Апроксимація – процес знаходження функції $f(x)$, максимально наближеної до заданих точок. Тобто під апроксимацією варто розуміти наближення певної функції $f(x)$ іншою, більш простою подібною функцією $\varphi(x)$, використання якої дозволяє спростити процес дослідження [6].

Інтерполяція полягає в обчисленні значень функції $f(x)$ у точках, відмінних від заданих (вузлових) у визначеному інтервалі $[a, b]$. Інтерполяція є частковим випадком апроксимації.

Задачу інтерполяції можна формалізувати наступним чином:

– нехай у межах інтервалу $[a, b]$ задано $n+1$ опірних (вузлових) точок: $a \leq x_0 \leq x_1 \leq \dots \leq x_n \leq b$. Задано $n+1$ значень y_i , що є значеннями функції $f(x_i)$ у вузлових точках. Необхідно знайти поліном $P_n(x)$: $(\forall 0 \leq i \leq n) P_n(x_i) = y_i$.

У випадку, якщо мова йде про знаходження значень функції $f(x)$ поза межами інтервалу $[a, b]$, то кажуть про екстраполяцію. Типові сценарії вирішення названої задачі пов'язані із дослідженням певних економічних процесів, коли виникає необхідність спрогнозувати значення деякої величини.

Типовий вид поліноміальної функції наступний:

$$P_n(x) = c_0 + c_1 \cdot x + c_2 \cdot x^2 + \dots + c_n \cdot x^n, \quad (1.1)$$

де $c_i = \text{const}$ ($i = 0, 1, \dots, n \in N$) – коефіцієнт.

Серед усіх видів функцій – раціональні, степеневі, логарифмічні тощо – саме поліноміальні функції вигляду (1.1) є найбільш придатними для вирішення задачі інтерполяції, оскільки дозволяють зручно варіювати ступінь наближення апроксимуючої функції $P_n(x)$ до $f(x)$ за рахунок зміни значення n .

Із збільшенням значення n експоненційно зростає обчислювальна складність вирішуваної задачі апроксимації. Саме тому необхідно підбирати таке значення n (найменше достатнє), за якого вирішення задачі апроксимації вдовольняло би заданій точності. Як правило, при вирішенні задачі апроксимації використовують поліноми із значеннями n не більше 4 або 5.

Можна виділити наступні види інтерполяції:

- інтерполяція поліномом $P_n(x)$ ступеню n ; у випадку $n = 2$ маємо справу з квадратичною інтерполяцією;
- кусково-лінійна інтерполяція – частковий випадок поліноміальної інтерполяції, коли $n = 1$; в даному випадку сусідні вузлові точки з'єднуються прямими;
- сплайн-інтерполяція – є своєрідним різновидом поліноміальної інтерполяції, але у даному випадку інтервал $[a, b]$ розбивається на підінтервали, на яких використовуються поліноми з різними n . Це необхідно для того, щоб зменшити певною мірою обчислювальну складність вирішуваної задачі інтерполяції із дотриманням необхідного наближення $P_n(x)$ до $f(x)$: на підінтервалах, де $f(x)$ є більш гладкою, використовується $P_n(x)$ із відносно меншим значенням n , і навпаки.

Виконання лабораторної роботи складається із чотирьох частин:

- підготовка даних до вирішення задач інтерполяції, екстраполяції та апроксимації, а саме – сортування даних;
- вирішення задачі інтерполяції;
- вирішення задачі екстраполяції;
- вирішення задачі апроксимації.

Зауваження: Перша, друга та третя частини роботи виконується у середовищі *SciLab*, четверта частина роботи виконується у середовищі *TableCurve 2D*.

1.1.1 Сортювання даних

Як правило, для вирішення задачі інтерполяції, вихідні дані подаються або в табличному вигляді, або у вигляді списку – пар значень (x_i, y_i) . Найчастіше такі дані вже є відсортованими за зростанням по x . Зазвичай значення опірних точок задаються із фіксованим кроком, тобто $x_{i+1} = x_i + h$, де $h = \text{const}$.

У випадку, якщо вихідні пари значень (x_i, y_i) не є відсортованими за x , в нагоді можуть стати вбудовані засоби середовища *SciLab*. Вони полягають у використанні алгоритму швидкого сортювання, а саме – вбудованої функції *qsort*:

`[B [, k]] = gsort (A, option, direction),`

де A – вектор або матриця, що підлягає сортюванню;

`option` – необов'язковий аргумент, що задає параметри сортювання; значення за замовчанням – 'g' – всі елементи вектору (матриці) A підлягають сортюванню;

`direction` – параметр для визначення напрямку сортювання: 'i' (increase) – за зростанням; 'd' (decrease) – за спаданням (за замовчанням);

B – масив (вектор або матриця), тих самих розмірів, що і вихідний масив A , але призначений для зберігання результату вирішення задачі сортювання;

k – масив індексів елементів масиву A у новому відсортованому масиві B .

Розглянемо приклад. Нехай маємо два масиви – x та y . Масив x містить Ox -координати вузлових точок, а масив y – Oy -координати. Масиви x та y є невідсортованими (табл. 1.1).

Таблиця 1.1 – Вихідні дані

i:	0	1	2	3	4
x:	6	4	2	8	10
y:	1	5	4	3	2

В табл. 1.1 і – порядкові номери заданих пар значень координат (x_i, y_i) . Необхідно упорядкувати пари (x_i, y_i) за зростанням Ox -значень. Засобами *SciLab* це можна здійснити наступним чином:

– сформувати вихідні масиви x та y :

```
x = [6 4 2 8 10];
y = [1 5 4 3 2];
```

– здійснити безпосередньо сортування за зростанням за x :

```
[xx, k] = gsort(x, 'g', 'i');
```

У результаті цього відсортована послідовність з x -елементів міститиметься у результуючому масиві xx . При цьому масив k зберігатиме індекси елементів вихідного масиву x у новому відсортованому масиві xx . У даному випадку, якщо переглянути результуючі масиви, матимемо наступне:

```
xx =
    2.  4.  6.  8. 10.
k =
    3.  2.  1.  4.  5.
```

– тепер необхідно підв'язати y -елементи до індексів масиву k :

```
yy = y(k);
```

У результаті цього матимемо послідовність, представлену масивом yy , відсортовану згідно нового порядку елементів вихідного масиву x , представленого масивом xx . Матимемо наступне:

```
yy =
    4.  5.  1.  3.  2.
```

– тепер необхідно сформувати масив пар значень – на основі відповідних елементів масивів xx та yy :

```
mx = [xx; yy];
```

Матимемо наступне:

```
mx =
    2.  4.  6.  8. 10.
    4.  5.  1.  3.  2.
```

Сортування завершено.

1.1.2 Вирішення задач інтерполяції та екстраполяції

Розглянемо кусково-лінійну інтерполяцію.

Здійснення кусково-лінійної інтерполяції та візуалізація одержуваних результатів засобами *SciLab* полягає у використанні зв'язки функцій `interp1n` та `plot2d`.

Функція `interp1n` має наступний вигляд:

```
[y]=interp1n(xyd,xrange),
```

де `xyd` – матриця з 2-ма рядками – Ox - та Oy -координат вузлових точок;

`xrange` – абсциса – діапазон по Ox , на якому вирішується задача інтерполяції;

`y` – вектор-розв'язок – контейнер результату вирішення задачі інтерполяції.

Представлення результатів вирішення задачі інтерполяції або екстраполяції як правило здійснюється у вигляді графіку. Для цього зручно використовувати функцію *SciLab* `plot2d`, яка має наступний вигляд:

```
plot2d([x,],y[,style[,strf[,leg[,rect]]]]),
```

де `x` – вектор Ox -значень;

`y` – вектор Oy -значень;

`style` – аргумент, що задає формат візуалізації кривих;

`strf` – бінарний код у форматі "xyz" (три позиції), що задає формат візуалізації підписів кривих;

`leg` – аргумент, що задає текстові підписи кривих;

`rect` – вектор з 4-х елементів, що задає діапазони зміни значень Ox - та Oy -координат – діапазони візуалізації;

Розглянемо приклад вирішення задачі кусково-лінійної інтерполяції. Нехай у якості вихідних даних маємо раніше одержану відсортовану матрицю `mx`.

Вирішимо задачу у три кроки:

– виведемо на графіку опірні точки:

```
plot2d(x',y',[-3],'011','',[0,0,12,6]);
```

Результат цього кроку подано на рис. 1.1.

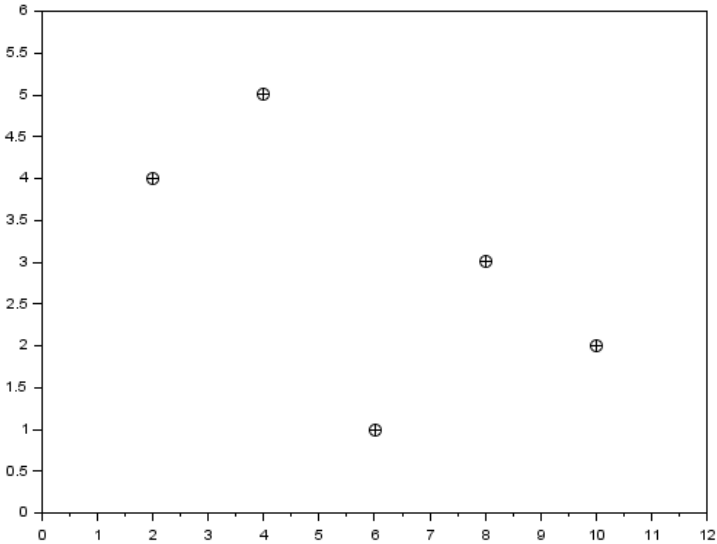


Рисунок 1.1 – Графічне представлення опірних точок

– здійснимо кусково-лінійну інтерполяцію за допомогою функції `interp1n`.

Для цього попередньо задамо межі Ох-діапазону, на якому задача має бути вирішена:

```
range=(0:12);
```

У нашому випадку цей діапазон повністю перекриває (із запасом) діапазон зміни значень Ох-координат опірних точок.

Збережемо результат вирішення задачі у масиві `res`:

```
res= interp1n(mx,range);
```

– представимо на графіку результат вирішення задачі кусково-лінійної інтерполяції (рис. 1.2):

```
plot2d(range',res',[3],'000');
```

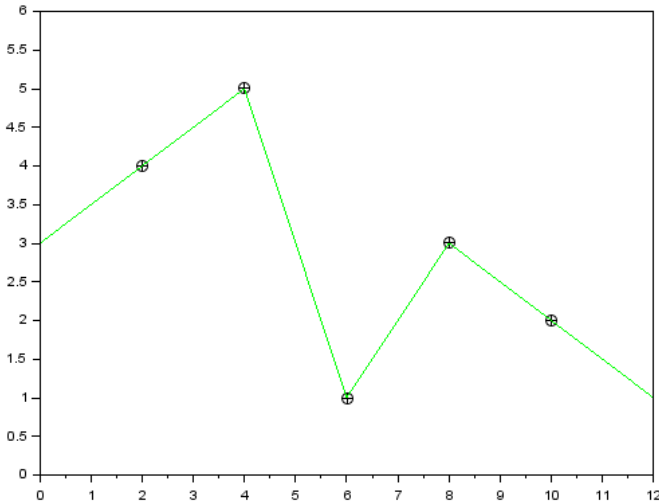


Рисунок 1.2 – Результат вирішення задачі кусково-лінійної інтерполяції

По суті у даному випадку (рис. 1.2) ми вирішили не лише задачу інтерполяції, а й задачу екстраполяції. Оскільки при вирішенні задачі інтерполяції прийнято вважати, що найменше та найбільше значення Ох-координат опірних точок формують собою межі інтервалу, на якому вирішується задача. Ми ж перевищили ці межі.

Для одержання значення функції $f(x)$ у довільній точці, у якості другого аргументу функції `interp1n` слід передавати не діапазон, а єдине значення Ох-координати. Наприклад, для Ох-координати 7 шуканої точки, Оу-координату якої намагаємось знайти, матимемо наступне:

```
point=interp1n(mx,7);
point =
    2.
```

Отже, маємо точку $(7;2)$ координатної площини, що не є вузловою (рис. 1.2). У випадку, наприклад, $x_i = 9$ матимемо точку $(9;2.5)$. Таким чином, можна вважати, що задачу кусково-лінійної інтерполяції було успішно вирішено – оскільки маємо можливість

знаходити значення функції $f(x)$, представленої опірними точками (послідовністю пар значень), у точках, відмінних від вузлових, на інтервалі, обмеженому найменшим та найбільшим значеннями Ох-координат опірних точок: у нашому випадку – у межах інтервалу $[2, 10]$.

Для вирішення задачі екстраполяції, у якості аргументу *SciLab*-функції `interp1n` слід передати значення, що знаходиться за межами інтервалу $[2, 10]$. Наприклад, для Ох-координати 11 матимемо наступне:

```
point=interp1n(mx,11);
point =
    1.5
```

Такий випадок вже є вирішенням задачі екстраполяції.

1.1.3 Вирішення задачі апроксимації

Для вирішення задачі апроксимації можна використовувати середовище *TableCurve 2D*, яке дозволяє отримати графічне подання заданої послідовності та одержати його аналітичне представлення.

Для цього відсортовану послідовність пар значень слід представити у вигляді двох стовпців даних текстового файлу.

Перший стовпець вмісту файлу – Ох-координати опірних точок, другий стовпець – відповідні Оу-координати. Кожний рядок файлу представляє собою координати окремої опірної точки. Елементи пар значень мають відокремлюватися пробілом.

Створений та заповнений текстовий файл подається у якості вихідних даних до середовища *TableCurve 2D*. Для цього слід виконати наступну послідовність кроків:

- відкрити середовище *TableCurve 2D*;
- виконати імпорт вихідних даних: *File -> Import...* (*Ctrl+I*), вказавши шлях до текстового файлу; ввести назви Ох- та Оу-осей.

У результаті виконання другого кроку буде виведено графік – результат вирішення задачі кусково-лінійної інтерполяції, що є ідентичним до графіку, наведеному на рис. 1.2.

Для вирішення саме задачі апроксимації, необхідно виконати також наступний крок:

– у меню обрати наступне: *Process -> Curve-Fit Poly/Ratl Equations (Ctrl+P)*; натиснути кнопку *Graph Start*.

У результаті виконання цього кроку буде виведено результат вирішення задачі апроксимації (рис. 1.3).

На рис. 1.3 представлено результат апроксимації функції $f(x)$ поліномом $P_3(x)$:

$$P_3(x) = a + b/x + c/x^2 + d/x^3, \quad (1.2)$$

де a, b, c, d – коефіцієнти: $a \approx 14,007$, $b \approx -194,010$, $c \approx 911,006$, $d \approx -1125,991$.

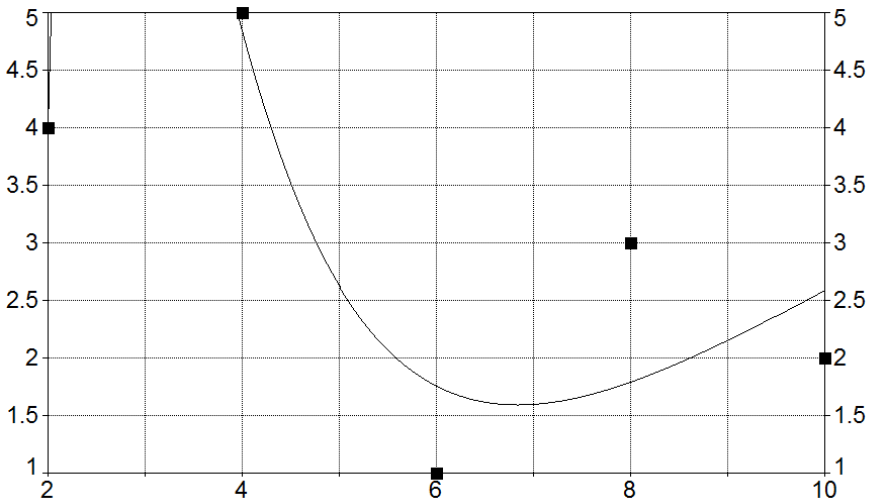


Рисунок 1.3 – Результат вирішення задачі апроксимації

Те, що мова йде саме про апроксимацію, а не про інтерполяцію, видно з рис. 1.3 по тому, що крива не проходить через опірні точки.

Середовище *TableCurve 2D* дозволяє підбирати апроксимуючий поліном $P_n(x)$ із списку доступних. Елементи списку різняться між собою за значеннями коефіцієнтів детермінації – R^2 . Коефіцієнт детермінації $0 \leq R^2 \leq 1$ характеризує ступінь наближення

апроксимуючої функції (поліному) до досліджуваної функції $f(x)$. Чим більше значення коефіцієнту, тим успішнішим є результат вирішення задачі апроксимації. У випадку розглянутого прикладу $R^2 \approx 0,758$, що є досить невтішним результатом. Бажано, щоб $R^2 \geq 0,95$.

Зауваження: значення коефіцієнту детермінації виводиться у нижній частині екрану в окремому вікні у якості елементу другого стовпця списку – навпроти відповідного апроксимуючого поліному. Аналітичне представлення поліному виводиться також у верхній частині екрану – на окремому вікні – разом із зазначенням коефіцієнтів.

1.2 Завдання до лабораторної роботи

1.2.1 Підготувати вихідні дані згідно варіанту до вирішення задачі інтерполяції, а саме, відсортувати за зростанням, керуючись прикладом, який розглянуто у п. 1.1.1. **Зауваження:** спочатку усі пари значень вихідних даних згідно варіанту необхідно попередньо розмістити у довільному порядку.

1.2.2 Вирішити задачу кусково-лінійної інтерполяції на діапазоні значень p згідно варіанту. Результатом вирішення інтерполяції має бути графік (див. приклад у п. 1.1.2). Знайти значення функції, представленої послідовністю пар значень, у точці, відмінній від вузлової.

1.2.3 Вирішити задачу екстраполяції, а саме, знайти значення функції, представленої послідовністю пар значень згідно варіанту, у точці, відмінній від вузлової, поза межами діапазону значень p (див. приклад у п. 1.1.2).

1.2.4 Вирішити задачу апроксимації, керуючись прикладом, який розглянуто у п. 1.1.3. Текстовий файл з вихідними даними для роботи у середовищі *TableCurve 2D* повинен мати назву *Surame_PC-num.txt*, де *Surame* – прізвище студента, а *PC-num* – номер комп'ютера. Результатом вирішення апроксимації має бути підібраний апроксимуючий поліном з відповідними значеннями коефіцієнтів та відповідним значенням коефіцієнту детермінації.

Номер варіанту визначається студентом за номером комп'ютера. У випадку, якщо значення номеру комп'ютера перевищує значення

номеру варіанту, номер варіанту визначається як залишок від ділення номеру комп'ютера на 10, плюс один.

Для всіх варіантів: у якості вихідних даних слід прийняти...

Варіант 1: ... пари значень $(n, F(n))$, де $n = 0, 1, \dots, 9$ – порядковий номер числа Фібоначчі $F(n)$ (Додаток А).

Варіант 2: ... пари значень $(n, G(n))$, де $n = 0, 1, \dots, 9$ – Ох-координати вузлових точок, $G(n)$ – Оу-координати – коди Грея для заданого n . Код Грея обчислюється наступним чином: $G(n) = n \wedge (n >> 1)$, тобто n зводиться у ступінь, значенням якого є результат зміщення значення n вправо на один біт. Коди Грея для $n = 0, 1, \dots, 9$ подано у табличному вигляді (табл. 1.2). У якості Оу-координат вузлових точок необхідно узяти десяткові представлення бінарних кодів, наведених у табл. 1.2.

Таблиця 1.2 – Коди Грея

n	0	1	2	3	4
$G(n)$	0000	0001	0011	0010	0110
n	5	6	7	8	9
$G(n)$	0111	0101	0100	1100	1101

Варіант 3: ... курси валют із прив'язкою до місяців року. Тобто $n = 1, 2, \dots, 12$ – Ох-координати вузлових точок – порядкові номери місяців року. У якості Оу-координат узяти курси певної валюти по відношенню до гривні. Наприклад, пари значень можуть мати наступний вигляд: $(2, 25.7)$, $(1, 25.5)$ тощо.

Варіант 4: ... значення температури навколишнього середовища із прив'язкою до днів тижня. Тобто $n = 1, 2, \dots, 7$ – Ох-координати вузлових точок – порядкові номери днів тижня. У якості Оу-координат узяти безпосередньо значення температури середовища.

Наприклад, пари значень можуть мати наступний вигляд: $(3, 22.1)$, $(1, 17.2)$ тощо.

Варіант 5: ... значення кількості одиниць певної продукції, виробленої деяким підприємством за тиждень. Тобто $n = 1, 2, \dots, 7$ – Ох-координати вузлових точок – порядкові номери днів тижня. У якості Оу-координат узяти безпосередньо значення кількості одиниць продукції. Наприклад, пари значень можуть мати наступний вигляд: $(1, 122)$, $(4, 99)$ тощо.

Варіант 6: ... значення кількості олімпійських нагород, здобутих національними збірними 10 країн. Тобто $n = 1, 2, \dots, 10$ – Ох-координати вузлових точок – порядкові номери країн-учасниць Олімпійських ігор. У якості Оу-координат узяти значення кількості нагород – золотих, срібних та бронзових разом. Наприклад, пари значень можуть мати наступний вигляд: $(1, 121)$, $(3, 70)$ тощо.

Варіант 7: ... значення кількості адміністративних правопорушень, скоєних у певній області країни за місяць. Тобто $n = 1, 2, \dots, 12$ – Ох-координати вузлових точок – порядкові номери місяців року. У якості Оу-координат узяти значення кількості правопорушень, скоєних за місяць. Наприклад, пари значень можуть мати наступний вигляд: $(1, 292)$, $(4, 334)$ тощо.

Варіант 8: ... значення кількості населення (у млн.) у певних 10 країнах. Тобто $n = 1, 2, \dots, 10$ – Ох-координати вузлових точок – порядкові номери країн. У якості Оу-координат узяти значення кількості населення (у млн.). Наприклад, пари значень можуть мати наступний вигляд: $(1, 42)$, $(2, 143)$ тощо.

Варіант 9: ... значення температури води Чорного моря із прив'язкою до місяця року. Тобто $n = 1, 2, \dots, 12$ – Ох-координати вузлових точок – порядкові номери місяців року. У якості Оу-координат узяти безпосередньо значення температури води. Наприклад, пари значень можуть мати наступний вигляд: $(7, 19.3)$, $(8, 22.2)$ тощо.

Варіант 10: ... значення кількості працівників певного підприємства, яким за місяць призначено премію. Тобто $n = 1, 2, \dots, 12$

– Ох-координати вузлових точок – порядкові номери місяців року. У якості Оу-координат узяти значення кількості працівників, яким призначено премію. Наприклад, пари значень можуть мати наступний вигляд: $(1, 325)$, $(2, 314)$ тощо.

1.3 Зміст звіту

1.3.1 Титульний лист.

1.3.2 Мета роботи.

1.3.3 Вихідні данні згідно варіанту, відсортований масив, графіки, одержані в результаті вирішення задач інтерполяції, екстраполяції та апроксимації, а також аналітичне представлення поліному, одержаного при вирішенні задачі апроксимації (із зазначенням коефіцієнтів поліному, а також коефіцієнту детермінації).

1.3.4 Відповіді на контрольні питання.

1.4 Контрольні питання

1.4.1 Визначення поняття апроксимації.

1.4.2 Визначення поняття інтерполяції.

1.4.3 Формалізація задачі інтерполяції.

1.4.4 Визначення поняття екстраполяції.

1.4.5 Приклади типових сценаріїв вирішення задачі інтерполяції.

1.4.6 Кусково-лінійна інтерполяція. Переваги і недоліки.

1.4.7 Поліноміальна інтерполяція. Переваги і недоліки.

1.4.8 Сплайн-інтерполяція. Переваги і недоліки.

1.4.9 Призначення коефіцієнту детермінації.

ЛАБОРАТОРНА РОБОТА №2

2 МЕТОДИ РОЗВ'ЯЗАННЯ СЛАР

Мета роботи – набути навичок використання чисельних методів розв'язання систем лінійних алгебраїчних рівнянь (СЛАР).

2.1 Теоретичні відомості

СЛАР в загальному випадку є системою з m лінійних рівнянь із n невідомими:

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \dots + a_{1n} \cdot x_n = b_1, \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \dots + a_{2n} \cdot x_n = b_2, \\ \dots \\ a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \dots + a_{mn} \cdot x_n = b_m, \end{cases} \quad (2.1)$$

де $b_i (i=1..m)$ – вільні коефіцієнти – елементи вектора правих частин;

$x_j (j=1..n)$ – елементи вектора невідомих – шуканих рішень системи;

a_{ij} – елементи матриці коефіцієнтів при невідомих.

Вираз (2.1) також може бути представлений у матричному вигляді:

$$A \cdot \bar{x} = \bar{b}, \quad (2.2)$$

де $A = (a_{ij})_{m \times n}$ – матриця коефіцієнтів при невідомих;

$\bar{x} = (x_1 \dots x_n)^T$ – вектор невідомих;

$\bar{b} = (b_1 \dots b_m)^T$ – вектор правих частин.

Існують прямі та ітераційні методи розв'язання СЛАР.

2.1.1 Прямі методи

До прямих відносять, зокрема, наступні методи: метод Крамера, метод Гауса, метод оберненої матриці (матричний метод) та інші.

Теорема Крамера. Якщо визначник Δ системи N лінійних алгебраїчних рівнянь з N невідомими відмінний від нуля ($\Delta \neq 0$), то ця система має єдиний розв'язок, який знаходиться за формулами Крамера:

$$x_1 = \Delta_1 / \Delta, x_2 = \Delta_2 / \Delta, \dots, x_N = \Delta_N / \Delta, \quad (2.3)$$

Δ_j ($j=1,2,\dots,N$) - визначники, утворений із заміною j -го стовпця, стовпцем із вільних членів.

Якщо $\Delta=0$, а хоча б один з $\Delta_1, \Delta_2, \dots, \Delta_N$ відмінний від нуля, то СЛАР розв'язків немає. Якщо ж $\Delta_j=0$ ($j=1,2,\dots,N$), то СЛАР має безліч розв'язків.

Метод Гауса розв'язання системи лінійних алгебраїчних рівнянь полягає в послідовному виключенні змінних і перетворенні системи рівнянь (2.1) до трикутного (східчастого) вигляду:

$$\begin{cases} c_{11} \cdot x_1 + c_{12} \cdot x_2 + \dots + c_{1n} \cdot x_n = d_1, \\ c_{22} \cdot x_2 + \dots + c_{2n} \cdot x_n = d_2, \\ \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ c_{nn} \cdot x_n = d_n, c_{kk} = 1, (k = 1, 2, \dots, n). \end{cases} \quad (2.4)$$

Припустимо, що в системі коефіцієнт при першому елементі відмінний від нуля $a_{11} \neq 0$. Якщо ця умова не виконується, то на перше місце переносимо рівняння, яке її задовольняє.

За допомогою першого рівняння виключимо x_1 із решти рівнянь. Для цього ділять перший рядок на a_{11} , позначимо це

$$a_{1k}^1 = \frac{a_{1k}}{a_{11}}, (k = 0, 1, 2, \dots, n). \quad (2.5)$$

Далі від другого рядка віднімаємо перший рядок, помножений на a_{21} , від третього перший рядок, помножений на a_{31} , і так далі до останнього рядка.

Одержимо таблицю коефіцієнтів:

$$\begin{array}{cccccc}
 x_1 & x_2 & \dots & x_n & 1 \\
 a_{11}^1 & a_{12}^1 & \dots & a_{1n}^1 & b_1^1 \\
 0 & a_{22}^1 & \dots & a_{2n}^1 & b_2^1 \\
 \dots & \dots & \dots & \dots & \dots \\
 0 & a_{n2}^1 & \dots & a_{nn}^1 & b_n^1
 \end{array} \cdot \quad (2.6)$$

Для невідомих $x_2, \dots, x_n$, маємо систему $n-1$ рівнянь. Виконуючи, як і раніше, виключимо x_2 з усіх рівнянь, починаючи з третього. Для цього спочатку поділимо другий рядок на a_{22}^1 .

Якщо коефіцієнт $a_{22}^1 = 0$, то переставимо рівняння так, щоб виконувалася умова $a_{22}^1 \neq 0$. Позначивши

$$a_{2k}^2 = \frac{a_{2k}^1}{a_{22}^1}, (k = 2, \dots, n), b_2^2 = \frac{b_2^1}{a_{22}^1}, \quad (2.7)$$

Потім від третього рядка віднімемо другий рядок, помножений на a_{32}^1 ; від четвертого рядка віднімемо другий рядок, помножений на a_{42}^1 і т.д. Продовжуючи процес виключення невідомих отримаємо

$$\begin{array}{ccccccc}
 x_1 & x_2 & x_3 & \dots & x_{n-1} & x_n & 1 \\
 a_{11}^1 & a_{12}^1 & a_{13}^1 & \dots & a_{1,n-1}^1 & a_{1n}^1 & b_1^1 \\
 0 & a_{22}^2 & a_{23}^2 & \dots & a_{2,n-1}^2 & a_{2n}^2 & b_2^2 \\
 0 & 0 & a_{33}^3 & \dots & a_{3,n-1}^3 & a_{3n}^3 & b_3^3 \\
 \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
 0 & 0 & 0 & \dots & a_{n-1,n-1}^{n-1} & a_{n-1,n}^{n-1} & b_{n-1,0}^{n-1} \\
 0 & 0 & 0 & \dots & 0 & a_{nn}^n & b_{n0}^n
 \end{array} \cdot \quad (2.8)$$

Таблицю коефіцієнтів при невідомих, яка має вигляд верхньої трикутної матриці. Всі елементи на головній діагоналі рівні одиниці $a_{kk}^1 = 1$. Запишемо відповідну систему рівнянь:

$$\left\{ \begin{array}{l} a_{11}^1 x_1 + a_{12}^1 x_2 + a_{13}^1 x_3 + \dots + a_{1,n-1}^1 x_{n-1} + a_{1n}^1 x_n = b_1^1, \\ a_{22}^2 x_2 + a_{23}^2 x_3 + \dots + a_{2,n-1}^2 x_{n-1} + a_{2n}^2 x_n = b_2^2, \\ a_{33}^3 x_3 + \dots + a_{3,n-1}^3 x_{n-1} + a_{3n}^3 x_n = b_3^3, \\ \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ a_{n-1,n-1}^{n-1} x_{n-1} + a_{n-1,n}^{n-1} x_n = b_{n-1}^{n-1}, \\ a_{n-1,n}^{n-1} x_n = b_n^n. \end{array} \right. \quad (2.9)$$

Перехід від першої системи рівнянь (2.1) до останньої (2.9) називається прямим ходом методу Гауса. Обернений хід методу Гауса починається з останньої системи рівнянь. Її розв'язують з кінця до початку. З останнього рівняння знаходять x_n . Підставивши це значення в передостаннє – знаходять x_{n-1} і т.д. З першого рівняння знаходять x_1 .

Матричний метод. Систему рівнянь запишемо у вигляді матричного рівняння (2.2). Якщо квадратна матриця A має відмінний від нуля визначник Δ , то для неї існує обернена A^{-1} . Помноживши зліва в рівнянні 2.2 на A^{-1} , одержимо

$$A^{-1} \cdot A \cdot \bar{x} = A^{-1} \cdot \bar{b}, \quad (2.10)$$

Враховуючи, що добуток оберненої матриці на саму матрицю дає одиничну $A^{-1} \cdot A = E$, а також формулу $E \cdot \bar{x} = \bar{x}$, одержимо матричний розв'язок системи

$$\bar{x} = A^{-1} \cdot \bar{b}, \quad (2.11)$$

Розглянемо приклад розв'язання СЛАР методами Крамера, Гауса та оберненої матриці у середовищі *Scilab*.

Нехай маємо наступну систему:

$$\begin{cases} 2 \cdot x_1 + 2 \cdot x_2 - x_3 + x_4 = 4, \\ 4 \cdot x_1 + 3 \cdot x_2 - x_3 + 2 \cdot x_4 = 6, \\ 8 \cdot x_1 + 5 \cdot x_2 - 3 \cdot x_3 + 4 \cdot x_4 = 12, \\ 3 \cdot x_1 + 3 \cdot x_2 - 2 \cdot x_3 + 2 \cdot x_4 = 6. \end{cases} \quad (2.12)$$

Метод Крамера:

– задати матрицю коефіцієнтів A та вектор правих частин $\bar{b}$:

$$A = [2, 2, -1, 1; 4, 3, -1, 2; 8, 5, -3, 4; 3, 3, -2, 2]; \\ b = [4; 6; 12; 6];$$

– сформувати допоміжні матриці A_1, A_2, A_3 із заміщеними стовпцями:

$$A_1 = A; \quad A_1(:, 1) = b; \\ A_2 = A; \quad A_2(:, 2) = b; \\ A_3 = A; \quad A_3(:, 3) = b; \\ A_4 = A; \quad A_4(:, 4) = b;$$

– розрахувати основний та допоміжний визначники:

$$D = \det(A); \\ d(1) = \det(A_1); \\ d(2) = \det(A_2); \\ d(3) = \det(A_3); \\ d(4) = \det(A_4);$$

– одержати вектор невідомих $\bar{x}$:

$$x = d / D;$$

– перевірити правильність виконаних обчислень:

$$P = A * x - b.$$

Метод Гауса:

– задати матрицю коефіцієнтів A та вектор правих частин $\bar{b}$;

– сформувати трапецієподібну матрицю з матриці A та вектору

$\bar{b}$ – прямий хід:

$$C = \text{rref}([A, b]);$$

– визначити розмірність одержаної розширеної матриці C :

$$[m, n] = \text{size}(C),$$

де $m=4$ – кількість рядків, $n=5$ – кількість стовпців;

– виокремити останній стовпець із розширеної матриці C , який і буде шуканим розв’язком:

$$x = C(:, n)$$

Матричний метод:

– задати матрицю коефіцієнтів A та вектор правих частин $\bar{b}$;

– одержати вектор-розв’язок:

$$x = \text{inv}(A) * b.$$

2.1.2 Ітераційні методи

Метод простої ітерації [5]. Метод полягає в наступному:

– система (2.12) представляється в ітеративному форматі:

$$\bar{x}^{(k)} = A' \cdot \bar{x}^{(k-1)} + \bar{b}', \quad (2.13)$$

де $k = 1, 2, \dots$ – порядковий номер ітерації;

$\bar{x}$ – шуканий вектор;

A' – модифікована матриця коефіцієнтів при невідомих;

$\bar{b}'$ – модифікований вектор правих частин.

Згідно із ітеративним форматом (2.13), розрахунок компонентів шуканого вектору $\bar{x}$ на кожній ітерації здійснюється наступним чином:

$$x_j^{(k)} = - \sum_{\substack{j=1 \\ j \neq i}}^n \frac{a_{ij}}{a_{ii}} x_j^{(k-1)} + \frac{b_i}{a_{ii}}. \quad (2.14)$$

В якості критерію зупинки ітераційного процесу зручно використовувати наступний вираз:

$$\left\| \bar{x}^{(k)} - \bar{x}^{(k-1)} \right\| \leq \varepsilon, \quad (2.15)$$

де $\bar{x}^{(k)}$ – результат обчислень поточної ітерації;

$\bar{x}^{(k-1)}$ – попередньої ітерації;

ε – задана точність обчислень;

$\left\| \bar{x}^{(k)} - \bar{x}^{(k-1)} \right\|$ – норма вектору-різниці, яка може бути розрахована, наприклад, за наступною формулою:

$$\left\| \bar{x}^{(k)} - \bar{x}^{(k-1)} \right\| = \sum_{j=1}^n \left| x_j^{(k)} - x_j^{(k-1)} \right| \quad (2.16)$$

Перед запуском ітераційного процесу необхідно переконатися у збіжності методу простих ітерацій. Щоб виконувалася достатня умова збіжності, справедливою має бути наступна нерівність:

$$q < 1, \quad (2.17)$$

де $q = \|A'\|$ – норма матриці A' :

$$\|A'\| = \max_j \left\{ \sum_{i=1}^m |a'_{i,j}| \right\}. \quad (2.18)$$

Отже, алгоритм розв'язання СЛАР методом простих ітерацій можна представити наступним чином:

- сформувати матрицю A та вектор правих частин $\bar{b}$;
- перетворити матричний формат (2.2) до ітераційного формату (2.13);
- перевірити метод на збіжність;
- запустити ітераційний процес, який продовжувати доти, доки не виконуватиметься нерівність (2.15).

Для одержання апріорної оцінки кількості ітерацій k^* , що знадобляться для одержання розв'язку із заданою точністю ε , можна скористатися наступним виразом:

$$k^* \geq \frac{\ln \left(\frac{\varepsilon \cdot (1-q)}{\left\| \frac{x^{(1)} - x^{(0)}}{x} \right\|} \right)}{\ln q}, \quad (2.19)$$

де k^* - слід округляти до більшого цілого. Як правило така аналітична оцінка є дещо завищеною, але вона дає змогу переконатися у достатності наявних часових/обчислювальних ресурсів.

Розглянемо метод простих ітерацій на прикладі. Нехай маємо наступну систему, яку необхідно вирішити із заданою точністю $\varepsilon = 10^{-2}$ [6]:

$$\begin{cases} 4x_1 + x_2 - x_3 = 0, \\ x_1 - 5x_2 + x_3 = 1, \\ x_1 - x_2 + 4x_3 = -1. \end{cases} \quad (2.20)$$

Для одержання представлення СЛАР в ітераційному форматі (2.13), скористаємося виразом (2.14). В результаті цього одержимо наступну систему:

$$\begin{cases} x_1^{(k+1)} = 0 \cdot x_1^{(k)} - 0.25 \cdot x_2^{(k)} + 0.25 \cdot x_3^{(k)} + 0, \\ x_2^{(k+1)} = 0.2 \cdot x_1^{(k)} + 0 \cdot x_2^{(k)} + 0.2 \cdot x_3^{(k)} - 0.2, \\ x_3^{(k+1)} = -0.25 \cdot x_1^{(k)} + 0.25 \cdot x_2^{(k)} + 0 \cdot x_3^{(k)} - 0.25, \end{cases} \quad (2.21)$$

для якої

$$A' = \begin{pmatrix} 0 & -0.25 & 0.25 \\ 0.2 & 0 & 0.2 \\ -0.25 & 0.25 & 0 \end{pmatrix}, \quad \bar{b}' = \begin{pmatrix} 0 \\ -0.2 \\ -0.25 \end{pmatrix}. \quad (2.22)$$

Розрахуємо норму матриці A' :

$$q = \|A'\| = \max\{|0.2| + |-0.25|; |-0.25| + |0.25|; |0.25| + |0.2|\} = 0.5.$$

Для одержання представлення СЛАР в ітераційному форматі у середовищі *Scilab* використовуємо наступний код, який зберігаємо у файл *1_varN.sce*:

```
A=[4 1 -1; 1 -5 1; 1 -1 4];
B=[0;1;-1];
b=zeros(3,1);
a=zeros(3,3);
for i=1:3
    b(i)=B(i)/A(i,i);
    a(i,i)=0;
        for j=1:3
            if i~=j
                a(i,j)=-A(i,j)/A(i,i);
            end
        end
    end
end
q=norm(a);
```

Після запуску файлу *1_varN.sce* маємо наступне:

```
a  =
    0.      - 0.25    0.25
    0.2      0.      0.2
   - 0.25    0.25    0.
b  =
    0.
   - 0.2
   - 0.25
q  =
    0.4717737
```

Із нерівності $q < 1$ бачимо, що умова збіжності виконується. Отже можемо запускати ітераційний процес. Для цього в якості початкового (нульового) наближення $\bar{x}^{(0)}$ прийнемо модифікований вектор правих частин $\bar{b}'$ (2.22), тобто $\bar{x}^{(0)} = \bar{b}'$.

Тоді на першій ітерації одержимо такі компоненти вектору $\bar{x}^{(1)}$:

$$x_1^{(1)} = -0.25 \cdot (-0.2) + 0.25 \cdot (-0.25) = -0.0125;$$

$$x_2^{(1)} = 0.2 \cdot (-0.25) - 0.2 = -0.25;$$

$$x_3^{(1)} = 0.25 \cdot (-0.2) - 0.25 = -0.30.$$

$$\text{Отже } \bar{x}^{(1)} = (-0.0125, -0.25, -0.30)^T.$$

Для вирішення СЛАР методом простої ітерації у середовищі *Scilab* створимо функцію *iter*, куди будемо передавати СЛАР в ітераційному форматі (одержані матриці *a* та *b*) й точності численного рішення ε . Повертати функція *iter* буде $\bar{x}$ – шуканий вектор, а також кількість ітерацій *k*. Файл збережемо як *iret.sce*:

```
function [X,k]=iter(A,B,eps)
delta=eps*(1-norm(A))/norm(A);
X0=B;
X1=A*X0+B;
k=1;
    while norm(X1-X0)>delta
        X0=X1;
        X1=A*X0+B;
        k=k+1;
    end;
X=X1;
```

Далі для одержання результату вирішення СЛАР методом простої ітерації заданою точності численного рішення $\varepsilon = 10^{-1}$ у середовищі *Scilab*, виконуємо наступне:

```
exec('l_varN.sce',-1)
exec('iter.sce',-1)
[X,k]=iter(a,b,10^-1)
```

Одержимо:

```
X  =
    - 0.0125
    - 0.25
    - 0.3
k   =
    1.
```

При точності численного рішення $\varepsilon = 10^{-2}$ одержимо наступне:

```
[X,k]=iter(a,b,10^-2)
X  =
    - 0.0117188
    - 0.264375
    - 0.3125
k   =
    3.
```

2.2 Завдання до лабораторної роботи

Номер варіанту визначається студентом за номером комп'ютера. У випадку, якщо значення номеру комп'ютера перевищує значення номеру варіанту, номер варіанту визначається як залишок від ділення номеру комп'ютера на 10, плюс один.

2.2.1 Вирішити СЛАР згідно варіанту (табл. 2.1) методами Крамера, Гауса та оберненої матриці у середовищі *Scilab* (приклад у п. 2.1.1.).

2.2.2* Вирішити СЛАР згідно варіанту (табл. 2.1) методами Крамера, Гауса та оберненої матриці у середовищі *Dev-C++* (у додатку Б подано програмний код необхідних функцій). Порівняти результати обчислень із попередньо одержаними результатами.

2.2.3 Розв'язати СЛАР згідно варіанту (табл. 2.1) для заданої точності $\varepsilon = 10^{-2}, 10^{-3}, \dots, 10^{-6}$ методом простих ітерацій у середовищі *Scilab* (приклад у п. 2.1.2.).

2.2.4* Вирішити СЛАР згідно варіанту (табл. 2.1) для заданої точності $\varepsilon = 10^{-2}, 10^{-3}, \dots, 10^{-6}$ методом простих ітерацій у середовищі *Dev-C++* (у додатку Б подано програмний код необхідних функцій).

Порівняти результати обчислень із попередньо одержаними результатами.

Таблиця 2.1 – Варіанти до виконання завдань

№	СЛАР	№	СЛАР
1	$\begin{cases} 18 \cdot x_1 + 3 \cdot x_2 - 10 \cdot x_3 = 84, \\ -3 \cdot x_1 + 10 \cdot x_2 - 5 \cdot x_3 = -32, \\ x_1 - 3 \cdot x_2 + 8 \cdot x_3 = 69 \end{cases}$	6	$\begin{cases} 15 \cdot x_1 - 4 \cdot x_2 + 6 \cdot x_3 = 100, \\ 4 \cdot x_1 + 8 \cdot x_2 - x_3 = 72, \\ -8 \cdot x_1 - 4 \cdot x_2 + 18 \cdot x_3 = -84. \end{cases}$
2	$\begin{cases} 17 \cdot x_1 - 3 \cdot x_2 - 9 \cdot x_3 = -165, \\ -4 \cdot x_1 + 16 \cdot x_2 + 8 \cdot x_3 = 120, \\ -8 \cdot x_1 + 3 \cdot x_2 + 18 \cdot x_3 = 165. \end{cases}$	7	$\begin{cases} 18 \cdot x_1 - 6 \cdot x_2 - 9 \cdot x_3 = 69, \\ -6 \cdot x_1 + 14 \cdot x_2 - 4 \cdot x_3 = -170, \\ 9 \cdot x_1 + 7 \cdot x_2 + 26 \cdot x_3 = 239. \end{cases}$
3	$\begin{cases} 7 \cdot x_1 + 3 \cdot x_2 = 31, \\ -4 \cdot x_1 + 17 \cdot x_2 - 10 \cdot x_3 = -69, \\ 6 \cdot x_1 + 2 \cdot x_2 + 13 \cdot x_3 = 117. \end{cases}$	8	$\begin{cases} 9 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 = -60, \\ -2 \cdot x_1 + 14 \cdot x_2 - 3 \cdot x_3 = -51, \\ 2 \cdot x_1 - 6 \cdot x_2 + 11 \cdot x_3 = 99. \end{cases}$
4	$\begin{cases} 21 \cdot x_1 - 3 \cdot x_2 - 8 \cdot x_3 = 127, \\ 7 \cdot x_1 + 14 \cdot x_2 = 91, \\ 8 \cdot x_1 - 2 \cdot x_2 + 14 \cdot x_3 = 166. \end{cases}$	9	$\begin{cases} 16 \cdot x_1 - 4 \cdot x_2 - 8 \cdot x_3 = -80, \\ -10 \cdot x_1 + 24 \cdot x_2 - 8 \cdot x_3 = -6, \\ 2 \cdot x_1 + x_2 + 10 \cdot x_3 = -2. \end{cases}$
5	$\begin{cases} 19 \cdot x_1 - 7 \cdot x_2 + 4 \cdot x_3 = 113, \\ 11 \cdot x_2 + 7 \cdot x_3 = 5, \\ -3 \cdot x_1 - 6 \cdot x_2 + 16 \cdot x_3 = 127. \end{cases}$	10	$\begin{cases} 16 \cdot x_1 + 3 \cdot x_2 - 2 \cdot x_3 = 66, \\ 2 \cdot x_1 + 13 \cdot x_2 - 8 \cdot x_3 = 86, \\ -x_1 - 5 \cdot x_2 + 7 \cdot x_3 = -22. \end{cases}$

2.3 Зміст звіту

2.3.1 Титульний лист.

2.3.2 Мета роботи.

2.3.3 Scilab-код для завдань згідно варіанту (із коментарями).

2.3.4 За наявності – C++ код для завдань згідно варіанту (позначених символом *).

2.3.5 Відповіді на контрольні питання.

2.4 Контрольні питання

2.4.1 Прямі методи розв'язання СЛАР. Стисла характеристика. Обчислювальна складність.

2.4.2 Метод простих ітерацій. Ітераційний формат СЛАР.

2.4.3 Метод простих ітерацій. Умова зупинки ітераційного процесу.

2.4.4 Метод простих ітерацій. Апріорна оцінка кількості ітерацій.

2.4.5 Апріорна та апостеріорна оцінки кількості ітерацій. Порівняльна характеристика.

2.4.6 Ітераційні методи розв'язання СЛАР. Залежність кількості ітерацій від заданої точності ε .

2.4.7 Прокоментувати розбіжність значень апріорної оцінки кількості ітерацій та фактичної кількості – при вирішенні СЛАР методом простих ітерацій.

2.4.8 Порівняльна характеристика прямих та ітераційних методів розв'язання СЛАР. Переваги та недоліки.

ЛАБОРАТОРНА РОБОТА №3 З БАЗОВІ АЛГОРИТМИ

Мета роботи – набути практичних навичок реалізації та використання базових алгоритмів пошуку та сортування.

3.1 Теоретичні відомості

Алгоритми пошуку та сортування відносять до базових алгоритмів – таких, які зазвичай використовуються у складі більш складних (комплексних) алгоритмів.

3.1.1 Алгоритми пошуку

Існують алгоритми лінійного та бінарного пошуків. Алгоритм лінійного пошуку є найпростішим. Його асимптотична складність становить $O(n)$, де n – розмір входу – кількість елементів масиву даних, в якому здійснюється пошук. Робота алгоритму (лістинг 3.1) полягає у послідовному співставленні шуканого образу (значення) із фактичними значеннями елементів масиву, доки не буде досягнуто однієї з наступних умов:

- шуканий елемент знайдено на i -му кроці;
- співставлення здійснено з усіма елементами масиву, але шуканого елементу знайдено не було.

Лістинг 3.1 – Реалізація алгоритму лінійного пошуку

```
//arr - масив, у якому здійснюється пошук
//key - шуканий елемент
//n - розмір масиву
int lSearch(int arr[], int key, int n)
{ for (int i = 0; i < n; i++)
    if (arr[i] == key) {
        cout<<"found: arr["<<i<<"]="<<arr[i]<<endl;
        return i;
    }
return -1;
}
```

Функція `lSearch` повертає індекс (порядковий номер) знайденого елементу або `-1` – у випадку, якщо шуканий елемент так і не було знайдено.

Схему перевірки розглянутого алгоритму подано у лістингу 3.2.

Лістинг 3.2 – Використання алгоритму лінійного пошуку

```
#include <iostream>
using namespace std;
int main() {
    int arr[] = {1,2,3,4,5};          // вихідний масив
    int n;                             // шукане значення
    cout<<"enter the value to find...\n";
    cin>>n;
    lSearch(arr,n,5); // виклик функції
    system("pause"); return 0;}
```

На відміну від алгоритму лінійного пошуку, алгоритм бінарного пошуку характеризується кращою асимптотичною складністю $O(\log n)$, однак для його використання елементи масиву необхідно попередньо відсортувати, наприклад за зростанням.

Алгоритм бінарного пошуку (лістинг 3.3):

- знайти центральний елемент відсортованого масиву;
- якщо центральний елемент є шуканим, завершити роботу алгоритму, інакше – перейти до наступного кроку;
- якщо шукане значення менше значення центрального елемента, перейти до першого кроку для лівого підмасиву, інакше – до наступного кроку;
- якщо шукане значення більше значення центрального елемента, перейти до першого кроку для правого підмасиву.

Лістинг 3.3 – Реалізація алгоритму бінарного пошуку

```
//arr - масив, у якому здійснюється пошук
//key - шуканий елемент
//l - індекс початкового елемента відсортованого масиву
//r - індекс останнього елемента відсортованого масиву
int bSearch(int arr[], int key, int l, int r)
{
```

```

int m=0; //індекс центрального елемента масиву
while (true)
{
    m=(l+r)/2; // крок 1
    if(key == arr[m]) { // крок 2
        cout<<"found: arr["<<m<<"]="<<arr[m]<<endl;
        return m; // крок 2
    }
    else if (key<arr[m]) r=m-1; // крок 3
    else if (key>arr[m]) l=m+1; // крок 4
    if (l > r) return -1; // якщо не знайшли
}
}

```

Виклик функції `bSearch`, у відповідності до лістингу 3.3, здійснюється вже із чотирма аргументами:

```

//arr - масив, у якому здійснюється пошук
//key - шуканий елемент
//0 - фактичний індекс початкового елемента
//4 - фактичний індекс останнього елемента
bSearch(arr, key, 0, 4);

```

3.1.2 Алгоритми сортування

Виділяють наступні групи алгоритмів сортування: прості та складні алгоритми.

До простих можна віднести такі алгоритми сортування:

- бульбашковий алгоритм (*bubble sort*);
- сортування вибором (*selection sort*);
- сортування вставкою (*insertion sort*).

Усі зазначені алгоритми є простими з позиції їх реалізації – у цьому полягає їх перевага у порівнянні з іншими (складними алгоритмами). Ключовий недолік простих алгоритмів – їх неполіноміальна асимптотична (обчислювальна) складність. Асимптотичні оцінки зазначених простих алгоритмів в середньому складають $O(n^2)$, що є незадовільним показником.

На відміну від простих алгоритмів сортування, представники групи складних алгоритмів мають кращу асимптотичну складність – в

середньому $O(n \cdot \log n)$. Типовими представниками складних алгоритмів є наступні приклади:

- алгоритм швидкого сортування (*quick sort*);
- алгоритм пірамідального сортування (*heap sort*).

Через відносну компактність реалізації алгоритму швидкого сортування та його високу ефективність, в даній роботі у якості представника складних алгоритмів буде розглядатися саме він.

При роботі із алгоритмами сортування, з метою зменшення обсягу програмного коду, створимо спеціальну функцію для обміну елементів значеннями (лістинг 3.4).

Лістинг 3.4 – Програмна реалізація функції обміну значеннями елементів масиву

```
void iswap(int &a, int &b)
{
    int tmp(a);
    a = b;
    b = tmp;
}
```

Розглянемо реалізації зазначених алгоритмів сортування (лістинги 3.5 – 3.8).

Лістинг 3.5 – Програмна реалізація бульбашкового алгоритму сортування

```
void bubbleSort(int arr[], int n)
{
    for(int j=n-1; j>=1; j--)
        for(i=0; i<j; ++i)
            if(arr[i]>arr[i+1])
                iswap(arr[i], arr[i+1]);
}
```

Лістинг 3.6 – Програмна реалізація алгоритму сортування вибором

```
void sSort(int arr[], int n)
```

```

{
    int min;
    for(int i=0, j; i<n-1; i++)
    {
        min = i;
        for(j = i+1; j<n; j++)
            if(arr[j]<arr[min])
                min=j;
        iswap(arr[i],arr[min]);
    }
}

```

Ключовою відмінністю алгоритму сортування вибором (лістинг 3.6) від бульбашкового алгоритму (лістинг 3.5) є потенційна кількість обмінів значеннями елементів масиву. У випадку бульбашкового алгоритму ця кількість $\leq n^2$, оскільки функцію `iswap` поміщено у тіло внутрішнього циклу. У випадку алгоритму сортування вибором – $\leq n$, оскільки функція `iswap` знаходиться у тілі зовнішнього циклу. Це є суттєвою перевагою алгоритму сортування вибором з позиції зменшення супутніх часових витрат. Хоча асимптотичні оцінки названих алгоритмів однакові – $O(n^2)$, через зазначену особливість алгоритм сортування вибором потребує суттєво менших часових витрат.

Лістинг 3.7 – Програмна реалізація алгоритму сортування вставкою

```

void iSort(int arr[], int n)
{
    int tmp;
    for(int i=1, j; i<n; i++)
    {
        tmp = arr[i];
        j=i;
        while(j>0 && arr[j-1]>=tmp)
        {
            arr[j] = arr[j-1];
            j--;
        }
        arr[j] = tmp;
    }
}

```

Лістинг 3.8 – Програмна реалізація алгоритму швидкого сортування

```
void qSort(int arr[], int l, int r)
{ //l – ліва межа діапазону; спочатку – 0
  //r – права межа діапазону; спочатку – n-1
  int p = arr[(r+l)/2];
  int i=l, j=r;
  do
  {
    while(arr[i]<p) i++;
    while(arr[j]>p) j--;
    if(i<=j)
    { swap(arr[i], arr[j]);
      i++;
      j--;
    }
  }
  while(i<=j);
  if(i<r) qSort(arr, i, r);
  if(j>l) qSort(arr, l, j);
}
```

3.2 Завдання до лабораторної роботи

Виконання роботи орієнтоване на апостеріорне оцінювання обчислювальної асимптотичної ефективності наведених вище алгоритмів. Це означає, що вимірюваним показником виступатиме час, витрачений на роботу програмної реалізації алгоритму.

Робота виконується у середовищах *Dev-C++* та *TableCurve 2D*.

Зауваження: налаштування компілятора середовища *Dev-C++5.11*:

– *Tools -> Compiler Options... -> Settings -> Code Generation ->* опцію *Language standard (-std)* встановити в *ISO C++11*;

– *Tools -> Compiler Options... -> Programs ->* у полі *g++:* дописати *-std=c++11*, тобто разом має бути: *g++.exe -std=c++11*.

Завдання до виконання:

3.2.1 Порівняти роботу алгоритмів лінійного та бінарного пошуків. Для цього слід здійснити наступні кроки (лістинг 3.9):

– виділити пам'ять під динамічний масив (у якості розміру ввести значення 10^7);

- заповнити елементи масиву випадковими значеннями;
- виміряти час лінійного пошуку випадкового елементу масиву;
- відсортувати масив значень за зростанням;
- виміряти час бінарного пошуку того самого елементу;
- вивільнити пам'ять, виділену під динамічний масив.

Для заміру часових витрат на роботу алгоритмів використовується бібліотека `<ctime>` та відповідна функція `clock`, що дозволяє одержувати значення часових витрат в умовних одиницях процесорного часу – тіках (*clock ticks*). Для одержання значень часу у секундах необхідно виконувати ділення на константу `CLOCKS_PER_SEC`.

Для виконання завдання слід створити C++ проект на основі лістингів 3.1, 3.3, 3.8 та 3.9.

Лістинг 3.9 – Робота з алгоритмами пошуку

```
#include <iostream>
#include <ctime> //для заміру часу
#include <cstdlib>
#include <random> //для генератору вип. чисел
using namespace std;
int main() {
    int n; //розмір масиву
    cout<<"enter the size of array...\n";
    cin>>n; //ввести 10000000
    std::mt19937 gen; //створити змінну-генератор
    gen.seed(time(0)); //ініціалізувати генератор
    time_t start, stop; //для збереження часу
    int *arr = new int[n]; //виділити пам'ять під масив
    for(int i=0; i<n; i++)
        arr[i] = gen()%n; //ініціалізувати ел. масиву
    int key = gen()%n; //згенерувати шукане число
    cout<<"to find: "<<key<<endl;
    cout<<"linear search started...\n";
    start = clock(); //заміряти час до лінійного пошуку
    lSearch(arr, key, n);
    stop = clock() - start; //... після лінійного пошуку
    printf("time elapsed: %.3f s\n", \
```

```

float((float)stop/CLOCKS_PER_SEC));
cout<<"binary search started...\n";
cout<<"sorting started (please, wait a bit)...\n";
qSort(arr, 0, n-1); // здійснення швидкого сортування
cout<<"sorting stopped.\n";
start = clock();
bSearch(arr, key, 0, n-1);
stop = clock() - start;
printf("time elapsed: %.3f s\n", \
float((float)stop/CLOCKS_PER_SEC));
system("pause");
delete[] arr; //вивільнити пам'ять
return 0;}

```

Проаналізувати та пояснити одержані результати.

3.2.2 Спробувати використати замість швидкого сортування (функція `qSort`) будь-який простий алгоритм сортування (бульбашковий, вибору або вставки). Спробувати запустити на виконання модифіковану таким чином програму. **Зауваження:** для екстреного завершення роботи програми натиснути комбінацію клавіш *Ctrl+C*. Пояснити роботу програми.

3.2.3 Модифікувати код лістингу 3.9 таким чином, щоб у результаті роботи програми на консоль було виведено таблицю значень середніх арифметичних (по 5 замірах) часових витрат для лінійного та бінарного пошуків при $n = 10^4, 10^5, \dots, 10^7$. Тобто таблиця має містити три стовпці – n , $f_1(n)$, $f_2(n)$, де $f_1(n)$ – часові витрати на здійснення лінійного пошуку, а $f_2(n)$ – бінарного пошуку.

Таблиця-лістинг має бути оформлена відповідно до табл. 3.1, яку необхідно занести до звіту. Проаналізувати одержані результати.

Таблиця 3.1 – Шаблон лістингу результату виконання завдання 3

n	$f_1(n), c$	$f_2(n), c$
10^4	...	...
...	...	...
10^7	...	...

3.2.4 Виконати завдання згідно варіанту. Завдання виконується подібно до завдання 3.2.3, але замість алгоритмів лінійного та бінарного пошуків фігурувати мають алгоритми, задані згідно варіанту. При цьому значення n мають бути наступними: $n = 10^4, 2 \cdot 10^4, \dots, 4 \cdot 10^4$.

Варіант 1 (непарні номери ПК). Використовувати алгоритми бульбашкового сортування (лістинг 3.5) та сортування вибором (лістинг 3.6).

Варіант 2 (парні номери ПК). Використовувати алгоритми бульбашкового сортування (лістинг 3.5) та сортування вставкою (лістинг 3.7).

Проаналізувати одержані результати. Чи відповідають вони оцінкам асимптотичних складностей алгоритмів? Одержані результати, оформлені у табличному вигляді, занести до звіту.

3.2.5 Загальне завдання. Аналогічно до виконаних завдань, дослідити алгоритм швидкого сортування (лістинг 3.8). Результатом також має бути таблиця, подібна до табл. 3.1. Однак вона повинна містити вже два стовпці (без стовпчика $f_2(n)$), оскільки алгоритм швидкого сортування має досліджуватися сам по собі. При цьому значення n слід задати наступними: $n = 10^6, 2 \cdot 10^6, \dots, 4 \cdot 10^6$.

Проаналізувати одержані результати.

3.2.6 Візуалізувати результати, одержані при виконанні завдань 3–5. Для цього слід виконати наступні кроки:

– результати виконання попередніх завдань 3.2.3–3.2.5 представити у вигляді п'яти окремих текстових *txt*-файлів, оформлених згідно наступного шаблону:

```
10000 0.3306
20000 1.6596
30000 4.7734
40000 10.0524
```

, де перший стовпець – значення n , другий стовпець – середні арифметичні значення відповідних часових витрат, s . При цьому елементи різних стовпців одного рядку обов'язково мають бути відділені один від одного пробілами.

Назви файлів мають бути наступними: *task_3_1.txt* (*task_3_2.txt*) – для алгоритмів лінійного (бінарного) пошуку; *task_4_1.txt* (*task_4_2.txt*) – для бульбашкового алгоритму (алгоритму вибору або вставки – згідно варіанту); *task_5.txt* – для алгоритму швидкого сортування;

– використати утиліту *TableCurve 2D* та виконати імпорт вихідних даних: *File -> Import...* (*Ctrl+I*); вказати потрібний текстовий файл у якості вихідних даних; у результаті буде побудовано графік, що являє собою результат кусково-лінійної інтерполяції на основі вихідних даних;

– обрати пункт меню *Process -> Curve-Fit Simple Equations* (*Ctrl+S*); натиснути кнопку *Graph Start*. У результаті буде побудовано графік поліноміальної апроксимації на основі вихідних даних. Змінити колір фону графіків на білий та занести їх до звіту. Відповідні апроксимуючі функції та коефіцієнти будуть подані на робочій області графіку. Чи підтверджують графіки асимптотичні складності?

3.3 Зміст звіту

3.3.1 Титульний лист.

3.3.2 Мета роботи.

3.3.3 Результати виконання завдань 3.2.3 – 3.2.5, оформлені у табличному вигляді.

3.3.4 Графіки, одержані у результаті виконання завдання 3.2.6. Усього має бути 5 графіків – по два – для завдань 3.2.3 і 3.2.4, один – для завдання 3.2.5.

3.3.5 Відповіді на контрольні питання.

3.4 Контрольні питання

3.4.1 Алгоритм лінійного пошуку.

3.4.2 Алгоритм бінарного пошуку.

3.4.3 Пояснити логарифмічну асимптотичну складність алгоритму бінарного пошуку.

3.4.4 Прокоментувати відмінність індексів знайдених елементів при лінійному та бінарному пошуках при виконанні завдання 1.

3.4.5 Переваги та недоліки простих алгоритмів сортування.

3.4.6 Переваги та недоліки складних алгоритмів сортування.

- 3.4.7 Бульбашковий алгоритм сортування. Переваги та недоліки.
- 3.4.8 Алгоритм сортування вибором. Переваги та недоліки.
- 3.4.9 Алгоритм сортування вставкою. Переваги та недоліки.
- 3.4.10 Алгоритм швидкого сортування. Переваги та недоліки.

ЛАБОРАТОРНА РОБОТА №4

4 МЕТОДИ РОЗВ'ЯЗАННЯ ДИФЕРЕНЦІЙНИХ РІВНЯНЬ

Мета роботи – набути навичок використання чисельних методів розв'язання диференціальних рівнянь у середовищі *SciLab*.

4.1 Теоретичні відомості

Диференціальні рівняння – це рівняння, в яких значення невідомої функції в певній точці зв'язується із значеннями її похідних різних порядків у той самій точці.

Диференціальним рівнянням n -го порядку є рівняння наступного вигляду:

$$f(x, y, y', \dots, y^{(n)}) = 0, \quad (4.1)$$

де x – незалежна змінна; $y(x)$ – невідома функція; n – порядок рівняння.

Порядок диференціального рівняння – це найбільший порядок похідної у складі рівняння.

Рівняння вигляду (4.1) прийнято називати ЗДР – звичайними диференціальними рівняннями.

Рішенням диференціального рівняння є невідома функція $y(x)$, що перетворює рівняння вигляду (4.1) у тотожність. Тобто така функція на певному інтервалі $[a, b]$ має похідні порядку $1 \dots n$.

Процес вирішення ЗДР полягає у здійсненні інтегрування.

Розв'язання ЗДР пов'язане із вирішенням задач механіки, термодинаміки тощо.

Часто рівняння (4.1) подають у вигляді рівняння відносно старшої похідної:

$$y^n = f(x, y, y', \dots, y^{(n-1)}). \quad (4.2)$$

Формат (4.2) часто використовують при поступовому пониженні порядку рівняння (рівнянь). Це означає, що розв'язання ЗДР порядку n можна одержати шляхом вирішення n рівнянь першого порядку.

4.1.1 Чисельне розв'язання ЗДР

Необхідною умовою для здійснення чисельного інтегрування – чисельного вирішення ЗДР – є формулювання задачі Коші. Це означає, що необхідно задати початкові умови.

Нехай маємо ЗДР 1-го порядку $y' = f(x, y)$ та заданий інтервал значень незалежної змінної $[a, b]$: $x \in [a, b]$. Для чисельного розв'язання такого ЗДР здійснимо дискретизацію названого інтервалу із фіксованим кроком h . Для цього значення x представимо послідовністю табличних значень у межах інтервалу $[a, b]$, тобто $x_0 = a$, $x_1 = x_0 + h$, $x_2 = x_1 + h = x_0 + 2 \cdot h$, і т.д. У якості початкової умови приймемо, що $y(x_0) = y_0$, тобто задамо початкове наближення. У результаті цього кроку маємо недозаповнену таблицю пар значень (табл. 4.1).

Таблиця 4.1 – Передумови до здійснення чисельного інтегрування

$x_0 = a$	$x_1 = x_0 + h$	$x_2 = x_0 + 2 \cdot h$	...	$x_n \approx b$
$y_0 = y(x_0)$	-	-	...	$y(n) \approx y(b)$

Здійснення чисельного інтегрування полягає у поступовому заповненні вільних комірок для y -значень.

Крок інтегрування h визначається кількістю точок розбиття k :

$$h = \left\lfloor \frac{b - x_0}{k} \right\rfloor = \frac{x_n - x_0}{k}, \quad (4.3)$$

де квадратні знизу дужки означають округлення до меншого цілого.

Чим менше значення h , тим вищою є точність рішення ЗДР, і навпаки. Із зменшенням h зростає і обсяг обчислень, і навпаки.

Задача Коші формулюється наступним чином:

$$\begin{cases} y' = f(x, y), x \in [x_0, b], \\ y(x_0) = y_0, \end{cases} \quad (4.4)$$

де $f(x, y)$ – задана, в загальному випадку нелінійна, функція від двох змінних; $y(x_0) = y_0$ – задані початкові умови, необхідні для запуску обчислювального процесу.

При вирішенні задачі Коші оперують наступними припущеннями:

- рішення задачі існує і єдине $\forall x \in [x_0, b]$;
- шукане рішення характеризується необхідною гладкістю – для застосування обраного методу чисельного інтегрування.

4.1.2 Чисельне розв'язання ЗДР у середовищі SciLab

Для розв'язання ЗДР у середовищі *SciLab* призначена функція **ode** (*Ordinary Differential Equations*):

```
y = ode([type,] y0, t0, t, f);
```

де y – вектор-результат розв'язання ЗДР; *type* – опційний параметр, що дозволяє задати метод чисельного інтегрування; y_0 – вектор (матриця) початкових умов, необхідний для формулювання задачі Коші; t_0 – дійсне значення (скаляр) початкового часу – ліва межа інтервалу $[a, b]$, а саме $a = x_0 \in [a, b]$; t – вектор дійсних значень часу для яких знаходиться рішення ЗДР; f – функція $f(x, y)$.

Якщо у якості *type* вказати значення "rk", чисельне інтегрування здійснюватиметься методом Рунге-Кутти 4-го порядку.

За замовчанням робота функції **ode** ґрунтується на використанні методу Ейлера:

$$y_{i+1} = y_i + h \cdot f(x_i, y_i), \quad (4.5)$$

де $i = 0, 1, \dots, n \in N$ – порядковий номер точки у межах інтервалу $[a, b]$.

Перевагою використання методу Ейлера є простота його реалізації, а суттєвим недоліком – необхідність обчислення часткових похідних функції $f(x, y)$ на кожному кроці інтегрування.

Реалізація методу Ейлера згідно виразу (4.5) називається явною. У випадку неявного методу Ейлера маємо наступне:

$$y_{i+1} = y_i + h \cdot f(x_{i+1}, y_{i+1}). \quad (4.6)$$

Існує також уточнений метод Ейлера:

$$y_{i+1} = y_{i-1} + 2 \cdot h \cdot f(x_i, y_i). \quad (4.7)$$

Він характеризується вищою точністю, проте потребує додаткових початкових умов для запуску процесу обчислень.

Альтернативою до методу Ейлера є метод Рунге-Кутти 4-го порядку. Він характеризується більшою точністю, у порівнянні із методом Ейлера, проте потребує більших обчислень:

$$\left\{ \begin{array}{l} \xi_1^i = f(x_i, y_i), \\ \xi_2^i = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} \xi_1^i\right), \\ \xi_3^i = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} \xi_2^i\right), \\ \xi_4^i = f(x_i + h, y_i + h \xi_3^i), \\ \Delta y_i = \frac{h}{6} (\xi_1^i + 2\xi_2^i + 2\xi_3^i + \xi_4^i), \\ y_{i+1} = y_i + \Delta y_i. \end{array} \right. \quad (4.8)$$

Для кожної i -ї точки інтервалу $[a, b]$ обчислення згідно методу (4.8) здійснюється у 4 етапи:

1) Обчислюється поправка ξ_1^i , що відповідає явному методу Ейлера (4.5);

2) Обчислюється поправка ξ_2^i , що відповідає уточненому методу Ейлера (4.7);

3) Обчислюється поправка ξ_3^i , що відповідає уточненому методу Ейлера (4.7);

4) Обчислюється поправка ξ_4^i , що відповідає неявному методу Ейлера (4.6).

Розглянемо приклад розв'язання ЗДР 1-го порядку (відносно першої похідної) у середовищі *SciLab*. При цьому позначимо x через t . Тобто вісь Ox слід розглядати як представлення часу.

Нехай маємо функцію $f(t, y)$ наступного вигляду:

$$f(t, y) = y^2 - y \cdot \sin t + \cos t. \quad (4.9)$$

Для функції (4.9) ЗДР 1-го порядку матиме наступний вигляд:

$$\frac{\partial y}{\partial t} = y^2 - y \cdot \sin t + \cos t. \quad (4.10)$$

Для чисельного розв'язання диференційного рівняння (4.10) необхідно сформулювати задачу Коші:

$$\begin{cases} \frac{\partial y}{\partial t} = y^2 - y \cdot \sin t + \cos t, \\ y(t_0) = y_0. \end{cases} \quad (4.11)$$

Для розв'язання диференційного рівняння (4.10) з використанням засобів *SciLab* необхідно виконати наступні п'ять кроків:

– задати функцію $f(t, y)$;

- сформулювати задачу Коші – задати початкові умови – для запуску процесу чисельного розв’язання ЗДР;
- задати діапазон пошуку рішення, вказавши фіксоване значення кроку інтегрування. Адже для кожного елементу цього вектору значень часу необхідно знайти значення функції $f(t, y)$ – щоб стверджувати про розв’язання ЗДР;
- викликати функцію **ode** у середовищі *SciLab*;
- візуалізувати одержаний результат роботи функції **ode** – розв’язок ЗДР – шукані значення функції $f(t, y)$ у точках заданого інтервалу, розміщених із фіксованим кроком, використавши функцію **ode**.

Керуючись вищенаведеним алгоритмом, створимо відповідну *SciLab*-реалізацію (лістинг 4.1).

Лістинг 4.1 – Чисельне розв’язання рівняння (4.10)

```
function ydot=f(t, y)
    ydot=y^2-y*sin(t)+cos(t)
endfunction
y0=0;
t0=0;
t=0:0.1:%pi;
y = ode(y0,t0,t,f);
plot(t,y)
```

Вміст лістингу 4.1 слід вводити у командному вікні *SciLab* порядково. Рядок `t=0:0.1:%pi`; визначає інтервал пошуку рішення $[0, \pi]$, із кроком 0.1, де `%pi` – константне значення π . Отже, маємо 32 точки у межах інтервалу, в яких будемо шукати значення функції $f(t, y)$.

Результат розв’язання рівняння (4.10) подано на рис. 4.1.

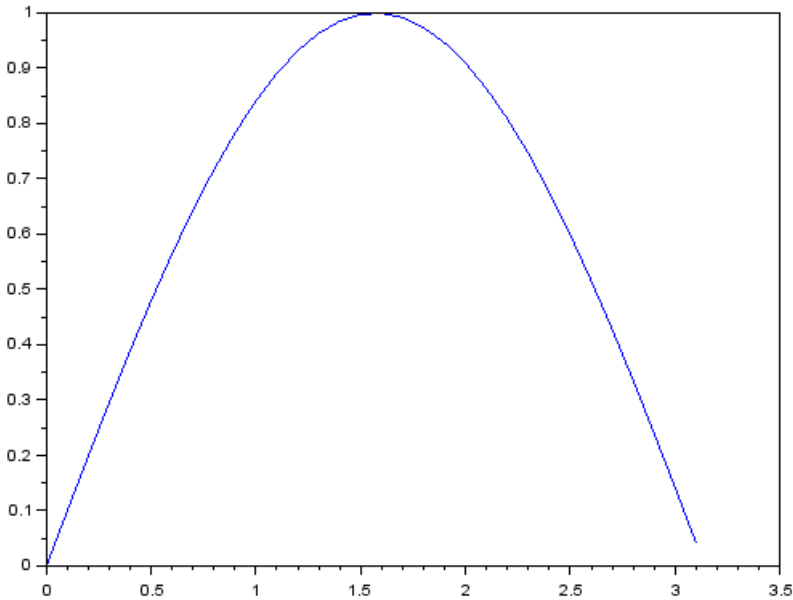


Рисунок 4.1 – Графічне представлення розв'язку рівняння (4.10)

4.2 Завдання до лабораторної роботи

4.2.1 Поекспериментувати із значенням кроку інтегрування, збільшивши його від 0.1 до 0.5. Переглянути та проаналізувати зміну графіку-розв'язку в залежності від цього. Як вплине таке збільшення на кількість точок у межах інтервалу – на кількість елементів вектору-розв'язку?

4.2.2 Провести чисельне інтегрування для розглянутого прикладу із використанням методу Рунге-Кутти 4-го порядку замість методу Ейлера, що використовується за замовчанням.

4.2.3 Керуючись вищенаведеним прикладом, розв'язати ЗДР 1-го порядку згідно варіанту (табл. 4.1). Використовувати метод Ейлера.

Зауваження: для обох варіантів інтервал пошуку рішення становить $[0,1]$, $y(0)=1$, $h=0.1$.

Таблиця 4.1 –Завдання до виконання

Варіант	$f(t, y)$	Точне рішення
1	$t^2 - 2 \cdot y$	$y(t) = \frac{3}{4} \cdot e^{-2t} + \frac{1}{2} \cdot t^2 - \frac{1}{2} \cdot t + \frac{1}{4}$
2	$2 \cdot t - 3 \cdot y$	$y(t) = -\frac{2}{9} + \frac{2}{3} \cdot t + \frac{11}{9} \cdot e^{-3t}$

4.2.4 Розв'язати ЗДР 1-го порядку згідно варіанту (табл. 4.1), використовуючи метод Рунге-Кутти 4-го порядку.

4.2.5 Побудувати графіки залежності абсолютної похибки чисельного інтегрування ($\Delta y = |y(t_i) - y_i|$) в залежності від розміру кроку інтегрування: $h = 0.1, 0.15, \dots, 0.25$. Графіки побудувати для обох методів. Проаналізувати одержані результати.

4.3 Зміст звіту

4.3.1 Титульний лист.

4.3.2 Мета роботи.

4.3.3 Графіки, одержані при виконанні завдання 4.2.5.

4.3.4 Відповіді на контрольні питання.

4.4 Контрольні питання

4.4.1 Диференційні рівняння. Визначення та призначення.

4.4.2 Сфери використання диференційних рівнянь.

4.4.3 Специфіка чисельних методів розв'язання ЗДР.

4.4.4 Формулювання задачі Коші.

4.4.5 Явний, неявний та уточнений методи Ейлера. Переваги та недоліки.

4.4.6 Метод Рунге-Кутти 4-го порядку. Характеристика. Переваги та недоліки.

4.4.7 Призначення функції ode середовища SciLab.

4.4.8 Етапи розв'язання ЗДР у середовищі SciLab.

5 РОЗРАХУНКОВО-ГРАФІЧНЕ ЗАВДАННЯ

Мета роботи – ознайомитися з алгоритмами генерації чисел Фібоначчі та закріпити набуті навички по оцінюванню асимптотичної складності алгоритмів.

5.1 Теоретичні відомості

Числа Фібоначчі представляються послідовністю наступного вигляду: 0, 1, 1, 2, 3, 5, 8, ... Узагальнююче аналітичне представлення таке: $F_n = F_{n-1} + F_{n-2}$, де $n \in N$ – порядковий номер числа Фібоначчі F_n , тобто значення n -го числа є сумою значень $n-1$ -го та $n-2$ -го чисел. У додатку А подано точні значення чисел Фібоначчі для $n < 90$. Числа Фібоначчі тісно пов'язані із поняттям золотого перетину, що поширене, зокрема, в архітектурі. Найпростішим підходом до одержання такого числа за заданою його позицією n у послідовності є рекурентна реалізація (лістинг 5.1).

Лістинг 5.1 – Рекурентний підхід до знаходження чисел Фібоначчі

```
#include <iostream>
using namespace std;
int getFibo(int n)          // рекурентна реалізація
{
    if(n<2) return n;
    else return getFibo(n-1)+getFibo(n-2);
}
int main()
{
    int n;
    cout<<"enter fibo position>>>"; cin>>n;
    cout<<"fibo number: "<<getFibo(n)<<endl;
    system("pause");
    return 0;
}
```

Функція `getFibo` може бути реалізована більш компактно – за рахунок використання тернарного оператору `?:` (лістинг 5.2).

Лістинг 5.2 – Альтернативна реалізація функції `getFibo`

```
int getFibo(int n)
{
return (n<2)?n:getFibo(n-1)+getFibo(n-2);
}
```

Аналізуючи рекурентний підхід до знаходження чисел Фібоначчі, можна дійти висновку щодо асимптотичної складності даної реалізації – $O(2^n)$. Вона є експоненційною, що незадовільно з позиції обсягу виконуваних обчислень для відносно великих позиційних номерів. Наприклад, для $n=40$ обчислення вже виконуватиметься кілька секунд.

Альтернативний підхід до знаходження чисел Фібоначчі – використання формули Біне (*Binet*):

$$F_n = \frac{\varphi^n - \left(1 - \frac{\sqrt{5}}{2}\right)^n}{\sqrt{5}}, \quad (5.1)$$

де $\varphi = \frac{1 + \sqrt{5}}{2} \approx 1,618$ – золотий перетин.

Із вигляду виразу (5.1) відповідну асимптотичну складність можна охарактеризувати оцінкою $O(\varphi^n)$.

Іноді для зручності використовують спрощену формулу:

$$F_n = \left\lfloor \frac{\varphi^n}{\sqrt{5}} + 0,5 \right\rfloor, \quad (5.2)$$

де замість обчислення виразу $\left(\frac{1 - \sqrt{5}}{2}\right)^n$ використовується

поправочне значення 0,5.

Іноді вираз (5.2) використовується і без поправки 0,5. Це пов'язано зі специфікою машинного округлення дійсних чисел, що

суперечить математичним правилам, тобто, наприклад, навіть значення 1,9 буде округлене до найменшого цілого – одиниці. Квадратні дужки в (5.2) означають округлення до меншого цілого.

Істотний недолік такого підходу – необхідність працювати з дійсними числами та потреба виконувати обчислення з високою точністю. Програмну реалізацію підходу на основі виразу (5.1) наведено в лістингу 5.3.

Лістинг 5.3 – Реалізація на основі формули Біне

```
long getFiboBinet(int n)
{
double squ = sqrt(5);
double phi = (1+squ)/2;
double no_phi = (1-squ)/2;
return (long) ((pow(phi,n)-pow(no_phi,n))/squ);
}
```

Для роботи функції лістингу 5.3 необхідно додатково підключити бібліотеку `<cmath>`.

Кожний із розглянутих підходів до генерації чисел Фібоначчі має як переваги, так і недоліки.

Рекурентний підхід є простим в реалізації, проте є неефективним при обчисленні F_n для $n > 40$.

Підхід на основі формули Біне дещо складніший у реалізації, проте характеризується ліпшою ефективністю у порівнянні із рекурентною реалізацією. Суттєвим недоліком такого підходу є залежність від точності обчислень, а саме – від обраного типу даних – `double` чи `float`. Тип `double` дозволяє одержувати вірні значення F_n для більших n .

Виконання роботи орієнтоване на апостеріорне оцінювання обчислювальної асимптотичної ефективності наведених вище алгоритмів генерації чисел Фібоначчі. Це означає, що вимірюваним показником виступатиме час, витрачений на роботу програмної реалізації алгоритму.

Для виконання РГЗ потрібно використовувати середовища *Dev-C++* та *TableCurve 2D*.

5.2 Завдання до виконання

5.2.1 Створити новий проект у середовищі *Dev-C++*. В ньому, керуючись завданням 3.2.3 лабораторної роботи 3, визначити дві функції – на основі лістингів 5.1 та 5.3. Одержати таблицю середніх арифметичних значень часових витрат для кожної із функцій для $n = 10, 20, \dots, 40$, де n – порядковий номер числа F_n , що подається у якості аргументу функції.

5.2.2 Побудувати графік апроксимуючої функції на основі одержаних результатів виконання завдання 5.2.1, керуючись рекомендаціями до завдання 3.2.6 лабораторної роботи 3. Для цього слід скористатися утилітою *TableCurve 2D*.

Приблизний вигляд одержуваного графіку подано на рис. 5.1. Вигляд графіку та відповідна підібрана апроксимуюча функція $g(n) = e^{a+b \cdot n}$ підтверджують експоненційну асимптотичну складність алгоритму на основі рекурентної реалізації (лістинг 5.1).

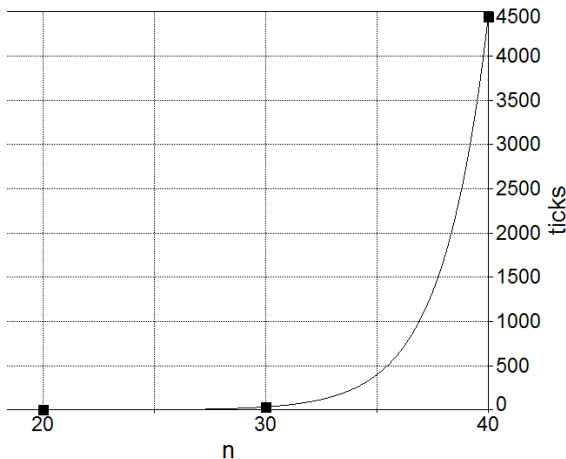


Рисунок 5.1 – Результат вирішення задачі апроксимації

5.2.3 Реалізувати програмно одержану у завданні 5.2.2 підібрану апроксимуючу функцію $g(n)$. Приклад реалізації функції $g(n) = e^{a+b \cdot n}$ для рис. 5.1, де $a = -10,842839$, $b = 0,48109095$, подано у лістингу 5.4.

Лістинг 5.4 – Програмна реалізація апроксимуючої функції

```
double iterEstim(int n)
{
    double a = -10.842839;
    double b = 0.48109095;
    return exp(a+b*(double)n);
}
```

5.2.4 Програмна реалізація апроксимуючої функції $g(n)$ дає змогу прогнозувати потенційні часові витрати на одержання чисел Фібоначчі згідно рекурентного підходу, а саме – вирішувати задачу екстраполяції. Необхідно вивести на консоль результати виклику функції `iterEstim` (у тіках та у секундах) для $n = 41, 42, \dots, 89$ (додаток А).

5.2.5 Обчислити та вивести на консоль значення абсолютної та відносної похибок апроксимації для $n = 0, 1, \dots, 40$:

$$\Delta_F = \text{abs}(\xi_n - g(n)), \quad (5.3)$$

$$\delta_F = \Delta_F / g(n), \quad (5.4)$$

де Δ_F та δ_F – абсолютна та відносна похибки відповідно;
 ξ_n – фактичне значення часових витрат на обчислення F_n .

5.2.6 На основі формули Біне (5.1), використовуючи лістинг 5.3, одержати числа Фібоначчі для $n = 0, 1, \dots, 99$ (додаток А), вивівши їх на консоль. Визначити, при якому граничному значенні n ще не починається переповнення розрядної сітки результату обчислень. Позначимо його як np .

5.2.7 Узяти реалізацію формули Біне лістингу 5.3 за еталонну. Позначимо її як $f_1(n)$. Обрати альтернативну реалізацію згідно варіанту – $f_2(n)$ (табл. 5.1). Розрахувати числа Фібоначчі кожною із реалізацій. Результат роботи програми має бути представлений у 4 стовпці (табл. 5.2).

Таблиця 5.1 – Завдання до виконання

Вар.	$f_2(n)$ до завдання 7 на основі виразу	
1	(5.2)	з урахуванням поправки 0,5;
2		без урахування поправки 0,5;
3	(5.1)	у якості типу даних використати float;
4	(5.2)	з урахуванням поправки 0,5;
		у якості типу даних використати float;
5		без урахування поправки 0,5;
		у якості типу даних використати float;
6		з урахуванням поправки 0,5;
7		без урахування поправки 0,5;
8	(5.1)	у якості типу даних використати float;
9	(5.2)	з урахуванням поправки 0,5;
		у якості типу даних використати float;
10		без урахування поправки 0,5;
		у якості типу даних використати float.

Таблиця 5.2 – Шаблон оформлення результату роботи програми згідно завдання 5.2.7.

n	$f_1(n)$	$f_2(n)$	$abs(f_1(n) - f_2(n))$
1	...		
...		...	
np			...

У табл. 5.2 $abs(f_1(n) - f_2(n))$ – абсолютна похибка обчислень.

Проаналізувати та обґрунтувати одержані результати.

5.3 Зміст звіту

5.3.1 Титульний лист.

5.3.2 Мета роботи.

5.3.3 Програмний код розроблених програм та результати їх роботи. Графік, побудований у середовищі *TableCurve 2D* при виконанні завдання 5.2.2, а також відповідну апроксимуючу функцію, із зазначенням коефіцієнтів.

5.3.4 Стислі висновки (2-3 речення) за результатами виконання кожного із завдань.

5.3.5 Відповіді на контрольні питання.

5.4 Контрольні питання

5.4.1 Охарактеризувати числа Фібоначчі.

5.4.2 Асимптотична складність, переваги та недоліки рекурентного підходу до генерації чисел Фібоначчі.

5.4.3 Асимптотична складність, переваги та недоліки використання формули Біне.

5.4.4 Залежність точності обчислень на основі формули Біне від обраного типу даних.

5.4.5 Обґрунтувати необхідність оцінювання потенційних часових витрат на розрахунок чисел Фібоначчі.

ЛИТЕРАТУРА

1. Алексеев Е. Р. Scilab: решение инженерных и математических задач / Е. Р. Алексеев, О. В. Чеснокова, Е. А. Рудченко. – М.: ALT Linux ; БИНОМ. Лаборатория знаний, 2008. – 269 с.
2. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн; пер. с англ., под ред. И.В. Красикова. – [3-е изд.]. – М.: Вильямс, 2013. – 1328 с.
3. Лафоре Р. Структуры данных и алгоритмы в Java / Р. Лафоре; пер. с англ. – [2-е изд.]. – СПб.: Питер, 2013. – 704 с.
4. Проскуряков И. В. Сборник задач по линейной алгебре: учеб. пособие / И. В. Проскуряков. – [13-е изд.]. – СПб.: Лань, 2010. – 480 с.
5. Горбаченко В. И. Вычислительная линейная алгебра с примерами на MATLAB: учеб. пособие / В. И. Горбаченко. – СПб.: БХВ-Петербург, 2011. – 320 с.
6. Вержбицкий В. М. Основы численных методов: учебник для вузов / В.М. Вержбицкий. – М.: Высш. шк., 2002. – 840 с.
7. Числа Фибоначчи [Электронный ресурс]. – Режим доступа: http://e-maxx.ru/algo/fibonacci_numbers. – Заголовок с экрана.

Додаток А

Числа Фібоначчі

Таблиця А.1 – Точні значення чисел Фібоначчі

<i>n</i>	Знач.	<i>n</i>	Знач.	<i>n</i>	Знач.
0	0	30	832040	60	1548008755920
1	1	31	1346269	61	2504730781961
2	1	32	2178309	62	4052739537881
3	2	33	3524578	63	6557470319842
4	3	34	5702887	64	10610209857723
5	5	35	9227465	65	17167680177565
6	8	36	14930352	66	27777890035288
7	13	37	24157817	67	44945570212853
8	21	38	39088169	68	72723460248141
9	34	39	63245986	69	117669030460994
10	55	40	102334155	70	190392490709135
11	89	41	165580141	71	308061521170129
12	144	42	267914296	72	498454011879264
13	233	43	433494437	73	806515533049393
14	377	44	701408733	74	1304969544928657
15	610	45	1134903170	75	2111485077978050
16	987	46	1836311903	76	3416454622906707
17	1597	47	2971215073	77	5527939700884757
18	2584	48	4807526976	78	8944394323791464
19	4181	49	7778742049	79	14472334024676221
20	6765	50	12586269025	80	23416728348467685
21	10946	51	20365011074	81	37889062373143906
22	17711	52	32951280099	82	61305790721611591
23	28657	53	53316291173	83	99194853094755497
24	46368	54	86267571272	84	160500643816367088
25	75025	55	139583862445	85	259695496911122585
26	121393	56	225851433717	86	420196140727489673
27	196418	57	365435296162	87	679891637638612258
28	317811	58	591286729879	88	1100087778366101931
29	514229	59	956722026041	89	1779979416004714189

Додаток Б

Програмні реалізації методів

Лістинг Б.1 – Функція одержання визначника матриці коефіцієнтів A

```
double determ(double** Arr, int size)
{
    int i,j;
    double det=0;
    double** matr;
    if(size==1)
        det=Arr[0][0];
    else if(size==2)
        det=Arr[0][0]*Arr[1][1]-Arr[0][1]*Arr[1][0];
    else
    {
        matr=new double*[size-1];
        for(i=0;i<size;++i)
        {
            for(j=0;j<size-1;++j)
            {
                if(j<i)
                    matr[j]=Arr[j];
                else
                    matr[j]=Arr[j+1];
            }
            det+=pow((double)-1,(i+j))*determ(matr,size-1)*
Arr[i][size-1];
        }
        delete[] matr;
    }
    return det;
}
```

Лістинг Б.2 – Функція визначення визначника допоміжних матриць коефіцієнтів $A_1..A_N$, в яких стовбець N замінюється вектором правих частин $\bar{b}$

```
double determl(double** Arr, double* Brr, int size, int n)
```

```

// до функції передається матриця A, вектор B, розмір матриці
A та номер стовбця, який буде змінений на вектор B
{double**Arr1;
Arr1=new double*[size];
for(int i=0;i<size;++i)
Arr1[i]=new double [size];
for(int i=0;i<size;++i)
for(int j=0;j<size;++j)
Arr1[i][j]=Arr[i][j];
for(int i=0;i<size;++i)
for(int j=n;j<n+1;++j)
Arr1[i][j]=Brr[i];
double D=determ(Arr1, size); // виклик функції (лістинг Б.1)
for(int i=0;i<size;++i)
delete[] Arr1[i];
delete[] Arr1;
return D;
}

```

Лістинг Б.3 – Метод Гауса

```

// матриця matrix складається з матриці  $A$  та вектору  $\bar{b}$ 
//прямий хід
double tmp, xx[m];
int k;
for (i=0; i<n; i++)
{tmp=matrix[i][i];
for (j=n;j>=i;j--)
matrix[i][j]/=tmp;
for (j=i+1;j<n;j++)
{
tmp=matrix[j][i];
for (k=n;k>=i;k--)
matrix[j][k]-=tmp*matrix[i][k];
}
}
// Обернений хід
xx[n-1] = matrix[n-1][n];
for (i=n-2; i>=0; i--)
{
xx[i] = matrix[i][n];
for (j=i+1;j<n;j++) xx[i]-=matrix[i][j]*xx[j];}

```

Лістинг Б.4 – Функція визначення оберненої матриці

```

void inversion(double **A, int N)
{double temp;
double **E = new double *[N];
for (int i = 0; i < N; i++)
E[i] = new double [N];
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++)
{
E[i][j] = 0.0;
if (i == j)
E[i][j] = 1.0;
}
for (int k = 0; k < N; k++)
{
temp = A[k][k];
for (int j = 0; j < N; j++)
{
A[k][j] /= temp;
E[k][j] /= temp;
}
for (int i = k + 1; i < N; i++)
{
temp = A[i][k];
for (int j = 0; j < N; j++)
{
A[i][j] -= A[k][j] * temp;
E[i][j] -= E[k][j] * temp;
}
}
}
for (int k = N - 1; k > 0; k--)
{
for (int i = k - 1; i >= 0; i--)
{
temp = A[i][k];
for (int j = 0; j < N; j++)
{
A[i][j] -= A[k][j] * temp;
E[i][j] -= E[k][j] * temp;
}
}
}
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++)
A[i][j] = E[i][j];

```

```

for (int i = 0; i < N; i++)
delete [] E[i];
delete [] E;
}

```

Лістинг Б.5 – Знаходження добутку матриці A на вектор B

```

for (int j = 0; j < N; j++)
{matrixX[j]=0;
    for (int i = 0; i < N; i++)
        matrixX[j]+=matrixA[j][i]*matrixB[i];
}

```

Лістинг Б.6 – Функція методу простої ітерації

```

#define eps 0.0000000001 // задана точність обчислень
#define N 3
void iteration(double a[N][N],double b[N],double x[N])
{int i,j, k=0; // k - кількість ітерацій
double norma; // норма вектора
double xn[N]={0}; // вектор для поточної ітерації
do
{
    k++;
    norma=0.0;
    for(i=0;i < N;i++)
    {
        xn[i]=-b[i];
        for(j=0;j < N;j++)
        {
            if(i!=j)
                xn[i]+=a[i][j]*x[j];
        }
        xn[i]/=-a[i][i];
    }
    for(i=0;i < N;i++)
    {
        if(fabs(x[i]-xn[i]) > norma)
            norma=fabs(x[i]-xn[i]); //визначення норми вектора
        x[i]=xn[i];
    }
}
while(norma>eps); //перевірка на задану точність обчислень
printf("Number of iterations - %d\n",k);
return;
}

```