ФАКУЛЬТЕТ АВТОМАТИКИ, ТЕЛЕМЕХАНІКИ ТА ЗВ'ЯЗКУ

Кафедра „Обчислювальна техніка та системи управління"

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

«*МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ НА ЕОМ*»

для студентів будівельного факультету

Харків 2010

Методичні вказівки розглянуто та рекомендовано до друку на засіданні кафедри «Обчислювальна техніка та системи управління» 31 березня 2008 р., протокол № 9.

Призначені для студентів будівельного факультету академії усіх форм навчання.

Укладачі:

доценти В.Г. Пчолін,
В.С. Меркулов,
асист. О.В. Казанко

Рецензент

проф. Г.І. Загарій

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з дисципліни
«*МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ НА ЕОМ*»
для студентів будівельного факультету

Відповідальний за випуск Казанко О.В.

Редактор Решетилова В.В.

Підписано до друку 03.06.08 р.
Формат паперу 60х84 1/16 . Папір писальний.
Умовн.-друк.арк. 5,25. Обл.-вид.арк. 5,5.
Замовлення № Тираж 150. Ціна

Видавництво УкрДАЗТу, свідоцтво ДК № 2874 від. 12.06.2007 р.
Друкарня УкрДАЗТу,
61050, Харків - 50, майд. Фейєрбаха, 7

УКРАЇНСЬКА ДЕРЖАВНА АКАДЕМІЯ
ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

Кафедра «Обчислювальна техніка та системи управління»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни
«Математичне моделювання на ЕОМ»
для студентів будівельного факультету

Харків – 2010

Методичні вказівки розглянуто та рекомендовано до друку на засіданні кафедри «Обчислювальна техніка та системи управління» 31 березня 2008 р., протокол № 9.

Призначені для студентів будівельного факультету академії усіх форм навчання.

Укладачі:

доценти В.Г. Пчолін,
В.С. Меркулов,
асист. О.В. Казанко

Рецензент

проф. Г.І. Загарій

Зміст

Вступ

Комплекс лабораторних робіт призначений для поглибленого вивчення властивостей ряду числових методів, що особливо часто використовуються у дослідженні математичних моделей різноманітних технічних систем.

Математичне моделювання у теперішній час є невід'ємною частиною сучасного проектування нових машин, механізмів, будівельних конструкцій. Розроблення будь-якого нового технічного об'єкта потребує попереднього ретельного дослідження й аналізу можливих варіантів реалізації. Проектування машин, споруд та обладнання для залізниці не є винятком. Тому метою даних методичних вказівок є формування навичок оволодіння набором методів і прийомів, що сприяють ефективному вирішенню найважливіших задач, пов'язаних з технічним удосконаленням засобів та організацією перевізного процесу на залізниці. Це відповідає загальній меті підготовки фахівців, що володіють апаратом математичного моделювання складних систем та здатні комплексно використовувати в інженерній практиці математичні методи побудови й дослідження моделей за допомогою засобів обчислювальної техніки.

Пропоноване видання є основним посібником для вивчення мінімуму теоретичних знань, яким необхідно володіти при виконанні лабораторних робіт, що проводяться за дисциплінами "Математичне моделювання БКВРМ" та "Математичне моделювання" на будівельному факультеті академії для студентів денної за заочної форм навчання.

"Математичне моделювання БКВРМ" та "Математичне моделювання" є базовими курсами для подальшого вивчення інших спеціальних дисциплін. Кількість і послідовність виконання лабораторних робіт визначаються робочою програмою з дисципліни з урахуванням спеціалізації студентів.

Лабораторна робота № 1

РОЗРОБЛЕННЯ МАТРИЧНИХ МОДЕЛЕЙ

Результатом дослідження практично в будь-якій предметній галузі (виключення – деякі гуманітарні науки: історія, філософія та ін.) є таблиця експериментального матеріалу. Для того щоб дослідження одержало статус наукового і вирішувало поставлені перед інженером завдання, необхідно:

- побудувати на підставі цієї таблиці математичну модель, що дозволяє встановити істотні закономірності досліджуваного об'єкта, і щоб висновки, отримані в результаті аналізу цієї моделі, підтверджувалися при наступній експериментальній перевірці;
- ураховувати, що досить повне розуміння закономірностей досліджуваного об'єкта може бути досягнуто, якщо він описаний не одним, а декількома різними підходами або взаємодоповнюючими моделями.

На початковому етапі побудови моделі інженер:

- визначає перелік вхідних і вихідних параметрів;
- окреслює значення вихідних параметрів (кількісні або якісні), які треба поліпшити;
- здійснює вибір постановки задачі з набору запропонованих варіантів;
- здійснює вибір звітних матеріалів, що його цікавлять, з набору запропонованих варіантів.

Набір формалізованих алгоритмів, за допомогою яких здійснюється побудова математичних моделей об’єкта, в тому числі і матричних, як для "активного", так і для "пасивного" експерименту, отримав назву — інтелектуальна технологія.

Розглянемо ***приклад побудови матричної моделі***.

Задано двовимірний масив числових значень (квадратна матриця, що містить k рядків і k стовпців) - $\{ x_{ij} \}$, i=1,2,..., k; j=1,2,..., k.

Необхідно:

- спроектувати алгоритм обчислювального процесу і зобразити його схему (рисунки 1.1 – 1.4);
- розробити та відлагодити багатомодульну програму.

1 У головній програмі (див. текст Basic-програми) здійснити уведення елементів $\{ x_{ij} \}$, звернення до процедур і виведення результатів.

2 У процедурі-підпрограмі сформувати масив $\{ z_t \}$ з елементів $2< x_{ij} <15$, розташованих у парних рядках (i=2, 4, …, k) і підрахувати кількість елементів у $\{ z_t \}$.

3 У процедурі-підпрограмі сформувати одновимірний масив $\{y_i\}$, i=1,2,..., k, кожний елемент якого є середнє арифметичне значення елементів відповідного рядка матриці $\{ x_{ij} \}$.

4 У процедурі-функції визначити різницю між значеннями максимальних елементів головної і побічної діагоналей масиву $\{x_{ij} \}$.

Звіт з роботи повинен містити:

- схему алгоритму обчислювального процесу;
- програму алгоритмічною мовою;
- вихідні дані;
- отримані результати.

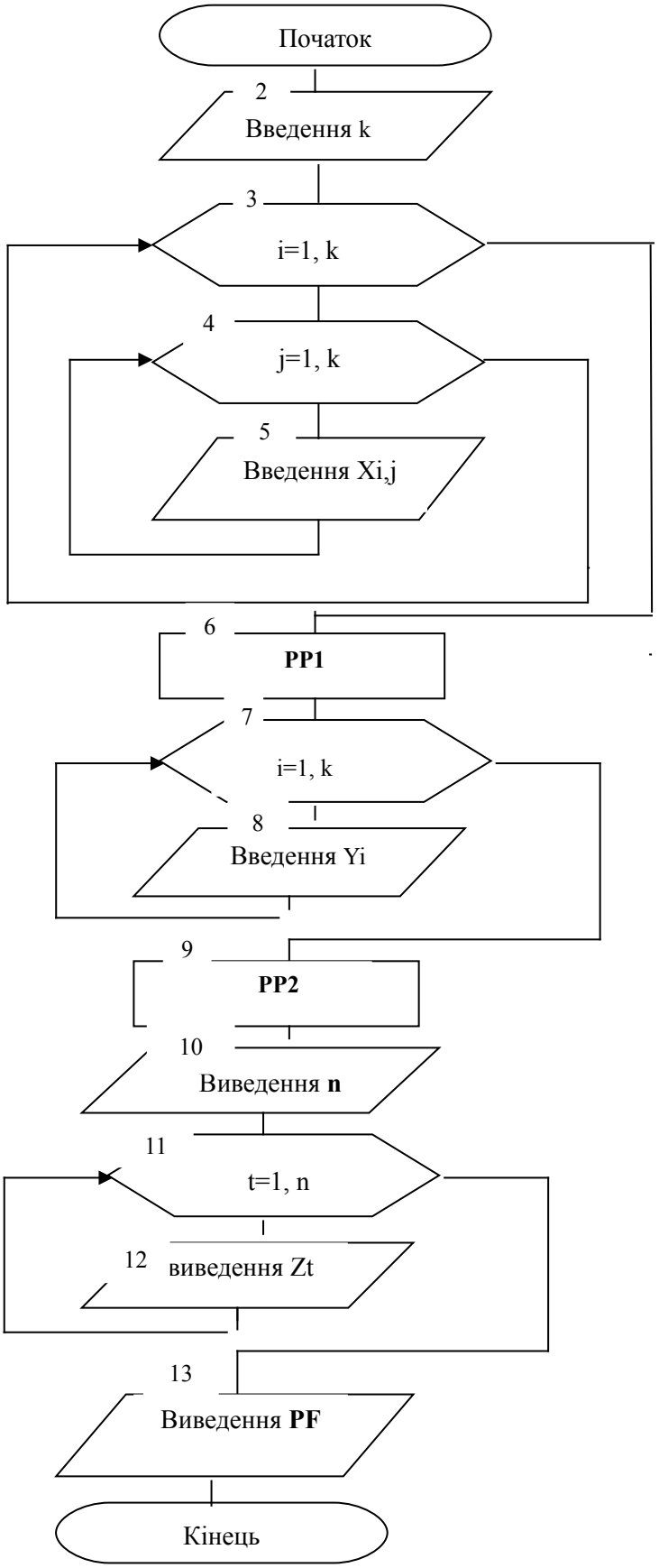


Рисунок 1.1 - Головний модуль

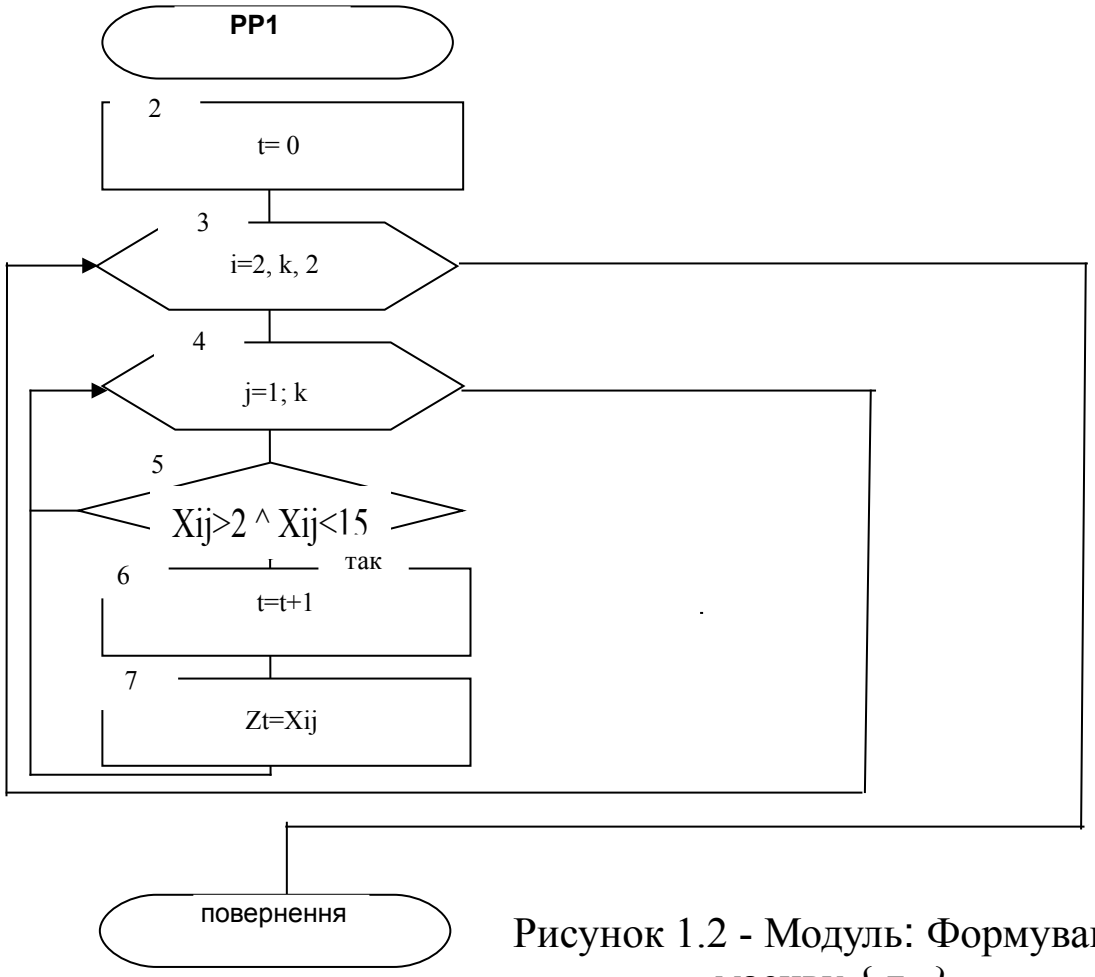


Рисунок 1.2 - Модуль: Формування масиву $\{ z_t \}$

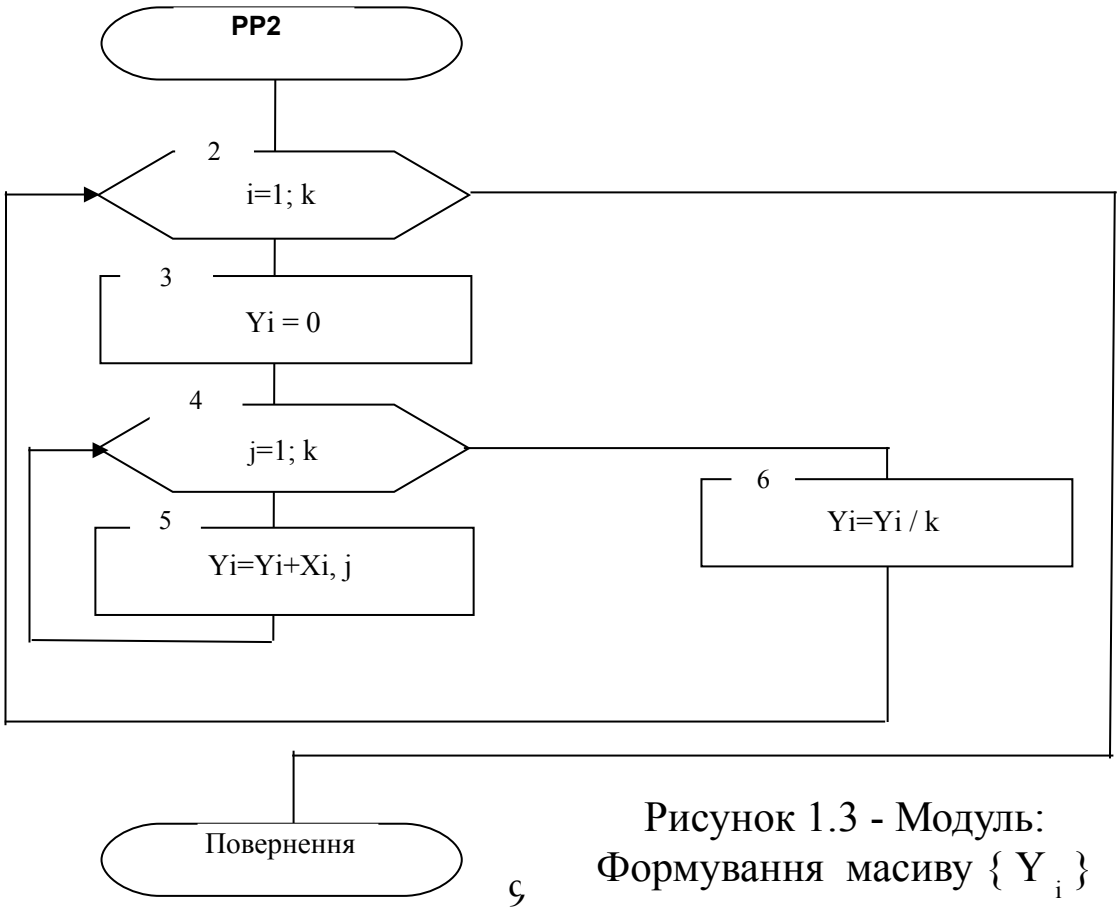


Рисунок 1.3 - Модуль: Формування масиву $\{ Y_i \}$

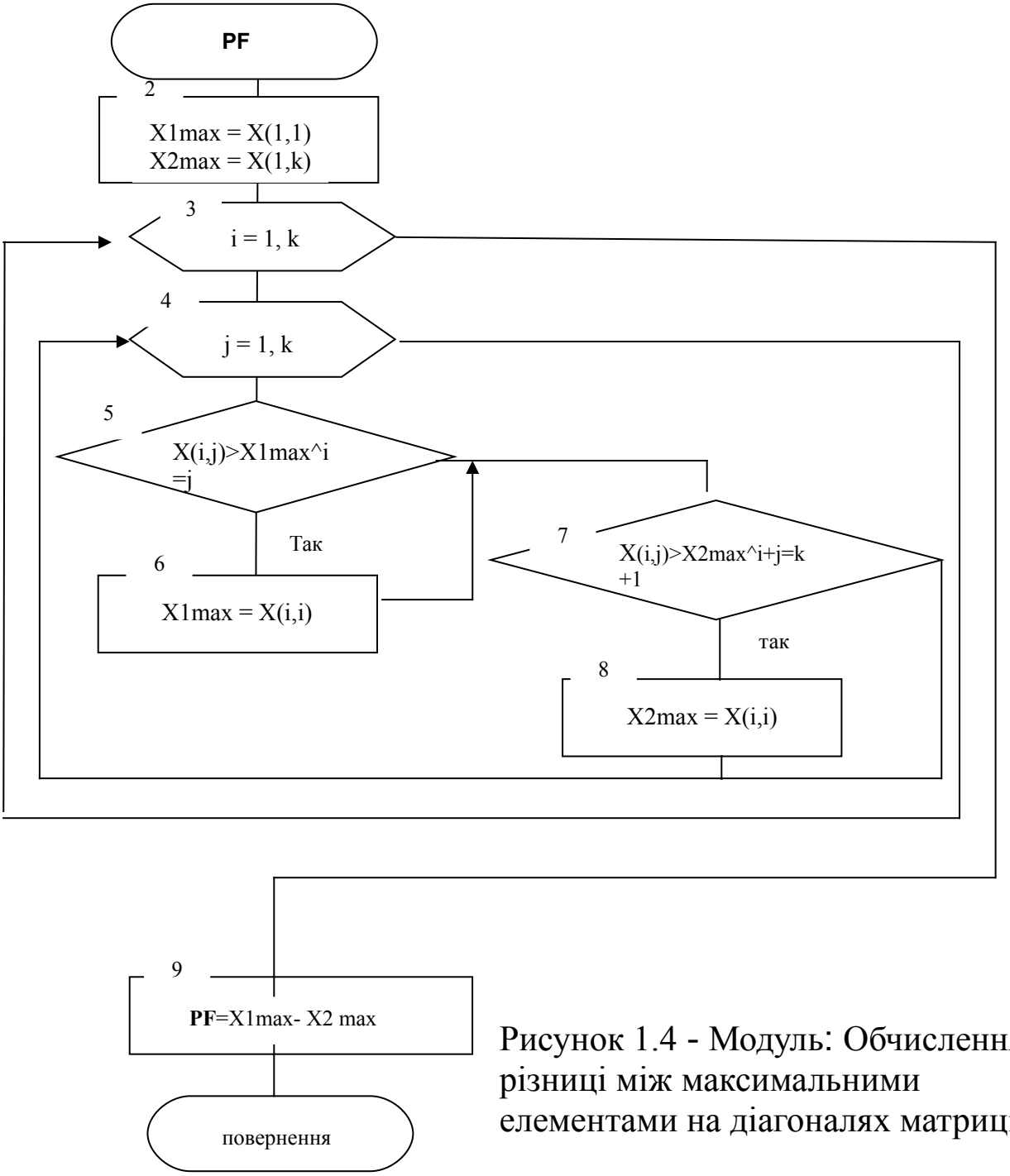


Рисунок 1.4 - Модуль: Обчислення різниці між максимальними елементами на діагоналях матриці

Basic-програма

(алгоритм на рисунках 1.1-1.4)

```
Головна програма
CLS: INPUT "ВВЕДИТе ЧИСЛО СТРОК (СТОЛБЦОВ) ";K
DIM X(K,K),Y(K),Z(K*K)
FOR i= 1 TO K
FOR j = 1 TO K
PRINT " ВВЕДИТе X(";i; j; ")=";
INPUT X(i,j)
NEXT j,i
CALL PP1(X(),K, Z(), N) ' вызов процедуры-
подпрограммы PP1
PRINT "У массиве Z елементов =";N
PRINT " массив Z "
FOR t = 1 TO N
    PRINT "Z(";t; ")=";Z(t)
NEXT t
PRINT
CALL PP2(X(),K, Y())' вызов процедуры-подпрограммы
PP2
PRINT " массив Y "
FOR i = 1 TO K
    PRINT "Y(";i ")=";Y(i)
NEXT i
PRINT: PRINT " Разность  ";PF(X(),K) ' вызов
процедуры-функции PF
END

'формування масиву  z(t)  з елементів 2< x(i,j) <15,
розташованих
'в парних рядках і підрахунок кількості елементів в
Z
SUB PP1(X(),K,Z(),N)
N=0
FOR i = 2 TO K STEP 2
FOR j = 1 TO K
IF X(i,j)>2 AND X(i,j)<15 THEN
    N=N+1:Z(N)=X(i,j)
END IF
NEXT J,I
END SUB

'формування масиву  Y  із середніх арифметичних
значень
```

```
'елементів кожного рядка окремо  масиву X
SUB PP2((X(),K, Y())
FOR I = 1 TO K
Y(I)=0
FOR J = 1 TO K
Y(I)=Y(I)+X(I,J)
NEXT J
Y(I)=Y(I)/K
PRINT "Y(";I;")=";Y(I)
NEXT I
END SUB

'визначення різниці між значеннями максимальних
елементів  головної та
'побічної діагоналей масиву X
FUNCTION PF((X(),K)
X1MAX=X(1,1): X2MAX=X(1,k)
FOR i = 1 TO K
FOR j = 1 TO K
IF X(i,j)>X1MAX   AND i=j   THEN  X1MAX=X(i,j)
IF X(i,j)>X2MAX AND i+j=k+1  THEN  X2MAX=X(i,j)
NEXT j,I
PF= X1MAX-X2MAX
END FUNCTION
```

Лабораторна робота № 2

ОБРОБКА РЕЗУЛЬТАТІВ ЕКСПЕРИМЕНТІВ

При дослідженні та моделюванні різних фізичних, технологічних, економічних та інших процесів часто виникає необхідність визначити:

- характер зміни якої-небудь змінної величини залежно від зміни іншої величини;
- аналітичний вираз цієї залежності;
- значення, які може приймати ця величина при різних умовах функціонування системи.

Ці дані дозволяють прогнозувати поведінку системи, приймати обґрунтовані рішення в питаннях проектування систем і т. д. Для одержання таких даних проводяться експерименти з моделлю системи або процесу.

Результати експериментів являють собою сукупності значень досліджуваних величин, які, як правило, задані таблично у вигляді координат точок.

В цій роботі розглядаються методи інтерполяції та апроксимації.

Задача проведення кривої *f(x)* через ці точки, що рівносильно побудові полінома (*n*-1)-го степеня, значення якого в точках $\{x_i\}$ збігаються зі значеннями $\{y_i\}$, і подальше визначення *f(x*)*, де ***x**** - проміжна точка інтервалу заданих координат, називається *задачею інтерполяції*.

2.1 Інтерполяція поліномом

Нехай задані значення координат точок $\{x_i\}$, $\{y_i\}$, $i = 1, 2, \ldots, N$ (отримані в результаті експерименту).

Багаточлен $p(x) = a_n x^n + \ldots + a_0$ називають *інтерполяційним багаточленом* (або *інтерполяційним поліномом*), якщо $p(x_i) = y_i$, $i = 1, 2, \ldots, n$. Точки $\{x_i\}$ – називають *вузлами інтерполяції* (або *вузловими точками*) (рисунок 2.1).

Для розв'язання такого типу задач застосовують метод Лагранжа, розрахункова формула якого має такий вигляд (інтерполяційний поліном у формі Лагранжа[1]):

$$f(x) = \frac{(x - x_2)(x - x_3)....(x - x_n)}{(x_1 - x_2)(x_1 - x_3)...(x_1 - x_n)} y_1 +$$

$$+ \frac{(x - x_1)(x - x_3)....(x - x_n)}{(x_2 - x_1)(x_2 - x_3)...(x_2 - x_n)} y_2 + ... + \frac{(x - x_2)(x - x_3)....(x - x_{n-1})}{(x_n - x_1)(x_n - x_2)...(x_n - x_{n-1})} y_n$$

$$f(x) = \sum_{i=1}^{n} y_i \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \quad . \qquad (2.1)$$

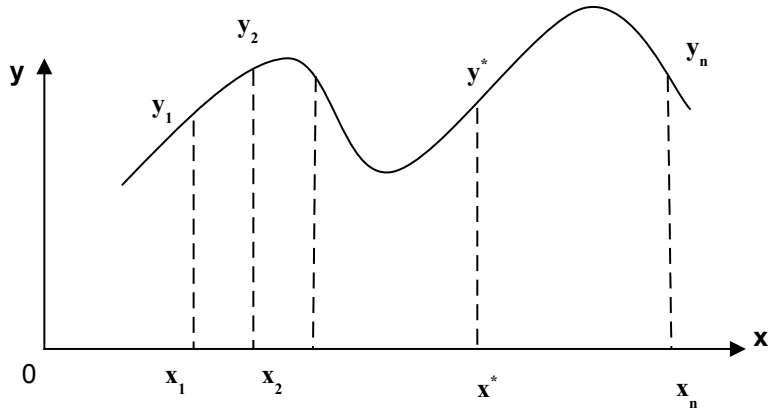


Рисунок 2.1 – Інтерполянт у формі Лагранжа

Алгоритм методу Лагранжа показано на рисунку 2.2.

Нижченаведена підпрограма-функція реалізує формулу (2.1).

Вхідні параметри:

t – аргумент інтерполяційного полінома,

***x*(), *y*()** – значення координат експериментальних точок,

n – кількість вузлових точок.

Результат роботи: **L** - значення інтерполяційного полінома в точці ***t.***

[1] Поряд з формою Лагранжа для побудови інтерполяційного багаточлена використовують також форму Ньютона, форму Чебишева та ін.

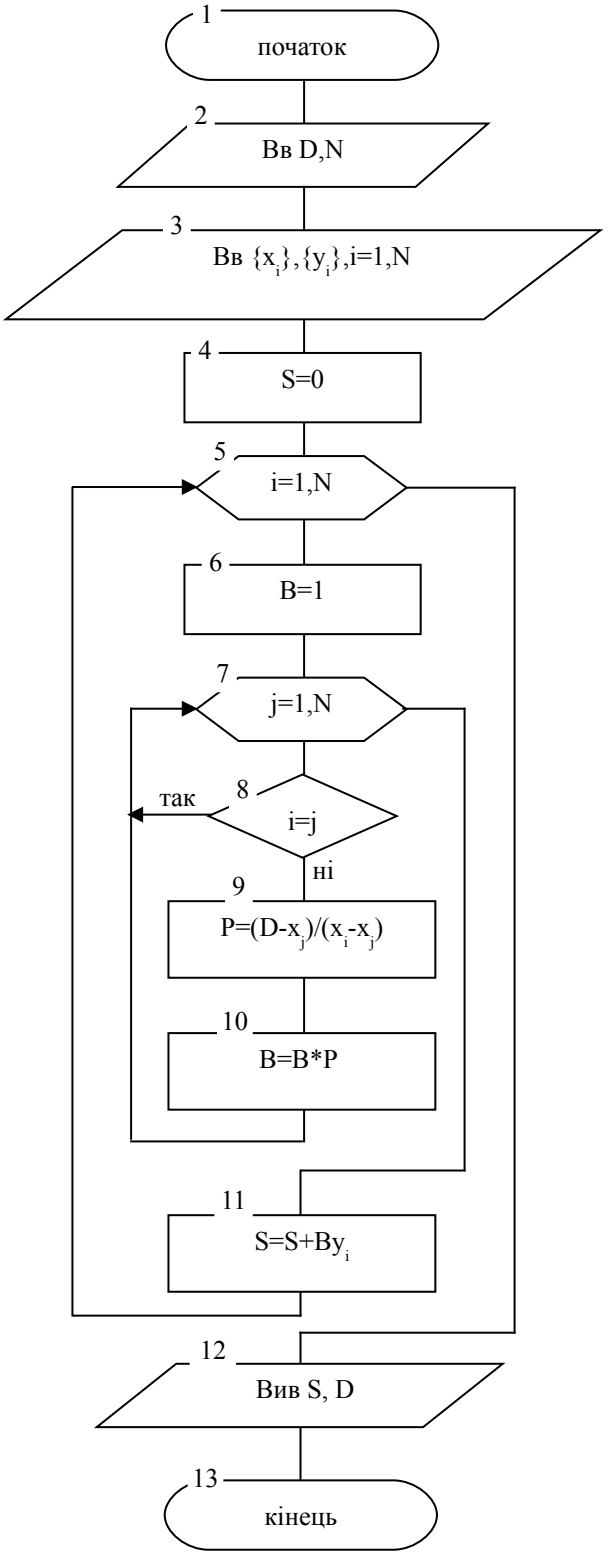


Рисунок 2.2 – Схема алгоритму методу Лагранжа

Підпрограма-функція, що реалізує метод Лагранжа

```
Function L (t, x(), y(), n)
     s=0
     FOR i = 1 TO n
      p=1
      FOR j=1 TO N
        IF j<>i THEN
          p=p*( t-x(j) )/( x(i)-x(j) )
        END IF
      NEXT j:
      s=s+y(i)*p
     NEXT i
     L=s
END Function
```

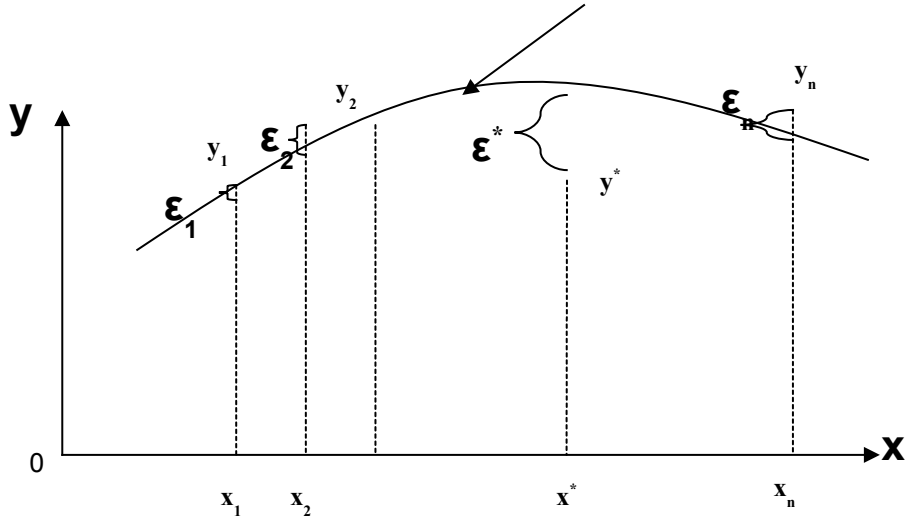
2.2 Апроксимація

Нехай задані значення координат точок $\{x_i\}$, $\{y_i\}$, $i = 1, 2, \ldots, N$ (отримані в результаті експерименту).

Регресійний аналіз дає можливість побудувати математичну модель залежності, що найкраще відповідає експериментальним даним. При цьому крива апроксимуючої функції проходить не через точки, отримані в результаті експерименту, а поблизу них, даючи усереднену характеристику процесу. Така процедура називається *згладжуванням* експериментальної залежності.

Метод найменших квадратів є одним з найбільш розповсюджених методів апроксимації, що використовуються на практиці. Сутність цього методу полягає у визначенні такого полінома $p(x) = q_0 x^n + \ldots + q_n$ ($q_0 \neq 0$), для якого значення функціонала $J(p) = \sum_{i=1}^{N} (p(x_i) - y_i)^2$ є мінімальним (тобто найменшою є сума квадратів відхилень значень полінома від експериментальних значень). За аналогією з інтерполяцією, функцію (не обов'язково поліном), що дає мінімум функціонала J, називають *апроксимантом* (рисунок 2.3).

апроксимуюча крива **f(x)**



2 2

Рисунок 2.3 - Апроксимант методу найменших квадратів

Нехай P є квадратний тричлен, тобто $p(x)=q_0x^2+q_1x+q_2$ ($q_0\neq 0$). Відповідно постановка задачі набуває такого змісту: необхідно визначити квадратний тричлен P, який дає мінімум функціонала $J(P)$. Оскільки коефіцієнти q_0, q_1, q_2 однозначно визначають квадратний тричлен P, то задача еквівалентна задачі про визначення екстремуму функції багатьох змінних. В нашому випадку $J(P)=J(q_0, q_1, q_2)\to \min$. Отримана задача у свою чергу зводиться до задачі розв'язання системи лінійних алгебраїчних рівнянь (СЛАР) третього порядку відносно невідомих коефіцієнтів q_0, q_1, q_2

$$\begin{cases} q_0\sum\limits_{i=1}^{N}x_i^4+q_1\sum\limits_{i=1}^{N}x_i^3+q_2\sum\limits_{i=1}^{N}x_i^2=\sum\limits_{i=1}^{N}y_ix_i^2, \\ q_0\sum\limits_{i=1}^{N}x_i^3+q_1\sum\limits_{i=1}^{N}x_i^2+q_2\sum\limits_{i=1}^{N}x_i=\sum\limits_{i=1}^{N}y_ix_i, \\ q_0\sum\limits_{i=1}^{N}x_i^2+q_1\sum\limits_{i=1}^{N}x_i+q_2\sum\limits_{i=1}^{N}1=\sum\limits_{i=1}^{N}y_i. \end{cases} \qquad (2.2)$$

Зазначимо, що $\sum\limits_{i=1}^{N}1=$N.

Розв'язавши цю СЛАР, знайдемо q_0, q_1, q_2, тобто визначимо квадратний тричлен $p(x)=q_0x^2+q_1x+q_2$, який мінімізує функціонал $J(P)$. Алгоритм методу найменших квадратів наведений на рисунку 2.4.

Для наведеної вище СЛАР використовуються такі позначення

$$\begin{vmatrix} SX4 & SX3 & SX2 \\ SX3 & SX2 & SX \\ SX2 & SX & N \end{vmatrix} \quad \text{еквівалентно} \quad \begin{vmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{vmatrix}$$

$$\begin{vmatrix} SX2Y \\ SXY \\ SY \end{vmatrix} \quad \text{еквівалентно} \quad \begin{vmatrix} d_0 \\ d_1 \\ d_2 \end{vmatrix}$$

Процедура-функція **Approccimation** реалізує метод найменших квадратів.

Вхідні параметри **Approccimation**: ***x*(), *y*()** – значення координат експериментальних, точок, ***n*** – кількість вузлових точок.

Результат роботи: Результат роботи:– значення коефіцієнтів ***a, в, c,*** що визначають апроксимант (квадратичну параболу) та відповідають вектору ***q***().

Ця процедура у свою чергу звертається до процедури-функції **`GaussMethod()`**[2] для визначення рішення СЛАР третього порядку:

Вхідні параметри **`GaussMethod`**: ***c***() – матриця коефіцієнтів та ***d***() – вектор-стовпчик вільних членів СЛАР, які обчислені в **Approccimation**.

Результат роботи: q_0, q_1, q_2 – розв'язання СЛАР.

Підпрограма-функція **Approccimation**, що реалізує метод найменших квадратів

```
FUNCTION Approccimation (x(), y(), q(), n)
 DIM c(3, 3), d(3)
 sx4 = 0: sx3 = 0: sx2 = 0: sx = 0
```

[2] **`GaussMethod()`** – підпрограма-функція, реалізує алгоритм Гауса. Автори цих методичних вказівок не наводять відповідний код програми; вважається, що цей код був розглянутий окремо і на момент виконання роботи є в наявності.

```
 FOR i = 1 TO n
  sx4 = sx4 + x(i) ^ 4
  sx3 = sx3 + x(i) ^ 3
  sx2 = sx2 + x(i) ^ 2
  sx = sx + x(i)
 NEXT i

  sx2y = 0:   sxy = 0: sy = 0
 FOR i = 1 TO n
  sx2y = sx2y + x(i) ^ 2 * y(i)
  sxy = sxy + x(i) * y(i)
  sy = sy + y(i)
 NEXT i
c(0, 0)=sx4: c(0, 1)=sx3: c(0, 2)=sx2: d(0)=sx2y
c(1, 0)=sx3: c(1, 1)=sx2: c(1, 2)= x:  d(1)= xy
c(2, 0)=sx2: c(2, 1)=sx:  c(2, 2)=n:   d(2)=sy
  P = GaussMethod(c(), d(), 3, q())
 PRINT "коефіцієнти апроксиманту", q(0), q(1), q(2)
END FUNCTION
```

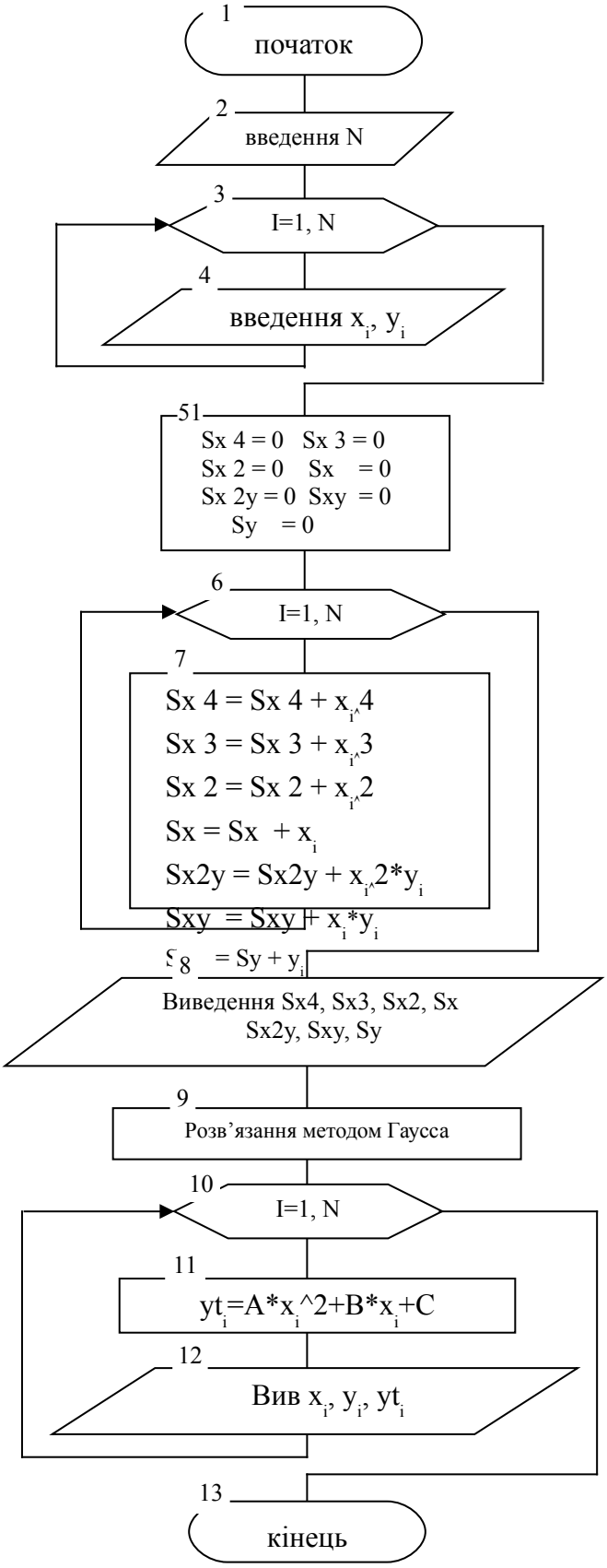


Рисунок 2.4 – Схема алгоритму методу найменших квадратів

Лабораторна робота № 3

МЕТОДИ ЧИСЕЛЬНОГО РОЗВ'ЯЗАННЯ НЕЛІНІЙНИХ РІВНЯНЬ

При дослідженні детермінованих моделей у ряді випадків виникає необхідність розв'язувати рівняння виду

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

де $f(x)$ – неперервна функція. Такі рівняння з однією змінною розділяють на дві групи: *алгебраїчні* та *трансцендентні.*

Визначення 3.1. Функцію називають *раціональною* або *алгебраїчною*, якщо для кожного раціонального значення аргументу $x = \frac{m}{n}$ (m – ціле, n – натуральне) значення функції $f(x)$ є раціональним числом. У противному випадку функцію $f(x)$ називають *ірраціональною* або *трансцендентною*, тобто, якщо знайдеться принаймні одне раціональне значення аргументу $x = \frac{m}{n}$, при якому функція $f(x)$ буде мати ірраціональне значення.

Нагадаємо, що число r є *раціональним* (точніше *раціональним дробом*), якщо це число можна зобразити у вигляді $\frac{m}{n}$, де m – ціле, n – натуральне.

Приклад 3.1.

$f(x) = x^2$ – алгебраїчна функція бо, якщо $x = \frac{m}{n}$, то $f(x) = \left(\frac{m}{n}\right)^2 = \frac{m^2}{n^2} = \frac{p}{q}$.

Приклад 3.2.

$f(x) = e^x$, $f(x) = \sin x$, $f(x) = \cos x$ – трансцендентні функції.

Визначення 3.2. Число x^* називають *коренем рівняння* (3.1), якщо $f(x^*) \equiv 0$.

Прийнята також термінологія: число x^* називають *нулем функції* f, або *розв'язком рівняння* (3.1).

Приклад 3.3.

Для рівняння $2x - 1 = 0$ корінь знаходиться таким чином: $2x = 1$, $x^* = \frac{1}{2}$

Методи розв'язання рівнянь поділяються на дві групи:

1 Аналітичні методи (прямі, точні). За їх допомогою корінь може бути виражений термінами функцій цілком визначеного класу (зазвичай таким класом є клас елементарних функцій). Наприклад, корені рівняння $\cos x - \frac{1}{2} = 0$ можна зобразити у такій аналітичній формі: $x = \pm\arccos\frac{1}{2} + 2\pi n$, $n = 0, \pm 1, \ldots$

2 Чисельні (наближені) методи, що базуються на методах чисельного аналізу. Наприклад, раціональне рівняння $x^2 - 2 = 0$ має нераціональні корені $x^* = \pm\sqrt{2}$.

У математичному моделюванні, як правило, використовуються саме такі методи групи 2, бо більшість нелінійних рівнянь не має раціональних коренів.

Розв'язати таке рівняння значить:

- установити, чи має воно корені;
- визначити їх кількість;
- знайти значення коренів із заданою точністю.

Виходячи з того, що функція може мати декілька коренів, процедура їх пошуку, як правило, складається з двох етапів:

- відділення коренів;
- уточнення коренів (метод половинного ділення, метод простих ітерацій) тощо.

3.1 Відділення коренів

Етап відділення коренів використовується для пошуку інтервалу існування кореня.

Сутність його полягає в наступному. Область допустимих значень (ОДЗ) розбивають на сегменти так, що на кожному з отриманих підмножин, функція буде мати один і тільки один корінь (рисунок 3.1). Потім до кожного з отриманих сегментів

можна застосувати методи уточнення коренів (метод половинного ділення, метод простих ітерацій тощо). Залежно від умови конкретної задачі інколи вдається зрозуміти, як саме слід розділити ОДЗ, щоб задовольнити умови методів уточнення.

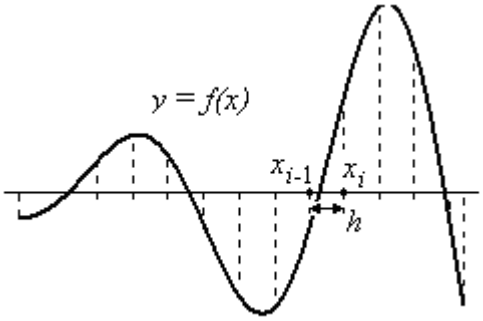


Рисунок 3.1 – Розбиття ОДЗ на сегменти

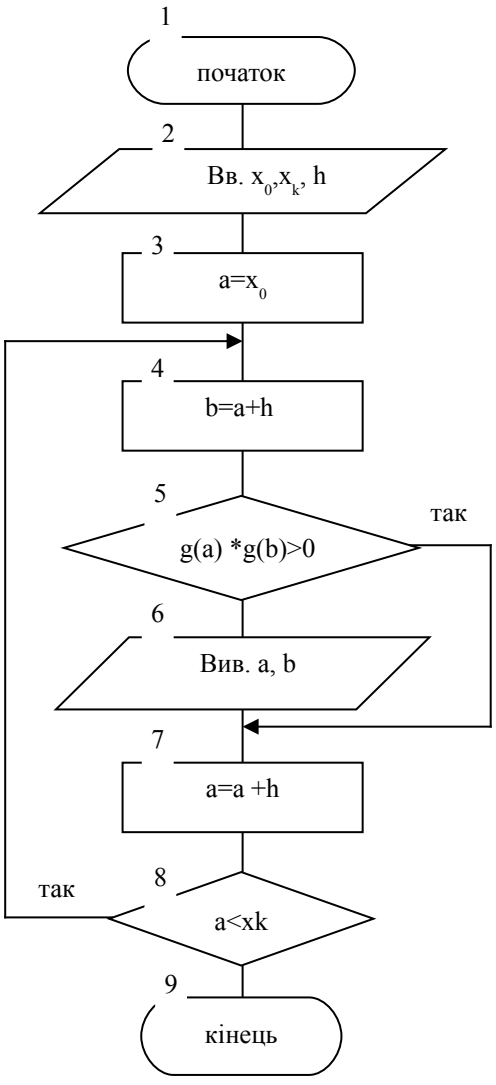


Рисунок 3.2 – Алгоритм відділення коренів

Обираємо рівномірно розподілені точки на ОДЗ: $x_0 < x_1 < ... < x_n$ $(x_i - x_{i-1} = h = const)$. Різницю $x_i - x_{i-1} = h$ будемо називати *кроком*. Далі обчислюються значення функції f у цих точках x_i. Врешті, здійснюється прямий перебір отриманих значень $f(x_i)$, причому, якщо при переході від точки x_{i-1} до точки x_i функція f змінює знак, то припускається, що на сегменті $[x_{i-1}, x_i]$ буде існувати один і тільки один корінь. При досить малому *кроці* h таке припущення буде правомірним. Слід також зазначити, що існують функції, які мають велику кількість корнів в ОДЗ, що розглядається. У цьому випадку вибір відповідного *кроку* h буде складним. Наприклад, функція $f(x) = \sin 100x$ на проміжку $[0, 2\pi]$ буде мати двісті корнів. Алгоритм відділення коренів наведений на рисунку 3.2.

В даному алгоритмі x_0 та x_n - це кінці ОДЗ даної функції, h – довжина сегмента, на якому очікується корінь (крок). В результаті виконання алгоритму на ЕОМ будуть отримані сегменти, на кожному з яких, за припущенням, буде один і тільки один корінь.

Процедура - підпрограма відділення коренів

```
SUB Otdel (x0, xk, priz)
    DIM a AS DOUBLE, b AS DOUBLE
    PRINT USING "ищем корень на интервале [ ####.##
    ####.## ]"; x0; xk
    INPUT " задайте шаг"; h
    priz = 0
    a = x0
    DO
        b = a + h
        IF g(a) * g(b) < 0 THEN
            PRINT USING "ищем корень на [ ####.##
        ####.## ]"; a; b
            priz = 1
        END IF
        a = a + h:
    LOOP WHILE a < xk:
END SUB
```

3.2 Уточнення коренів

Розглянемо два чисельних методи уточнення коренів рівнянь: метод простих ітерацій та метод половинного ділення (вихідні дані – результат етапу відділення коренів).

3.2.1 Метод простих ітерацій

Маємо рівняння вигляду

$$x = f(x). \tag{3.2}$$

Беремо із інтервалу, отриманого на попередньому етапі, деяке x_0. Розглянемо числову послідовність $x_1 = f(x_0),\ x_2 = f(x_1),$ ..., що задається рекурентною формулою

$$x_{n+1} = f(x_n), \tag{3.3}$$

Якщо така послідовність має кінцеву границю, то ця границя буде коренем рівняння (3.2). Із вищої математики (за визначенням неперервної функції) відомо, що границя неперервної функції (при відповідному прямуванні аргументу) дорівнює значенню функції в точці:

$$\lim_{x \to a} f(x) = f(a). \qquad (3.4)$$

Доведено, що послідовність $\{x_n\}$ збігається, якщо виконується так звана *умова стискання*:

$$|f(x') - f(x'')| \le \alpha |x' - x''|, \qquad (3.5),$$

при будь-яких x', x'', що належать області визначення функції f, $\alpha \in (0, 1)$ – *коефіцієнт стискання.*

Приклад 3.4. Функція $f(x) = \frac{1}{2}\sin x$, $x \in (-\infty, +\infty)$ задовольняє умову стискання. Справді,

$$|\sin x' - \sin x''| = \frac{1}{2} \cdot 2 \left| \sin \frac{x' - x''}{2} \cos \frac{x' + x''}{2} \right| \le$$

$$\le \left| \sin \frac{x' - x''}{2} \right| \le \frac{1}{2} |x' - x''|, \quad \text{тобто } \alpha = \frac{1}{2} \in (0, 1).$$

Таким чином, на основі вищенаведеного може бути побудований алгоритмічний метод, що дозволяє визначити корінь рівняння (3.2) з будь-якою наперед заданою точністю. Такий алгоритм відомий як *метод простих ітерацій* (рисунок 3.3).

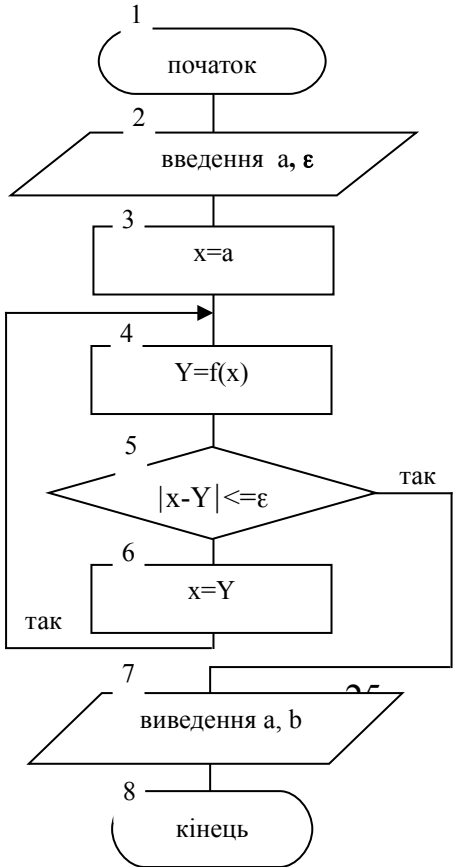


Рисунок 3.3 – Алгоритм методу простих ітерацій

Процедура - підпрограма методу простих ітерацій

Вихідні параметри: **x0** – початкове наближення розв'язку, **eps**- точність обчислення розв'язання.

Результат роботи: **c** – значення кореня рівняння**.**

```
SUB Simpleiteration (c AS DOUBLE)
    DIM x AS DOUBLE
    DIM a AS DOUBLE
    INPUT "начало интервала уточнения корня"; a
    INPUT " задайте точность"; eps
    x = a
    WHILE (ABS(x - f(x)) >= eps)
        x = f(x)
    WEND
    c = x
END SUB
'Процедура - функція f(x) для методу простих ітерацій
FUNCTION f (x)
     f = (1 / 2) * SIN(x)
END FUNCTION
```

3.2.2. Метод половинного ділення

Нагадаємо, що вихідні дані для цього методу є результат методу відділення коренів. Тоді функція f має один (і лише один) корінь на деякому сегменті $[a, b]$ (рисунок 3.4,а).

а) б) в)

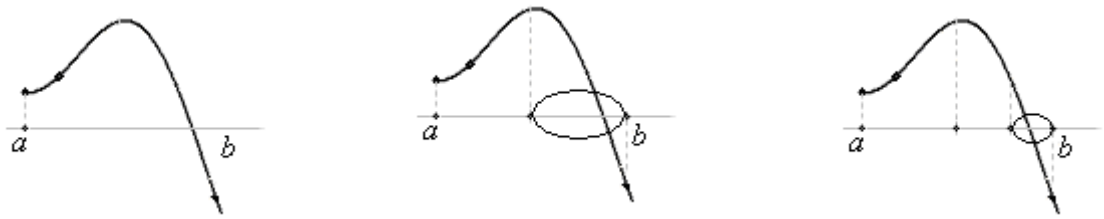


Рисунок 3.4

Розділимо цей сегмент $[a,b]$ навпіл і отримаємо два сегменти. Далі розглянемо той із отриманих сегментів, на якому лежить корінь функції (рисунок 3.4,б), та розділимо знову цей сегмент навпіл – отримаємо два сегменти. Після чого встановимо, на якому з останніх сегментів лежить корінь (рисунок 3.4,в). Будемо продовжувати цю процедуру ділення сегментів до тих пір, поки довжина чергового сегмента, на якому лежить корінь, не стане менше заданої точності $\varepsilon > 0$.

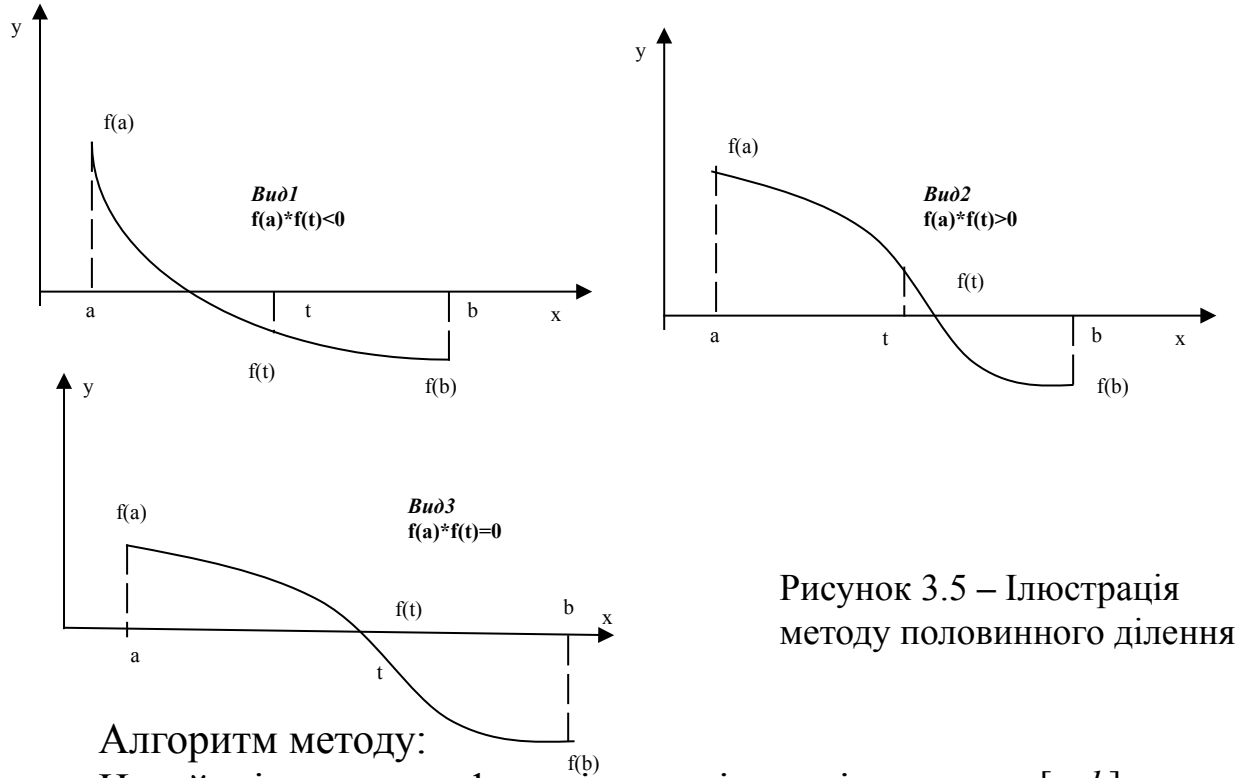


Рисунок 3.5 – Ілюстрація методу половинного ділення

Алгоритм методу:

Нехай відомо, що функція на кінцях інтервалу $[a,b]$ має значення різних знаків.

1) розділимо $[a,b]$ на дві рівні частини та визначимо значення функції у точці $t = \frac{a+b}{2}$;

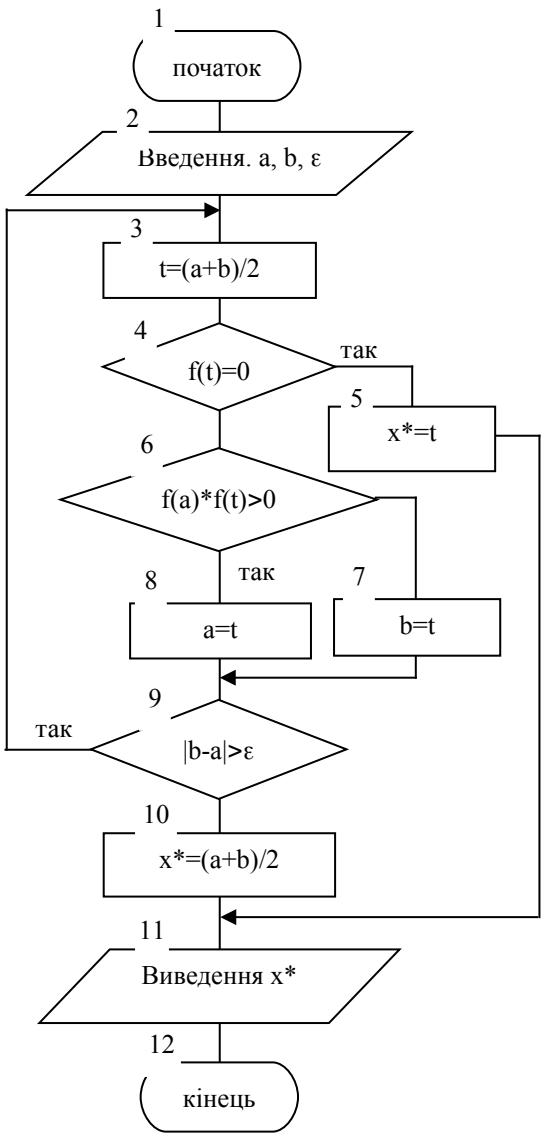


Рисунок 3.6 – Схема алгоритму методу половинного ділення

2) якщо f(t) = 0, то t — корінь рівняння; виведемо його значення (рисунок 3.5, вид 3). Інакше перейти до підп. 3;

3) аналізуємо знак добутку *f(a) * f(t);*

4) якщо цей добуток має від'ємне значення, то вважаємо b = t. Корінь буде на ділянці [*a*, *t*] (рисунок 3.5, вид 1);

5) якщо цей добуток має додатне значення, то вважаємо a = t. Корінь буде на ділянці **[***t***,** *b***]** (рисунок 3.5, вид 2);

6) аналізуємо виконання умови |b-a| > E, де E — точність уточнення кореня, і якщо вона виконується , то повертаємось до підп.1, у протилежному випадку закінчуємо алгоритм.

Схема алгоритму методу половинного ділення наведена на рисунку 3.6.

Процедура - підпрограма методу половинного ділення

Вихідні параметри: **a, b** – початкове та кінцеве значення інтервалу пошуку розв'язку, **eps** – точність обчислення розв'язку.
Результат роботи: **c** – значення кореня рівняння**.**

```
SUB Dihotom (c AS DOUBLE)
  DIM t AS DOUBLE, a AS DOUBLE, b AS DOUBLE
  INPUT "начало интервала уточнения корня"; a
  INPUT "конец интервала уточнения корня"; b
  INPUT " задайте точность"; eps
```

```
  DO
      t = (a + b) / 2
      IF g(a) * g(t) > 0 THEN
        a = t
      ELSE
        b = t
      END IF
    LOOP UNTIL b - a < eps
    c = (a + b) / 2
END SUB

'Процедура - функція f(x) для методу половинного
ділення
FUNCTION g (x AS DOUBLE)
        g = x-(1 / 2) * SIN(x)
END FUNCTION
```

Лабораторна робота № 4

МЕТОДИ ЧИСЕЛЬНОГО РОЗРАХУНКУ ВИЗНАЧЕНОГО ІНТЕГРАЛА

Визначений інтеграл ***I*** є одним з найважливіших понять вищої математики. Він широко використається у фізичних, геометричних та інженерних розрахунках, зокрема, при визначенні роботи механічних систем, площин геометричних фігур, об'ємів тіл і багато іншого.

$$I = \int_a^b f(x)dx \ . \qquad (4.1)$$

З геометричної точки зору визначений інтеграл являє собою площу криволінійної трапеції, яка обмежена віссю ***OX***, частиною графіка підінтегральної функції ***f(x)*** та прямими ***x=a*** і ***x=в*** (рисунок 4.1).

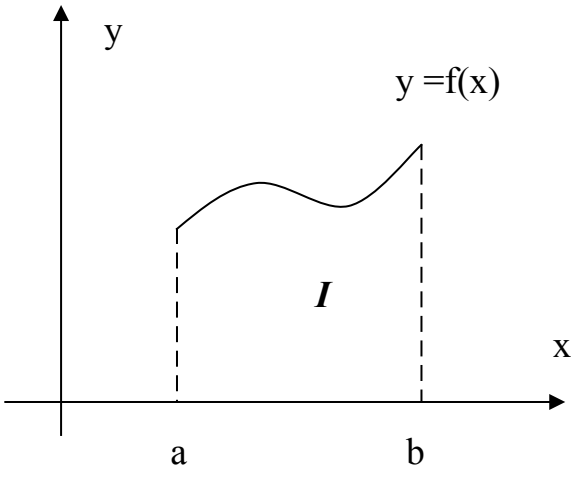


Рисунок 4.1 – Геометрична інтерпретація визначеного інтеграла

Нагадаємо декілька відомих з вищої математики понять, пов'язаних з визначеним інтегралом (докладне висвітлення цих питань можна знайти практично в кожному посібнику з інтегрального числення).

На рисунку 4.2 зображено графік функції f та схематично показані точки розбиття $\{x_i\}$, а також точки $\{\xi_i\}$.

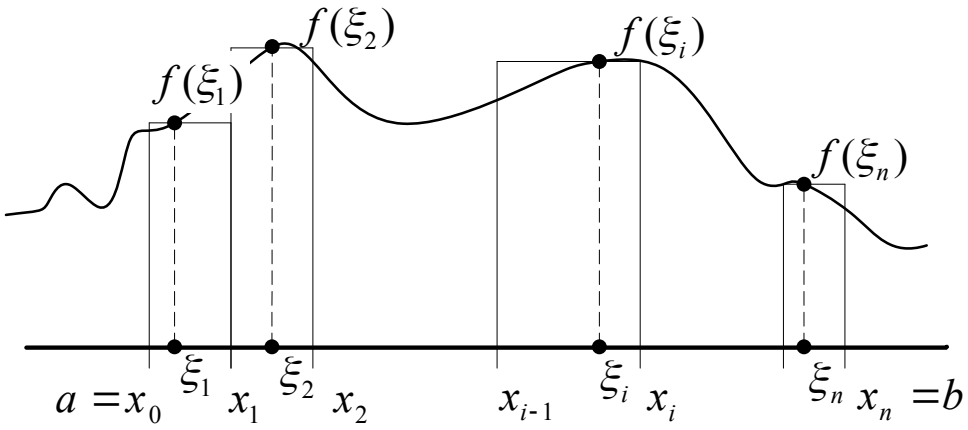


Рисунок 4.2

Визначення 4.1. Для функції $f(x)$, що визначена на проміжку $[a, b]$ сума

$$s_n = \sum_{i=1}^{n} f(\xi_i)(x_i - x_{i-1}) \qquad (4.2)$$

називається *інтегральною сумою Рімана*.

В (4.2) $\{x_i\}$ такими, що $a = x_0 < x_1 < \ldots < x_n = b$ - *розбиття* $[a, b]$, а $\xi_i \in [x_{i-1}, x_i]$, $i = 1, 2, \ldots, n$.

Визначення 4.2. Кажуть, що функція f *інтегрована на проміжку* $[a, b]$, якщо при будь-якому виборі точок $\{x_i\}$ і точок $\xi_i \in [x_{i-1}, x_i]$ існує кінцева границя інтегральних сум:

$$I = \lim_{max|\Delta x_i| \to 0} s_n = \lim_{max|\Delta x_i| \to 0} \sum_{i=1}^{n} f(\xi_i)(x_i - x_{i-1}),\ \Delta x_i = x_i - x_{i-1}.$$

Число ***I*** називають *визначеним інтегралом* (або *інтегралом Рімана*) і позначають символом $I = \int_a^b f(x)dx$.

Як видно з визначення, інтегральна сума Рімана залежить від вибору точок $\{x_i\}$, точок $\{\xi_i\}$ і функції $f(x)$. Інтегрованість

функції передбачає, що границя ***I*** не залежить від вибору точок $\{x_i\}$, $\{\xi_i\}$.

Таким чином, згідно з визначенням інтеграла, справедлива асимптотична рівність

$$I = \int_a^b f(x)dx \approx S_n, \ \max_i(x_i - x_{i-1}) \to 0 \qquad (4.3)$$

4.1 Метод прямокутників

Нехай *f (x)* інтегрована на проміжку $[a, b]$ функція (див. Визначення 4.1). Тоді $\int_a^b f(x)dx \approx S_n$ (див. формула (4.3)). Припустимо, що $\xi_i = x_i$, точки розбиття $x_i = a + hi$ (де $h = \frac{b - a}{n}$) Тоді в інтегральній сумі S_n, отримаємо, так звану, *формулу прямокутників* (4.4). В цьому методі крива підінтегральної функції замінюється ламаною лінією, що складається з відрізків, які паралельні осі абсцис з подальшим визначенням суми площин отриманих елементарних прямокутників (рисунок 4.3).

Наближене значення інтеграла визначається за формулою:

$$D = \int_a^b f(x)dx \approx \frac{b - a}{n}(y_1+y_2+...+y_n) = \frac{b - a}{n}\sum_{i=1}^{n} y_i, \qquad (4.4)$$

де y_i - значення $f(x_i)$ на початку кожного **i**-го відрізка; n – число відрізків інтегрування; a**,** b **-** нижня та верхня границі відрізків, на які розділено інтервал інтегрування; h**=**Δx

Алгоритм методу прямокутників наведено на рисунку 4.4.

Вихідні параметри: a, b – кінці інтервалу інтегрування; n – число доданків інтегральної суми.

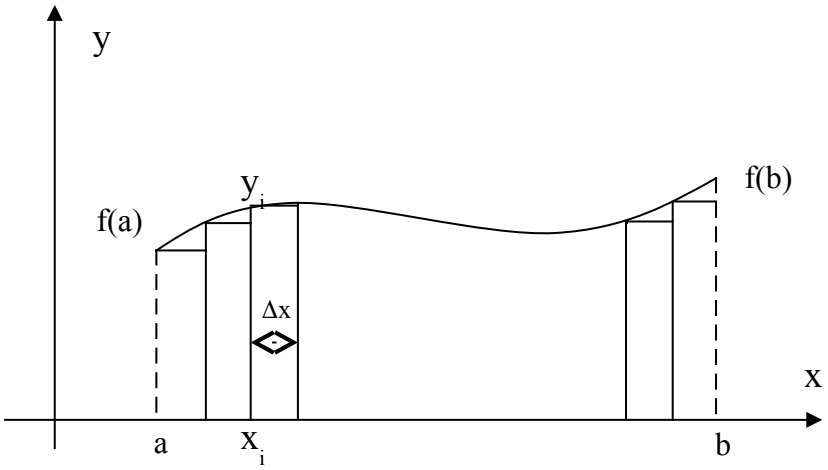
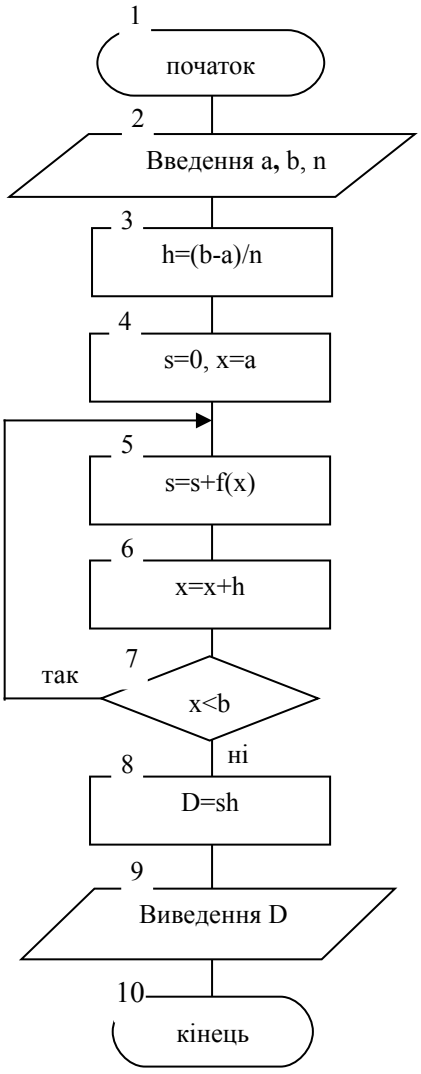


Рисунок 4.3 – Геометрична
інтерпретація методу прямокутників



Basic-програма(Метод прямокутників)

```
input "a, b, n"; a, b, n
h = (b-a)/n
s=0: x=a:
do while x<b
  s=s+f(x)
  x=x+h
loop
D = h*s:
print "D=";D

Function f(x)
  f = …
End Function
```

Рисунок 4.4 – Схема алгоритму
методу прямокутників

4.2 Метод трапецій

У методі трапецій інтервал інтегрування **[***a***,** *b***]** розділяється на n рівних відрізків довжиною $h = \frac{b - a}{n}$.

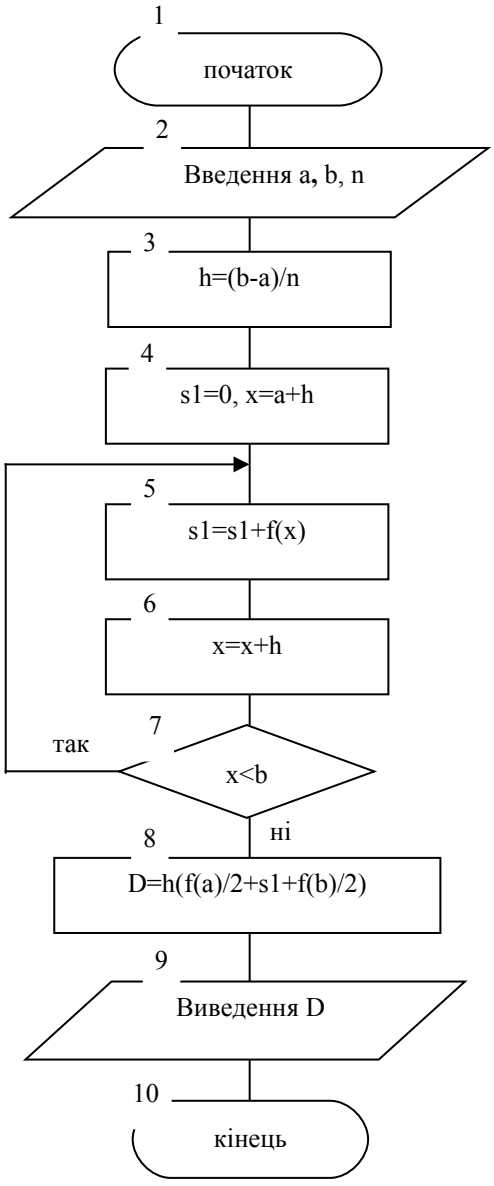


Рисунок 4.6 – Схема алгоритму методу трапецій

Відрізки ламаної з'єднують точки з координатами (x_i, y_i) і (x_{i+1}, y_{i+1}) та апроксимують підінтегральну функцію. Далі визначаємо суми площ отриманих елементарних трапецій (рисунок 4.5). Наближене значення інтеграла визначаємо за формулою

$$D = \int_a^b f(x)dx \approx$$

$$\approx \frac{b - a}{n}\left(\frac{y(a)}{2} + y_2 + y_3 + \ldots + \frac{y(b)}{2}\right), \qquad (4.5)$$

виходячи з того, що площа елементарної трапеції (рисунок 4.6) $S_i = \frac{y_i + y_{i+1}}{2}h$.

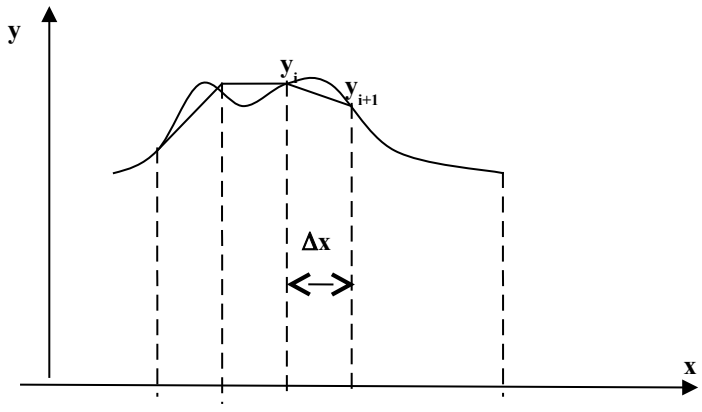


Рисунок 4.5 – Геометрична інтерпретація методу

Basic-програма (Метод трапецій)

```
input "a, b, n"; a, b, n

h=(b-a)/n
s=0: x=a+h
do while x<b
  s=s+f(x)
  x=x+h
loop

  D = h*((f(a)+ f(b))/2+s)
print "D=";D

Function f(x)
  f = …
End Function
```

4.3 Метод Сімпсона

В методі Сімпсона шуканий іінтеграл зображують у такому вигляді (рисунок 4.7):

$$\int_a^b f(x)dx = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)dx, \qquad (4.6)$$

де $x_i = a + hi$, $h = \frac{b - a}{n}$, n – натуральне число.

Далі підінтегральну функцію $f(x)$ у правій частині формули (4.6) (на проміжку $[x_{i-1}, x_i]$) замінюють інтерполяційним багаточленом 2-ї степені, що має такий вигляд (див. лабораторну роботу 2):

$$p_i(x) = \frac{(x - x_i)(x - \xi_i)}{(x_{i-1} - x_i)(x_{i-1} - \xi_i)} f(x_{i-1}) + \frac{(x - x_i)(x - \xi_i)}{(\xi_i - x_i)(\xi_i - x_{i-1})} f(\xi_i) + \frac{(x - x_{i-1})(x - \xi_i)}{(x_i - x_{i-1})(x_i - \xi_i)} f(x_i),$$

$$\xi_i = \frac{x_{i-1} + x_i}{2},$$

де ξ_i – середня точка проміжку $[x_{i-1}, x_i]$.

Застосувавши формулу Ньютона–Лейбниця до інтегралів $\int_{x_{i-1}}^{x_i} p_i(x)dx$, отримаємо

$$\int_{x_{i-1}}^{x_i} p_i(x)dx = \frac{1}{6}hf(x_{i-1}) + \frac{2}{3}hf(\xi_i) + \frac{1}{6}hf(x_i). \qquad (4.7)$$

Згідно з цими викладками маємо

$$\int_a^b f(x)dx \overset{(4.6)}{=} \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} f(x)dx \approx \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} p_k(x)dx \overset{(4.7)}{=}$$

$$= \frac{1}{3}\sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2} h + \frac{2}{3}h\sum_{i=1}^{n} f(\xi_i) = \frac{1}{3}\left[\frac{h}{2}f(a) + h\sum_{i=1}^{n} f(x_i) + \frac{h}{2}f(b) + 2h\sum_{i=1}^{n} f(\xi_i)\right] \qquad (4.8)$$

див. формулу трапеций

Таким чином, справедлива формула асимптотичного типу (формула Сімпсона)

$$\int_a^b f(x)dx \approx \frac{1}{3}\left[\frac{h}{2}f(a) + h\sum_{i=1}^{n} f(x_i) + \frac{h}{2}f(b) + 2h\sum_{i=1}^{n} f(\xi_i)\right], \text{ де } f \approx p_i$$

на $[x_{i-1}, x_i]$.

Підсумовуючи вищесказане, зробимо висновок, що визначений інтеграл обчислюється як сума площин елементарних криволінійних трапецій за формулою

$$D = \int_a^b f(x)dx \approx \frac{h}{3}\,(y(a)+4y_2+2y_3+\ldots 4y_{n-2}+2y_{n-1}+y(b)).$$

Алгоритм методу Сімпсона зручно розподілити на етапи (рисунок 4.8):

- знаходження значень функції з парними індексами
  - $S1=y_2+y_4+\ldots+y_{n-2}$;
- знаходження значень функції з непарними індексами
  - $S2=y_3+y_5+\ldots+y_{n-1}$;
- безпосередньо обчислення інтеграла

$$D = \int_a^b f(x)dx \approx \frac{h}{3}\,(y(a)+4S1+2S2+y(b)).$$

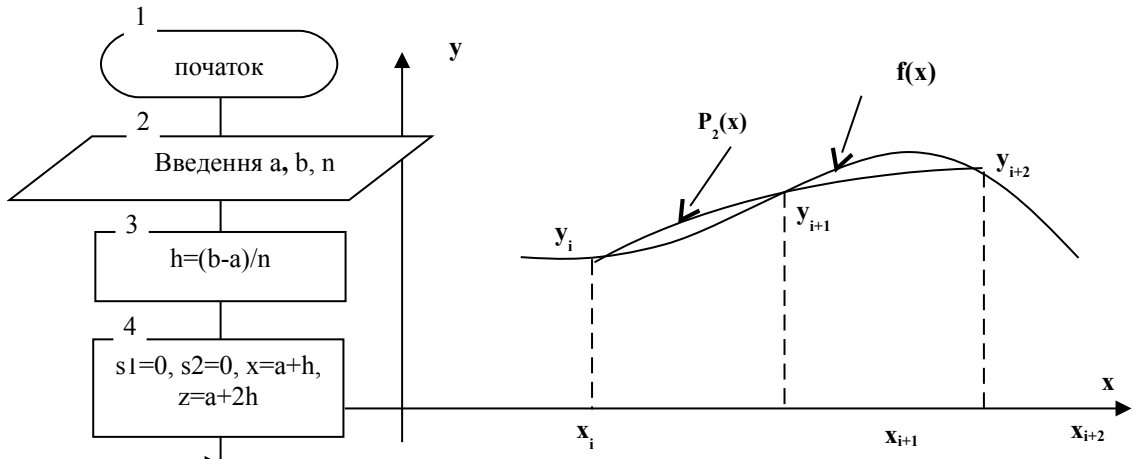


Рисунок 4.7 – Геометрична інтерпретація методу Сімпсона

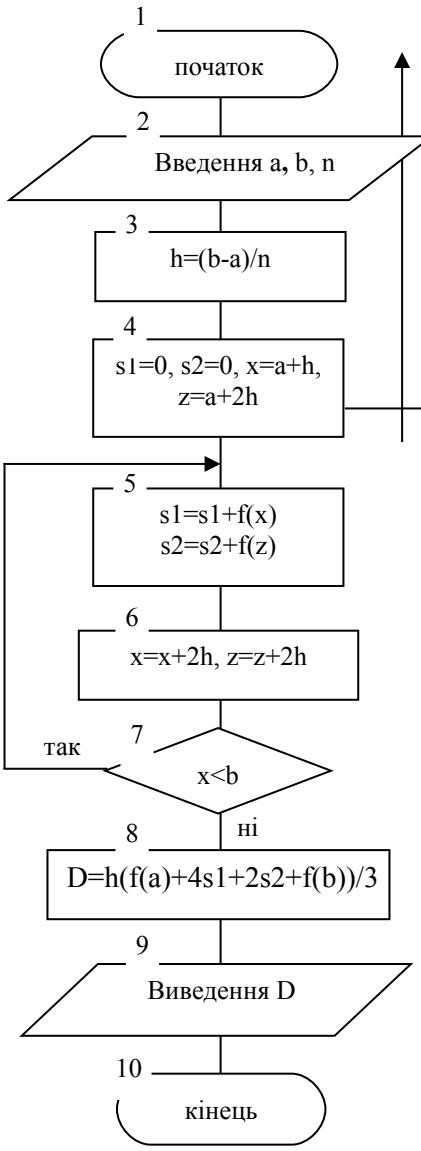


Рисунок 4.8 – Схема алгоритму методу Сімпсона

Через **x** в алгоритмі позначений аргумент, який відповідає парним номерам індексів, а через **z** – непарним номерам.

Basic-програма (метод Сімпсона**)**

```
input "a, b, n"; a, b, n
h = (b-a)/n
s1=0: s2=0
x=a+h:z=a+2*h
do while x<b
  s1=s1+f(x)
  s2=s2+f(z)
  x=x+2*h
  z=z+2*h
loop

D = h*(f(a)+ f(b)+2*s1+4*s2)/3
```

```
print "D=";D
```

Лабораторна робота № 5

ЧИСЕЛЬНЕ РОЗВ'ЯЗАННЯ ДИФЕРЕНЦІЙНИХ РІВНЯНЬ

Динамічні властивості різних природних і технічних систем такі, що рух у них відбувається за однаковими законами і описується однією і тією ж математичною моделлю – звичайними диференційними рівняннями (ЗДР).

У ЗДР *n*-го порядку як невідомі величини входять функція *y*(*x*) і її перші *n* похідних по аргументу **x**

$$F(x, y, y', y'', \ldots, y^{(n)}) = 0 \qquad (5.1)$$

Наприклад рівняння 1-го й 2-го порядку

$$F(x, y, y') = 0, \quad F(x, y, y', y'') = 0 \qquad (5.2)$$

У ряді випадків у рівняннях вигляду (5.1) вдається виразити старшу похідну в явному вигляді (нормальна форма Коші).

Наприклад, для (5.2)

$$y' = f(x, y), \qquad y'' = f(x, y, y') \qquad (5.3)$$

Надалі будемо розглядати рівняння вигляду (5.3).

Рівняння (5.1) еквівалентно системі ***n*** рівнянь 1-го порядку.

Наприклад, рівняння

$$Z'' = f(x, Z, Z') \qquad (5.4)$$

можна записати у вигляді системи рівнянь

$$Y_1' = f(x, Y_1, Y_2), \qquad (5.5)$$

$$Y_2' = Y_1,$$

де $Y_1 = Z_1; \quad Y_2 = Z_2$.

У загальному випадку система ЗДР має вигляд:

(5.6)

де y_1, y_2, ... y_n - змінні системи; x - аргумент, тобто величина, від якої залежать змінні системи (найчастіше такою величиною є час).

Як правило, доводиться мати справу із системою ЗДР 1-го порядку:

$$\begin{cases} \frac{dy_1}{dx} = f_1(x, y_1, y_2, \ldots, y_n) \\ \frac{dy_2}{dx} = f_2(x, y_1, y_2, \ldots, y_n) \\ \vdots \\ \frac{dy_n}{dx} = f_n(x, y_1, y_2, \ldots, y_n) \end{cases} \qquad (5.7)$$

або навіть із випадком, коли ***n*** = 1, тобто в системі є одна змінна і процес описується одним ЗДР.

$$\frac{dy}{dx} = f(x, y)\,. \qquad (5.8)$$

Щоб одержати уявлення про поведінку системи, необхідно знайти функцію $y(x)$, тобто ровязати ЗДР або систему ЗДР.

Потрібно проінтегрувати функцію $f(x,y)$, тобто знайти розвязок рівняння (5.8)

$$y(x) = \int f(x, y(x))dx + c\,, \qquad (5.9)$$

де c-довільна константа, при $y = y_0$.

ЗДР мають безліч розвязків – так званий загальний розвязок.

Для того щоб знайти єдиний (окремий) розвязок, потрібні додаткові умови, яким має задовольняти шуканий розвязок (рисунок 5.1).

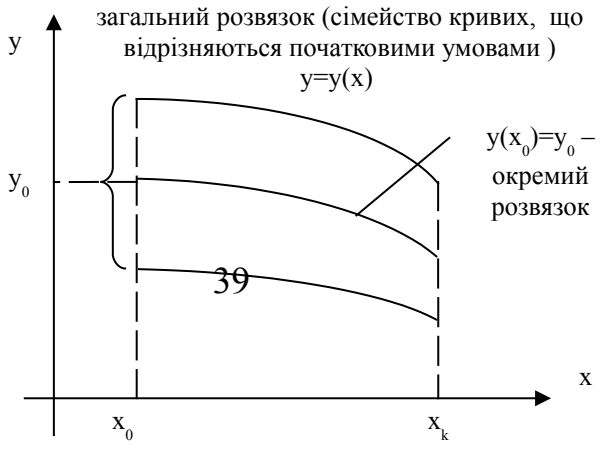


Рисунок 5.1 – Геометрична інтерпретація розв'язання ЗДР

5.1 Чисельні методи розвязання задачі Коші

При використанні чисельних методів розвязання ЗДР визначається не сама функція $y(x)$ (первісна) в аналітичному вигляді, а приблизно обчислюються її окремі значення y_i ($i = 1, 2, \ldots, n$) в заданих точках на $[x_0, x_к]$.

В основу чисельних методів покладена така процедура:

1.Задаються початкові умови $\mathbf{x_0}$; $\mathbf{y_0}$ і інтервал змінення аргументу x: $[x_0, x_к]$.

2. Точки, у яких обчислюються значення $y(x)$, задаються таким способом: інтервал $[x_0, x_k]$ розбивається на кінцеве число відрізків рівної довжини. Значення $x_0, x_1, \ldots, x_k$ і є заданими точками. Відстань між сусідніми точками h (довжина відрізка) називається кроком інтегрування (рисунок 5.2).

3. Значення $y(x)$ у точках $x_0, x_1, \ldots, x_k$ обчислюються за допомогою формули розкладання функції в ряд Тейлора:

$$y_{i+1} = y_i + \underbrace{h \cdot y_i' + \frac{h^2}{2!} y_i'' + \frac{h^3}{3!} y_i''' + \ldots}_{\text{Збільшення } \mathbf{y}}, \qquad (5.10)$$

де y_i, y_i', y_i'', **…** – значення $y(x)$ і її похідних в i -й точці, y_{i+1} – значення $y(x)$ в наступній, $(i+1)$-й точці. Значення $y(x)$ у кожній наступній точці обчислюється, виходячи зі значення $y(x)$ і її похідних у попередній точці за допомогою рекурентних співвідношень.

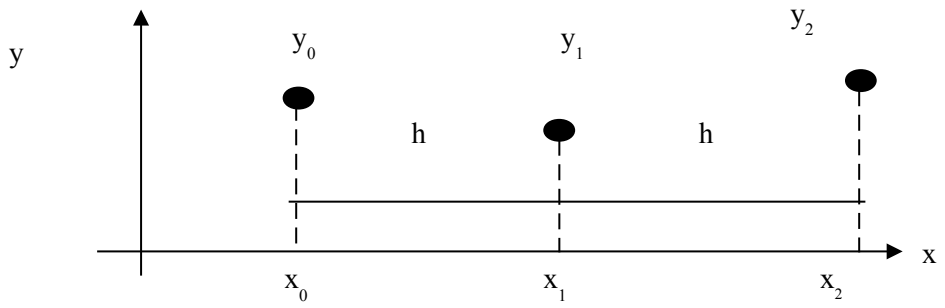


Рисунок 5.2 – Розбиття інтервалу інтегрування на відрізки

Методи визначення $y(x)$ відрізняються один від одного способами обчислення збільшення $y(x)$. Розглянемо однокрокові методи, які передбачають визначення y_{i+1} тільки через одне попереднє значення y_i

5.2 Метод Ейлера

У формулі розкладання функції в ряд Тейлора всі члени ряду, які містять похідні другого порядку й вище, відкидаються через їхню малість і за рахунок цього обчислення є наближеним (рисунок 5.3). Отже, формула набуває вигляду:

$$y_{i+1} = y_i + h \cdot y'_i \quad або \quad y_{i+1} = y_i + h \cdot f(x_i, y_i). \tag{5.11}$$

Для системи 2-х рівнянь формула (5.11) буде мати вигляд

$$\begin{aligned} y_{i+1} &= y_i + h \cdot f_1(x_i, y_i, z_i), \\ z_{i+1} &= z_i + h \cdot f_2(x_i, y_i, z_i). \end{aligned} \tag{5.12}$$

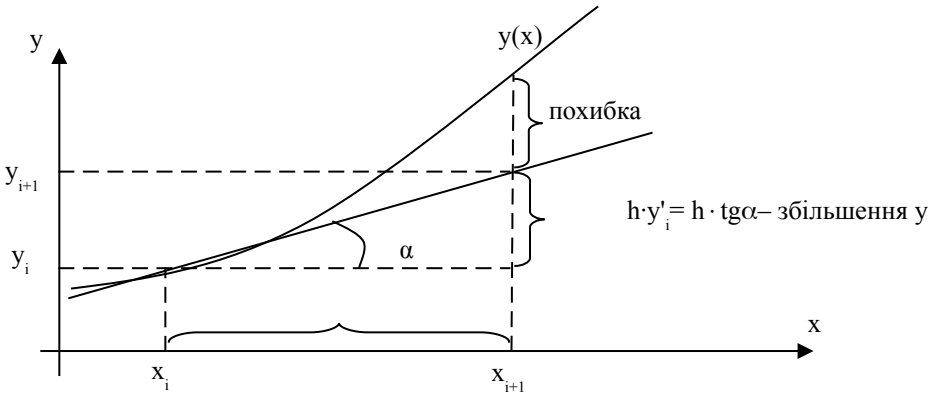


Рисунок 5.3 – Геометрична інтерпретація методу Ейлера

Відповідно до формули (5.11) функція $y(x)$ на кожному відрізку інтервалу інтегрування заміняється рівнянням прямої лінії, дотичної до кривої $y(x)$ на початку відрізка, тобто застосовується кусково-лінійна апроксимація. Погрішність методу має порядок h^2. Для підвищення точності потрібно зменшити h, але при цьому збільшується обсяг обчислень.

5.3 Метод Ейлера із проміжним виведенням результатів

У більшості інженерних завдань здійснюється обчислення функції $y(x)$ з малим кроком, а виведення значень із більшим кроком є зручним для побудови графіка цієї функції.

У таких випадках можна ввести параметр (назвемо його, наприклад**,** m), що дозволить здійснити виведення кожного m-го обчисленого значення. Алгоритм реалізації методу Ейлера для ЗДР наведений на рисунку 5.4, методу Ейлера із проміжним виведенням результатів для ЗДР – на рисунку 5.5, методу Ейлера для системи ЗДР – на рисунку 5.6.

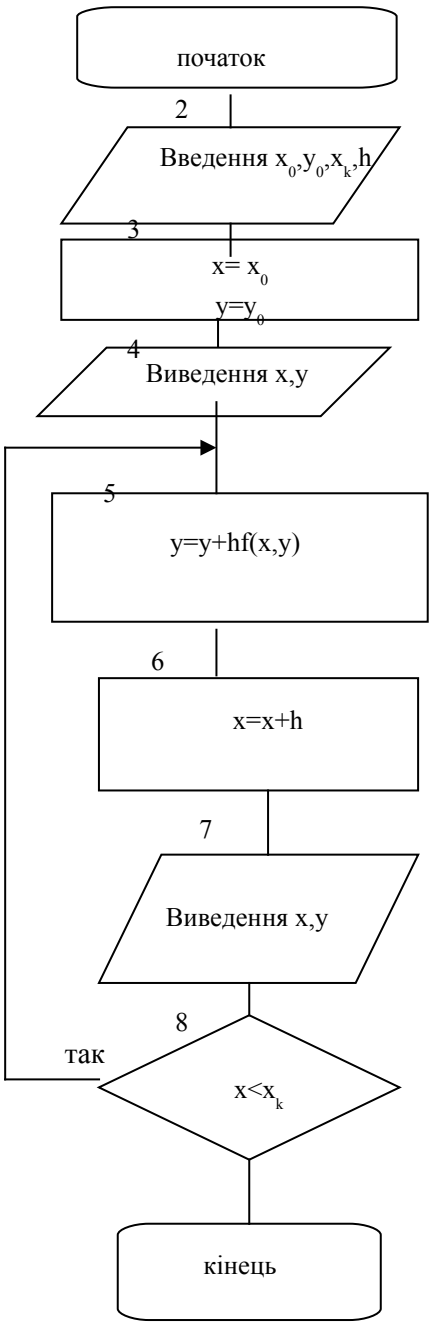


Рисунок 5.4 – Схема алгоритму методу Ейлера для ЗДР

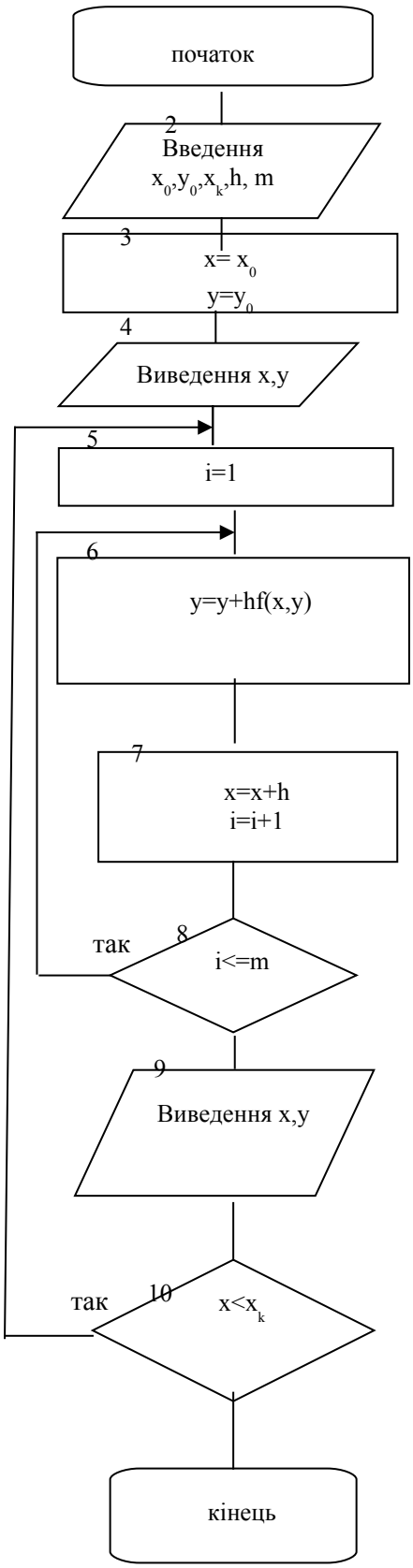


Рисунок 5.5 – Схема алгоритму методу Ейлера з проміжним виведенням результатів для ЗДР

Рисунок 4 Схема алгоритму методу Ейлера для ЗДР

Рисунок 5 Схема алгоритму методу Ейлера для ЗДР з проміжним виведенням результатів

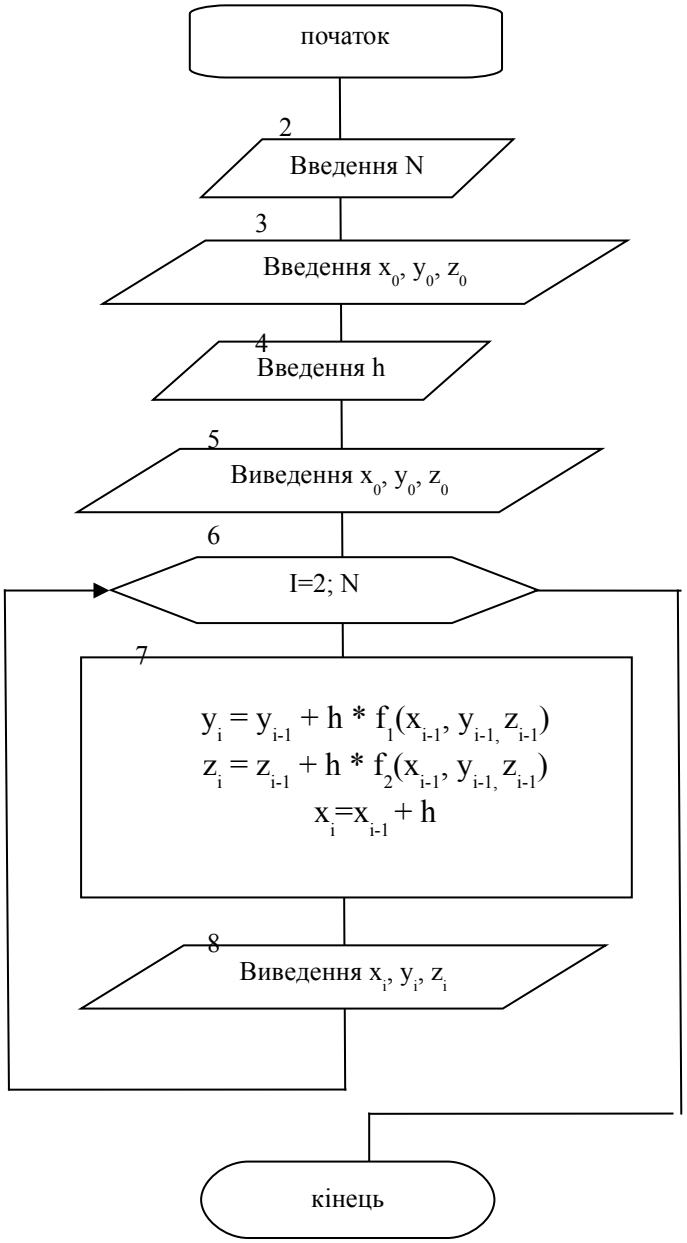


Рисунок 5.6 – Схема алгоритму методу Ейлера для ЗДР

5.4 Метод Рунге-Кутта

Цей метод є більш точним і більш розповсюдженим. Він використовує для обчислення похідних правої частини $f(x, y)$ схеми різного порядку точності й дає можливість при меншому обсязі обчислень більш точно обчислити звільшення ***y*** на кожному кроці.

Найбільш використовувана в обчислювальній практиці модифікація цього методу 4-го порядку на кожному кроці припускає обчислення правої частини рівняння у чотирьох сусідніх точках. Збільшення функції $y(x)$ для переходу від (x_i, y_i)

Рисунок 5.6 – Схема алгоритму методу Ейлера для системи ЗДР

до (x_{i+1}, y_{i+1}) визначається як усереднене значення збільшень у цих точках.

Уточнення досягається за рахунок спеціального підбору координат. Розрахункові формули:

$$y_{i+1} = y_i + (k_1 + 2k_2 + 2k_3 + k_4)/6, \tag{5.13}$$

де $i = 1, 2, \ldots, n$.

$$k_1 = hf(x_i, y_i);$$
$$k_2 = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right);$$
$$k_3 = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right);$$
$$k_4 = hf(x_i + h, y_i + k_3)$$

.

Для системи 2-х рівнянь формула (5.13) буде мати вигляд:

$$y_{i+1} = y_i + (k_1 + 2k_2 + 2k_3 + k_4)/6, \tag{5.14}$$
$$z_{i+1} = z_i + (l_1 + 2l_2 + 2l_3 + l_4)/6,$$

де $i = 1, 2, \ldots, n - 1$

$$k_1 = hf_1(x_i, y_i, z_i); \qquad l_1 = hf_2(x_i, y_i, z_i);$$
$$k_2 = hf_1\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, z_i + \frac{l_1}{2}\right); \qquad l_2 = hf_2\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, z_i + \frac{l_1}{2}\right);$$
$$k_3 = hf_1\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, z_i + \frac{l_2}{2}\right); \qquad l_3 = hf_2\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, z_i + \frac{l_2}{2}\right);$$
$$k_4 = hf_1(x_i + h, y_i + k_3, z_i + l_3) \qquad l_4 = hf_2(x_i + h, y_i + k_3, z_i + l_3)$$

.

Похибка методу становить близько h^5.

Схема алгоритму методу Рунге-Кутта для ЗДР наведена на рисунку 5.7, методу Рунге-Кутта для системи ЗДР – на рисунку 5.8.

Як ілюстрацію наведено:

а) програму 1, що реалізує алгоритм методу Рунге-Кутта для наступного ЗДР

$$y' = \sin 2x - \ln y + 0.32;$$

б) програму 2, що реалізує алгоритм методу Ейлера для наступної системи ЗДР.

$$\begin{cases} y' = \sin 2x - \ln y + 0.32 \\ z' = 3x - 2y + z \end{cases}.$$

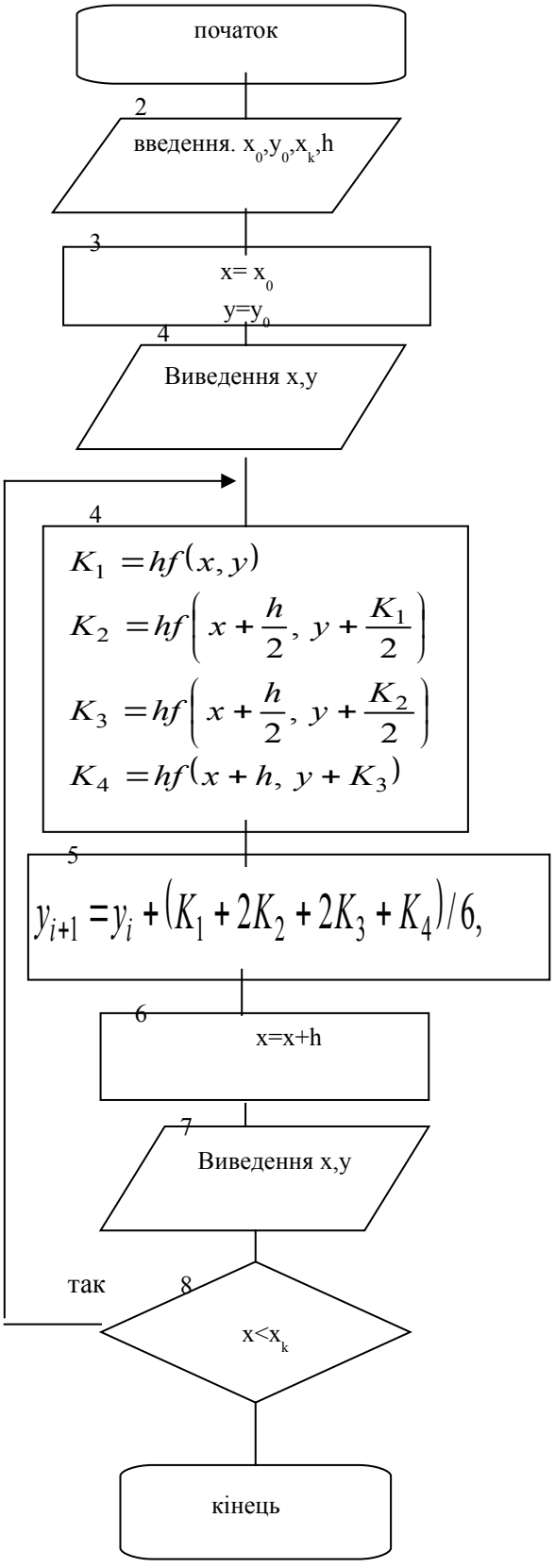


Рисунок 5.7 – Схема алгоритму методу Рунге-Кутта для ЗДР

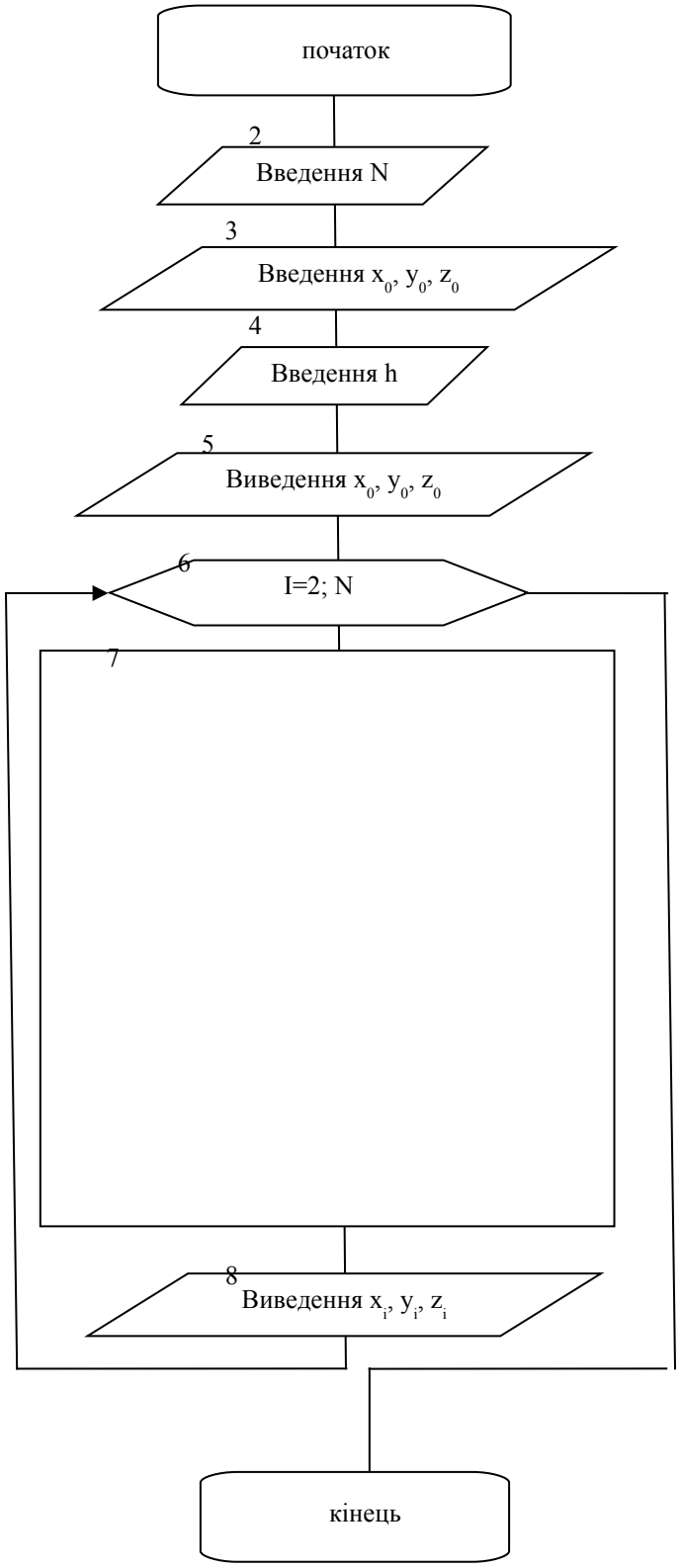


Рисунок 5.8 – Схема алгоритму методу Рунге-Кутта для системи ЗДР

Basic-програма (метод Рунге-Кутта)

```
REM Чисельне розвязання диференціального рівняння
методом Рунге-Кутта
REM Вихідне рівняння  y'  =sin (2x) - log(y)+0.32
DEF fnf (x,y)=sin(2*x)-log(y)+.32
CLS
INPUT "Початок інтервалу X0 ="; X0
INPUT " Початкове значення Y0 ="; Y0
INPUT "Кінець інтервалу Xk ="; XK
INPUT "Крок інтегрування H ="; H
X=X0: Y=Y0
PRINT " X="; X, " Y="; Y

  10 K1=H*fnf (X,Y)
  K2= H*fnf (X+H/2,Y+K1/2)
  K3= H*fnf (X+H/2,Y+K2/2)
  K4= H*fnf (X+H,Y+K3)
  Y=Y+(K1+2* K2+2* K3+ K4)/6
  X=X+H
  PRINT " X="; X, " Y="; Y

  IF X<XK THEN 10
```

Basic-програма (метод Ейлера)

```
REM Чисельне рішення системи диференціальних рівнянь
методом Ейлера
REM Вихідні рівняння  y'  = sin(2*x)-log(y)+.32
REM                z'=3x-2y+z
CLS
DEF FNF1 (X, Y, Z) = 2 * X - 3 * Y + 2 * Z
DEF FNF2 (X, Y, Z) = 3 * X - 2 * Y + Z
INPUT " N"; N
INPUT " XN,YN,ZN"; XN, YN, ZN
INPUT " H"; H
X = XN
Y = YN
Z = ZN
PRINT "    X     Y     Z"
PRINT USING "##.## ###.# ###.#"; X; Y; Z
FOR I = 2 TO N2
  Y = Y + H * FNF1(X, Y, Z)
  Z = Z + H * FNF2(X, Y, Z)
  X = X + H
PRINT USING "##.## ###.# ###.#"; X; Y; Z
NEXT I
```

Лабораторна робота № 6

МЕТОД ЗОЛОТОГО ПЕРЕТИНУ В ЗАДАЧАХ ОДНОПАРАМЕТРИЧНОЇ ОПТИМІЗАЦІЇ

Методи оптимізації дозволяють вибрати найкращий варіант системи, що розробляється з усіх можливих варіантів. Пошук оптимальних інженерних рішень є основним у завданні проектування і моделювання систем. Звичайно це процес або послідовність операцій, які дозволяють отримати уточнене рішення. Вираз, для значення якого інженер-дослідник має обчислити *max* або *min,* називається цільовою функцією**.**

Цільова функція дозволяє кількісно порівняти два альтернативні розвязки задачі оптимізації.

Прикладами цільових функцій, які часто зустрічаються в інженерній практиці, є *вартість, вага, міцність, габарити, ККД* тощо.

Незалежно від типу розв'язуваної задачі, на екстремум можна користуватися одним і тим же алгоритмом, тобто задачу мінімізації можна легко перетворити в задачу на знаходження *max*, змінивши знак цільової функції на зворотний.

У більшості задач, особливо потребуючих негайного розвязання (у режимі реального часу), основним критерієм застосовності того або іншого методу є час, що витрачається на розрахунки.

Із усього різноманіття відносно "швидких" методів зупинимося на методі *золотого перетину*, основні ідеї якого знайшли застосування в багатьох предметних галузях.

Цей метод передбачає розподіл інтервалу пошуку оптимуму точками, координати яких обчислюються за певним законом, обчислення значень цільової функції *Q(x)* в них, порівняння цих значень між собою і відкидання тієї частини інтервалу, на якій відсутній оптимум. При цьому не потрібно безперервності *Q(x)*, досить, щоб вона була *унімодальною*.

Функція, задана на $[a, b]$**,** називається унімодальною (рисунок 6.1), якщо існує єдина точка x* мінімуму **Q(x**)

$$\mathbf{Q(x^*)=MIN\ Q(x)}$$

a<=x<=b

і якщо для будь-яких двох точок $x_1, x_2 \in [a, b]$ виконуються співвідношення

$x_1<x_2<x^*$ (для усіх точок, що розташовані зліва точки мінімуму**⇒ $Q(x_1)>Q(x_2)$**

$x_2>x_1>x^*$ (для усіх точок, що розташовані справа точки мінімуму) **⇒ $Q(x_1)<Q(x_2)$**

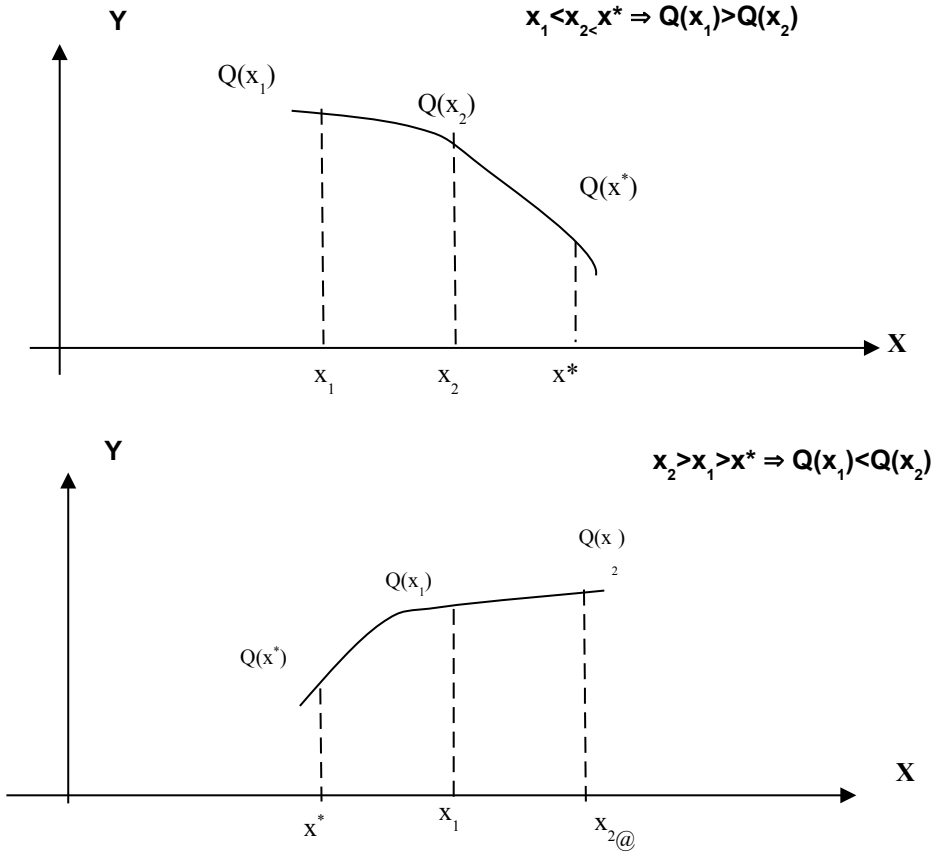


Рисунок 6.1 – Умови унімодальності функції

Нехай відомо, що *Q(x)* унімодальна на **[***a*, *b***]**. Тоді за будь-якими двома значеннями $Q(x_1)$ і $Q(x_2)$ можна вказати інтервал, у якому перебуває точка *x**, що мінімізує *Q(x)*. Цей інтервал має довжину, меншу довжини попереднього (рисунок 6.2). Для визначеності шукатимемо MIN (MAX визначається аналогічно).

Нехай для визначеності $\mathbf{x_1<x_2}$ Можливі 3 варіанти (рисунок 6.2).

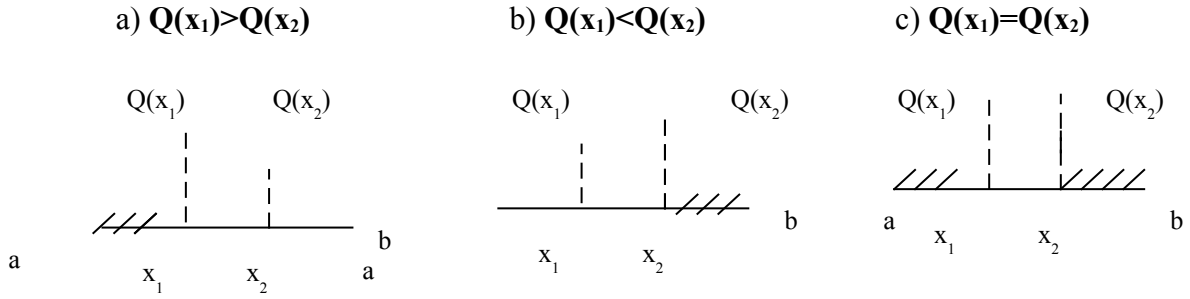


Рисунок 6.2 – Геометрична інтерпретація розбиття інтервалу пошуку мінімуму функції

Можливі три випадки:

a) варто відкинути інтервал $[a, x_1]$**;**

b) варто відкинути інтервал $[x_2, b]$**;**

c) варто відкинути інтервали $[a, x_1]$, бо на цих інтервалах не може перебувати *x**, виходячи з припущення про унімодальність *Q(x)*.

При заданій кількості обчислень *Q(x)* оптимальною є стратегія, що приводить до найменшого інтервалу невизначеності (інтервалу, на якому перебуває MIN).

На кожній ітерації будемо ділити $[a, b]$ точкою x_1 на дві частини так, щоб відношення довжини всього інтервалу до його більшої частини дорівнювало відношенню більшої частини до меншої частини (рисунок 6.3).

$$\frac{b-a}{b-x_1}=\frac{b-x_1}{x_1-a} \quad \text{або} \quad \frac{Z}{Z_1}=\frac{Z_1}{Z_2}$$

Із цього співвідношення $\mathbf{Z_1^2=Z\,Z_2}$. З урахуванням того, що $\mathbf{Z=Z_1+Z_2}$, підставимо **Z** з останнього виразу в попередній і поділимо на $\mathbf{Z_1^2}$. Одержимо

$$(\frac{Z_2}{Z_1})^2+\frac{Z_2}{Z_1}-1=0.$$

Для додатного кореня $\tau=\frac{Z_2}{Z_1}=0.618034$

Це відношення називається *золотим перетином.*

Зазначимо, що

$$\tau = \lim_{n\to\infty} \frac{v_{n-1}}{v_n} = 0.618034$$

$$v_1 = 1, \quad v_{n+1} = v_n + v_{n-1}.$$

За допомогою цієї формули можна легко визначити чергові значення послідовності $\{ \nu_n \}$. Маємо числа: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Ця послідовність чисел називається *числами Фібоначчі*.

Золотий перетин роблять дві точки: точка $x_1 = (b - a)\tau$ та точка x_2, розташована симетрично точці x_1 щодо середини відрізка $[a, b]$.

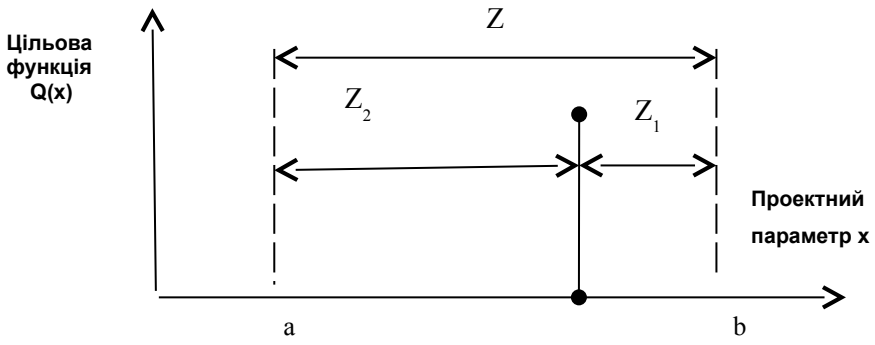


Рисунок 6.3 – Геометрична інтерпретація методу золотого перетину

Алгоритм методу Золотого перетину (рисунок 6.4)

1. Визначаємо значення точок x_1, x_2.

2 Обчислюємо значення $Q(x_1)$, $Q(x_2)$.

3 Якщо $Q(x_1) <= Q(x_2)$, то для подальшого розгляду залишаємо $[a, x_2]$, що еквівалентно переносу точки b у точку x_2.

4 Якщо $Q(x_1) > Q(x_2)$, то для подальшого розгляду залишаємо $[x_1, b]$, що еквівалентно переносу точки a у точку x_1.

Процес розподілу триває, поки довжина інтервалу невизначеності не стане менше заданої точності ε. На кожному кроці довжина нового інтервалу невизначеності приблизно дорівнює 0.618034 довжини старого інтервалу.

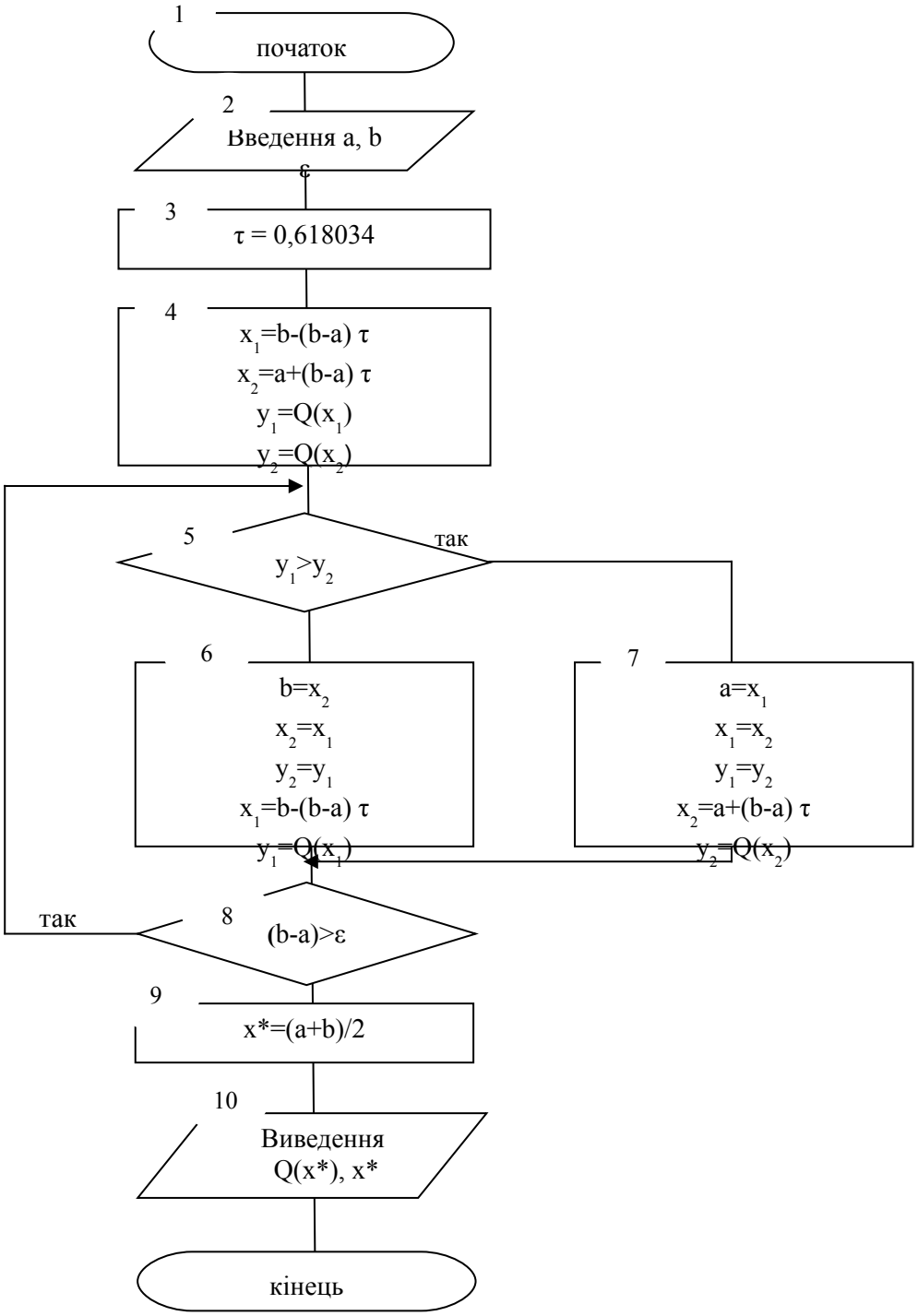


Рисунок 6.4 – Схема алгоритму методу золотого перетину

Підпрограма-функція, що реалізує метод Золотого перетину

```
SUB ZS (a, b, xr, N, m, eps)
```

```
  ' Подпрограмма метода золотого сечения
  t = .618034
  x1 = b - (b - a) * t
  x2 = a + (b - a) * t
  y1 = FC(x1, m)
  y2 = FC(x2, m)
  DO WHILE b - a > eps
   N = N + 1 'накопление числа итераций
   IF y1 > y2 THEN
       a = x1
       x1 = x2
       y1 = y2
       x2 = a + (b - a) * t
       y2 = FC(x2, m)
   ELSE
       b = x2
       x2 = x1
       y2 = y1
       x1 = b - (b - a) * t
       y1 = FC(x1, m)
   END IF
  LOOP
  xr = (a + b) / 2
END SUB

FUNCTION FC (x, m)
 f =
 IF m = 1 THEN f = -f  'm = 1 - ищем МАКСИМУМ,
  m = 2 - ищем МИНИМУМ
 FC = f
END FUNCTION
```

Лабораторна робота № 7

ДВОВИМІРНА ОПТИМІЗАЦІЯ МЕТОДАМИ ПОКООРДИНАТНОГО ТА ГРАДІЄНТНОГО СПУСКІВ

Опис *методу покоординатного спуску.*

Нехай треба знайти найменше значення цільової функції $u = f(x_1, x_2)$. Виберемо яку-небудь початкову точку $M_0 = (x_1^0, x_2^0)$, що належить області припустимих значень цієї функції, і розглянемо f як функцію змінної x_1 при фіксованому значенні другої змінної: $f(x_1, x_2^0)$. Тоді її можна вважати як функцію тільки однієї змінної x_1. Змінюючи цю змінну, будемо рухатися від початкової точки $x_1 = x_1^0$ у напрямку зменшення функції, доки не дійдемо до її мінімуму при $x_1 = x_1^1$, після якого вона починає зростати. Точку з координатами $\{x_1^1, x_2^0\}$ позначимо як M_1, при цьому $f(M_0) \geq f(M_1)$. Таку операцію називають простим скануванням.

Фіксуємо тепер змінну $x_1 = x_1^1$ й розглядаємо функцію f як функцію однієї змінної x_2: $f(x_1^1, x_2)$. Змінюючи x_2, будемо знов рухатися від початкового значення $x_2 = x_2^0$ у напрямку зменшення функції, доки не дійдемо до мінімуму при $x_2 = x_2^1$. Точку з координатами $\{x_1^1, x_2^1\}$ позначимо як M_2, при цьому $f(M_1) \geq f(M_2)$.

Процес продовжуємо далі: знову повернемося до змінної x_1 і повторимо раніше описані кроки. Такі дії цілком виправдовують назву методу. За їх допомогою ми знайдемо послідовність точок M_0, M_1, M_2, … , якій відповідає монотонна послідовність значень функції $f(M_0) \geq f(M_1) \geq f(M_2) \geq \ldots$ Обриваючи її у деякому кроці k, можна наближено прийняти значення функції $f(M_k)$ за її найменше значення у досліджуваній області.

Очевидно, що кожний крок методу, де проводиться знаходження найменшого значення функції за однією із вільних змінних при фіксованому значенні другої, можна здійснювати не шляхом вищеописаної примітивної процедури, а використовуючи який-небудь метод одновимірної оптимізації. Тоді, як вже зазначалося, цей метод зводить задачу пошуку найменшого значення функції декількох змінних до багаторазового розв'язання одновимірної задачі оптимізації.

Наочним доповненням до пояснення роботи методу є рисунок 7.1. На ньому зображені лінії рівня деякої функції двох змінних $u = f(x, y)$. Уздовж цих ліній функція зберігає постійні

значення, що дорівнюють 1, 3, 5, 7, 9. Також вказана траєкторія пошуку її найменшого значення, яке досягається у точці O, за допомогою методу покоординатного спуску. При цьому треба ясно розуміти, що рисунок служить тільки для ілюстрації методу. Коли ми приступаємо до розв'язання реальної задачі оптимізації, такого рисунка, що містить у собі готову відповідь, у нас немає.

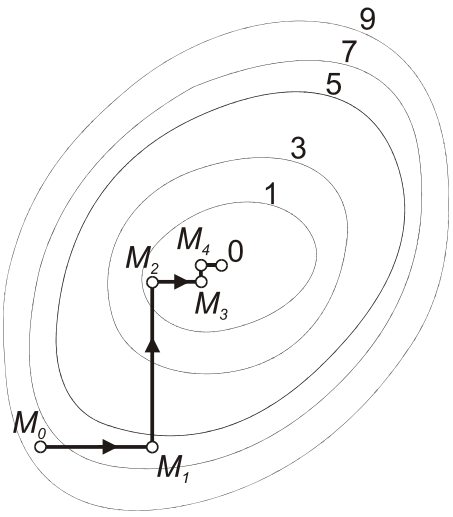


Рисунок 7.1

Узагальнимо обчислювальну схему методу для пошуку мінімуму функції в n-вимірному просторі R^n. Потрібно розв'язати задачу $f(x) \to \min$, $x \in R^n$. Розв'язання цієї задачі методом покоординатного спуску (інакше його називають методом Гауса-Зейделя) проводять так.

Вибирають довільно початкову точку $x^{(0)}$ із області визначення функції $f(x)$. Наближення $x^{(k)}$ визначаються співвідношеннями: $x^{(k+1)} = x^{(k)} + t^{(k)} \cdot S^{(k)}$, (k = 0, 1, 2, ...), де вектор напрямку спуску $S^{(k)}$ — це одиничний вектор, що збігається з яким-небудь координатним напрямом (наприклад, якщо $S^{(k)}$ рівнобіжний x_1, то $S^{(k)} = \{1,0,0, \ldots ,0\}$, якщо він рівнобіжний x_2, то $S^{(k)} = \{0,1,0, \ldots ,0\}$ і т.д.); величина $t^{(k)}$ є розв'язанням задачі одновимірної мінімізації:

$$f(x^{(k)} + t \cdot S^{(k)}) \to \min, \quad t \in R^1, \quad (k = 0, 1, 2, ...),$$

і може визначатися, зокрема, шляхом простого сканування (див. вище опис процедури пошуку мінімуму функції від двох змінних).

Припиняти обчислення можна по досягненні ітерації k, що була заздалегідь вказана, або ж по досягненні деякої заданої точності ε, коли виконуються нерівність $\| x^{(k+1)} - x^{(k)} \| < \varepsilon$ або нерівність $| f(x^{(k)}) - f(x^{(k+1)}) | < \varepsilon$.

Схема алгоритму пошуку мінімуму функції двох змінних методом покоординатного спуску показана на рисунку 7.2. Уведення даних для розрахунку й початкова підготовка змінних відбувається в блоках 2 – 8 відповідно до нумерації їх у схемі. Обчислення, які власне реалізують метод, визначені в блоках 9 – 25, виведення результатів пошуку задає блок 26. Для знаходження мінімального значення досліджуваної функції за одним з її аргументів при незмінному значенні іншого використовується одна спільна окрема процедура. За одну ітерацію процесу вона виконується двічі (блоки 11 та 15). На рисунку 7.3 наводиться схема алгоритму цієї процедури, що розроблена на основі методу простого сканування. Схема алгоритму цієї ж процедури, але де основою служить метод повного перебору значень функції при зміні її аргументу на заданий крок [10], подана на рисунку 7.4.

Крім дій, що здійснюють пошук мінімуму функції відповідно до методу, у схему алгоритму включені дії й обслуговуючі їх додаткові змінні, які дозволяють проводити дослідження процесу при заданні конкретних вхідних даних і закону обчислення функції. Так уведені в обчислення дії зі змінними *s* і *m*. Обидві вони є лічильниками (див. блоки 7, 15 на рисунку 7.2, блок 15 рисунка 7.3 і блок 10 рисунка 7.4). Змінна *s* фіксує кількість обчислених у процесі пошуку значень функції й може служити показником рівня обчислювальних витрат, що мають місце в даних дослідженнях. Змінна *m* лічить ітерації процесу (циклічні зміни пар етапів зменшення мінімального значення функції при зміні тільки аргументу x або тільки аргументу y). Показання цієї змінної можуть бути використані при порівнянні швидкості пошуку мінімуму функції для двох конкретних процесів.

У схему алгоритму уведені дії зі змінною *t*, яка відіграє роль перемикача при виборі критерію зупинки процесу пошуку. Забезпечена можливість застосовування кожного з названих вище способів припинення алгоритму. Використання змінної *t* як

перемикача між перевірками умови зупинки алгоритму задано операціями в блоках 2, 3, 10, 20, 22 на рисунку 7.2.

Обробка даних, що реалізована у програмі, яка складена за зазначеною схемою алгоритму, може використовуватись з демонстрацією проміжних результатів дослідження й без неї.

В цьому випадку наводяться тільки основні результати процесу пошуку, які не можуть показати особливості перебігу процесу. При введенні проміжних результатів аналіз останніх дозволить визначити траєкторію руху точки пошуку в області визначеності функції для кожного конкретного дослідження й порівняти їх. Управління виведенням результатів в алгоритмі й програмі проводиться за допомогою зміни значення змінної *z* (див. блоки 6, 12 і 18 на рисунку 7.2).

Текст програми, що реалізує мовою Basic алгоритм пошуку мінімуму функції відповідно до схеми на рисунку 7.2, наводиться нижче.

Якщо, наприклад, досліджувана функція задається формулою: $f(x, y) = (1 - x)^2 + (1 - y)^2$, то текст підпрограми-функції буде таким:

```
FUNCTION DvePerFun (x, y)
DvePerFun = (1 - x) ^ 2 + (1 - y) ^ 2
END FUNCTION
```

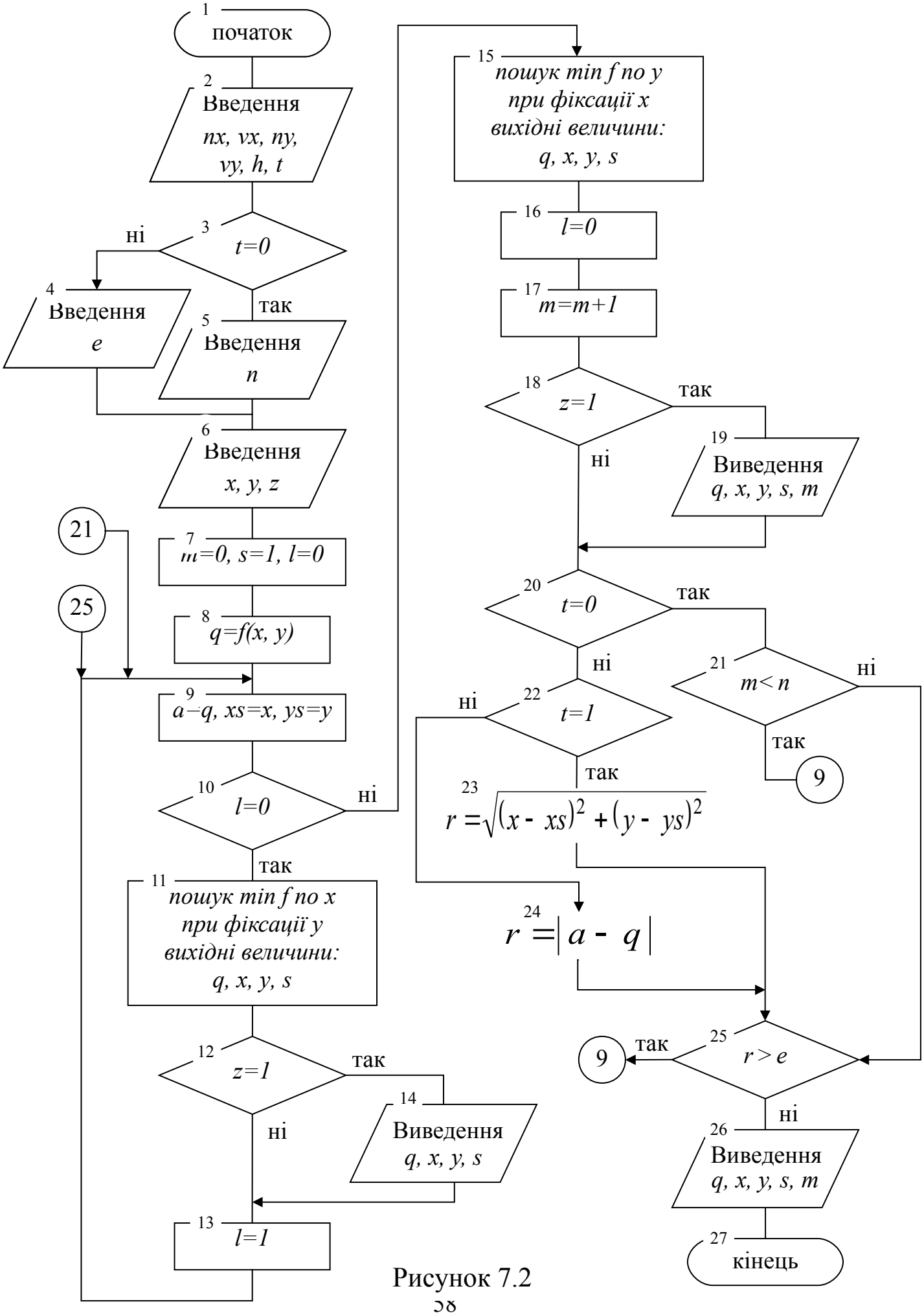

1 початок
2 Введення nx, vx, ny, vy, h, t
3 t=0
ні
так
4 Введення e
5 Введення n
6 Введення x, y, z
21
7 m=0, s=1, l=0
25
8 q=f(x, y)
9 a=q, xs=x, ys=y
10 l=0
ні
так
11 пошук min f по x при фіксації y вихідні величини: q, x, y, s
12 z=1
так
ні
14 Виведення q, x, y, s
13 l=1
15 пошук min f по y при фіксації x вихідні величини: q, x, y, s
16 l=0
17 m=m+1
18 z=1
так
ні
19 Виведення q, x, y, s, m
20 t=0
так
ні
21 m< n
ні
так
9
22 t=1
ні
так
23 $r=\sqrt{(x-xs)^2+(y-ys)^2}$
24 $r=|a-q|$
9
так
25 r > e
ні
26 Виведення q, x, y, s, m
27 кінець

Рисунок 7.2

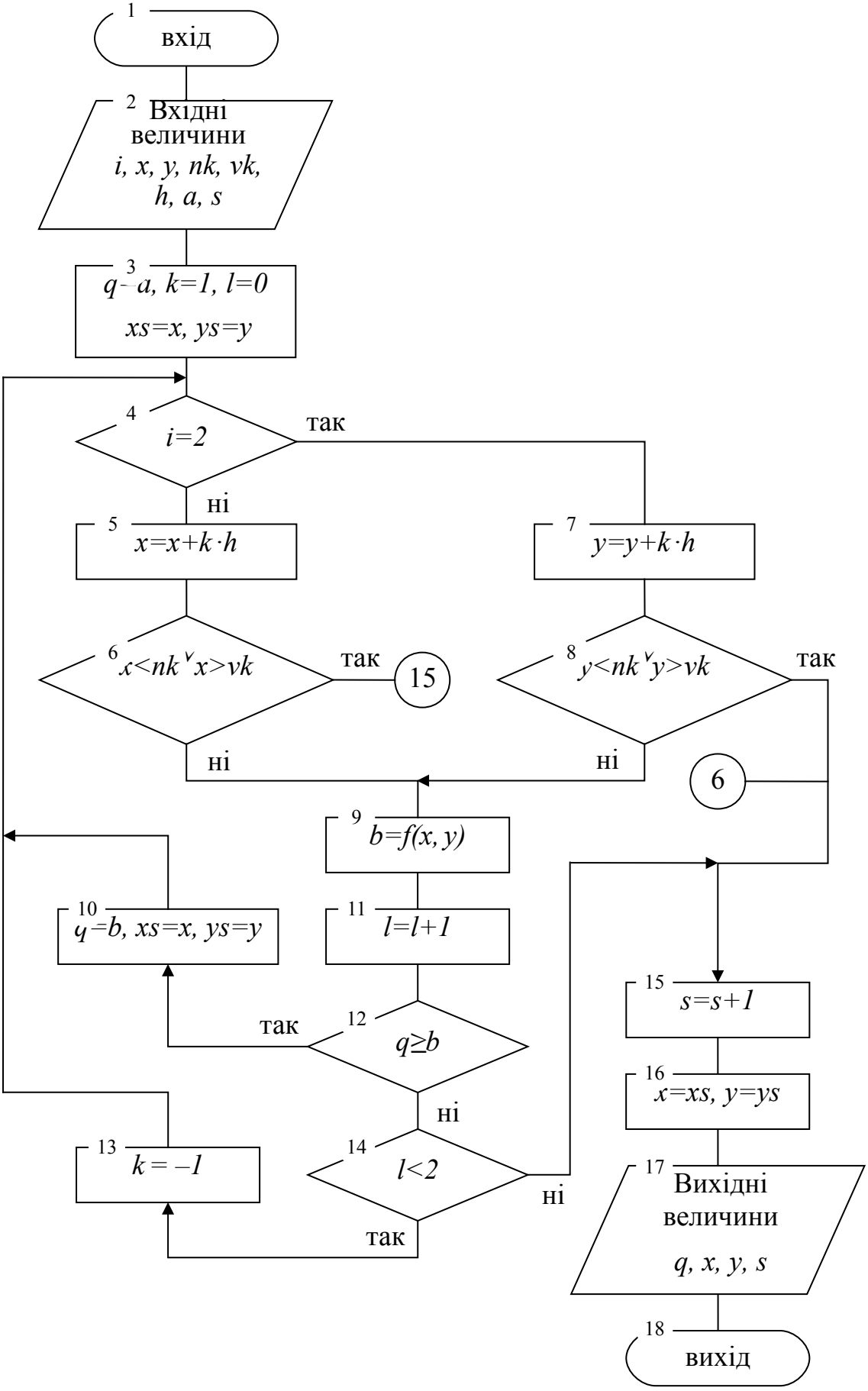

1
вхід
2
Вхідні величини
i, x, y, nk, vk,
h, a, s
3
q=a, k=1, l=0
xs=x, ys=y
4
i=2
так
ні
5
x=x+k·h
7
y=y+k·h
6
x<nk ˅ x>vk
так
15
8
y<nk ˅ y>vk
так
ні
ні
6
9
b=f(x, y)
10
q=b, xs=x, ys=y
11
l=l+1
15
s=s+1
так
12
q≥b
16
x=xs, y=ys
ні
13
k = –1
14
l<2
ні
17
Вихідні величини
q, x, y, s
так
18
вихід

Рисунок 7.3

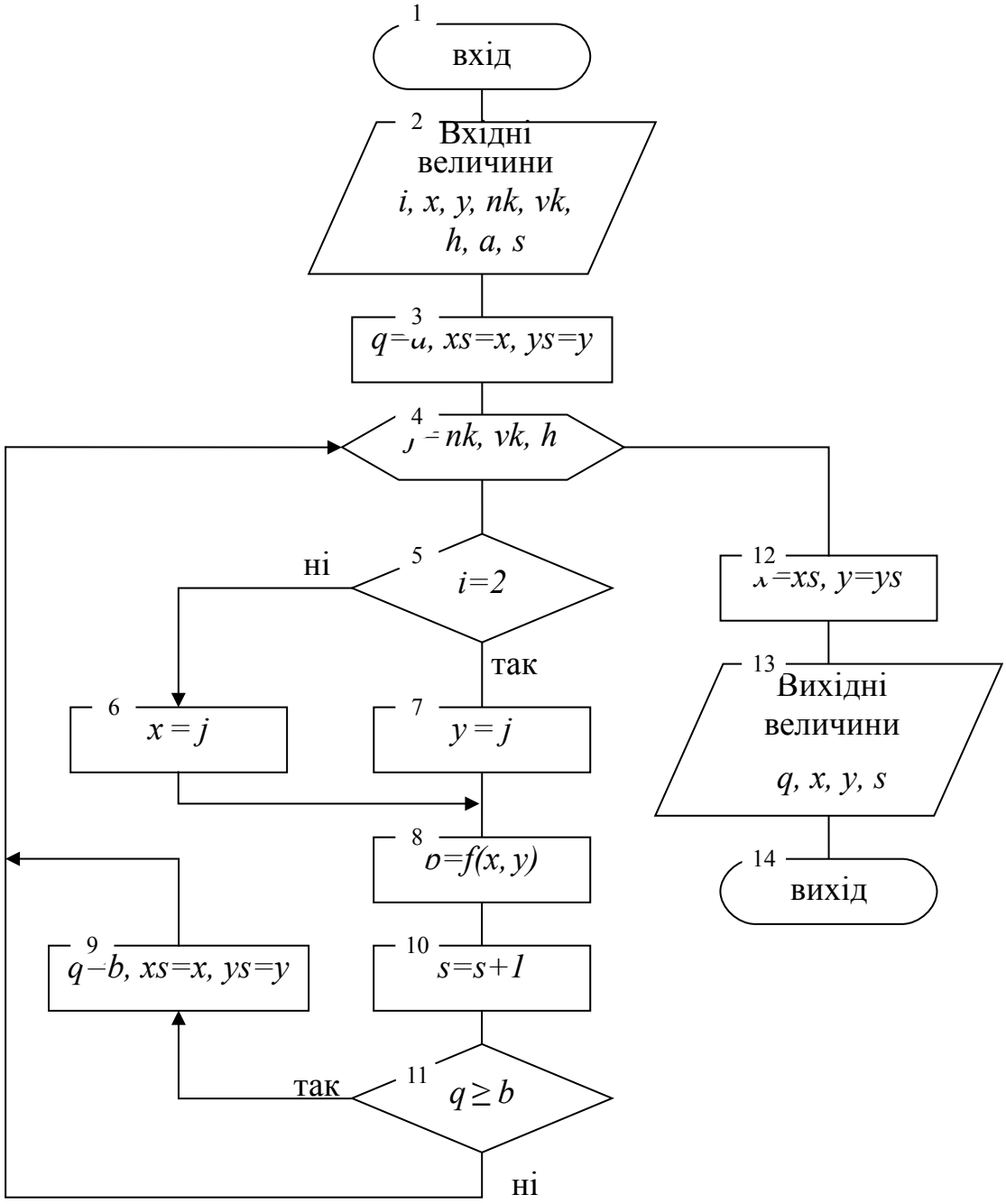

1
вхід
2
Вхідні величини
i, x, y, nk, vk, h, a, s
3
q=u, xs=x, ys=y
4
nk, vk, h
5
i=2
ні
так
6
x = j
7
y = j
8
f(x, y)
9
q−b, xs=x, ys=y
10
s=s+1
11
q ≥ b
так
ні
12
=xs, y=ys
13
Вихідні величини
q, x, y, s
14
вихід

Рисунок 7.4

Basic-програма, яка реалізує алгоритм, що наведений на рисунку 7.2 [3]

```
'Поиск минимума функции методом покоординатного
спуска
CLS
INPUT "Нижний и верхний пределы изменения координаты
                                    ↳X функции:", nx, vx
INPUT "Нижний и верхний пределы изменения координаты
                                    ↳Y функции:", ny, vy
INPUT "Укажите шаг исследования функции:", h
PRINT "Укажите критерий остановки поиска минимума:"
PRINT "0 - по количеству итераций"
PRINT "1 - по близости точек аргумента минимума
                    ↳функции между итерациями"
PRINT "2 - по близости значений минимума функции
                                ↳между итерациями"
INPUT "Номер критерия:", t
IF t = 0 THEN
INPUT "Введите количество допустимых итераций:", n
ELSE
INPUT "Введите точность поиска минимума:", e
END IF
INPUT "Введите координаты начальной точки поиска -
                                        ↳X,Y:", x, y
INPUT "Выводить ли промежуточные результаты? (0-нет,
                                          ↳1-да) ", z
m = 0
q = DvePerFun(x, y)
s = 1
l = 0
vozvrat: a = q: xs = x: ys = y
IF l = 0 THEN
CALL minpokoor(1, x, y, nx, vx, h, a, q, s)
IF z = 1 THEN
PRINT "x = "; x; " y = "; y; " текущий min функции =
                                                ↳"; q
PRINT "количество вычисленных значений функции = "; s
END IF
l = 1
GOTO vozvrat
```

---

[3] Знак ↳ показує, що рядок, де він перебуває, є продовженням попереднього.

```
ELSE
CALL minpokoor(2, x, y, ny, vy, h, a, q, s)
l = 0
m = m + 1
IF z = 1 THEN
PRINT "x = "; x; " y = "; y; " текущий min функции =
                                                    ↳"; q
PRINT "количество вычисленных значений функции = "; s
PRINT "Конец "; m; "-й итерации"
INPUT "Для продолжения нажмите Enter", zz
END IF
IF t = 0 THEN
IF m < n THEN GOTO vozvrat
ELSEIF t = 1 THEN
r = SQR((x - xs) ^ 2 + (y - ys) ^ 2)
IF r > e THEN GOTO vozvrat
ELSE
r = ABS(a - q)
IF r > e THEN GOTO vozvrat
END IF
END IF
PRINT "Результат поиска:"
PRINT "     минимальное значение функции = "; q
PRINT "     при x = "; x; "  и  y = "; y
PRINT "          количество вычисленных значений
функции = "; s
PRINT "          количество итераций = "; m
END
```

Basic-програма, яка реалізує алгоритм, що наведений на рисунку 7.3

```
SUB minpokoor (i, x, y, nk, vk, h, a, q, s)
q = a: xs = x: ys = y
FOR j = nk TO vk STEP h
IF i = 2 THEN y = j ELSE x = j
b = DvePerFun(x, y)
s = s + 1
IF q >= b THEN
q = b: xs = x: ys = y
END IF
NEXT j
x = xs: y = ys
END SUB
```

Basic-програма, яка реалізує алгоритм, що наведений на рисунку 7.4

```
SUB minpokoor (i, x, y, nk, vk, h, a, q, s)
q = a: xs = x: ys = y
FOR j = nk TO vk STEP h
IF i = 2 THEN y = j ELSE x = j
b = DvePerFun(x, y)
s = s + 1
IF q >= b THEN
q = b: xs = x: ys = y
END IF
NEXT j
x = xs: y = ys
END SUB
```

Метод градієнтного спуску

Основна його ідея полягає в тому, щоб при пошуку мінімуму функції рухатися в напрямку найбільш швидкого убування функції, який визначається антиградієнтом [10, 18].

Нагадаємо, що вектор ($\partial f/\partial x_1$, $\partial f/\partial x_2$) називається *градієнтом* функції $f(x_1, x_2)$, і позначається символом grad f, тобто

$$\text{grad}\, f(x_1, x_2) = \boldsymbol{i} \cdot \partial f(x_1, x_2)/\partial x_1 + \boldsymbol{j} \cdot \partial f(x_1, x_2)/\partial x_2 ,$$

де $\boldsymbol{i}$, $\boldsymbol{j}$ — одиничні вектори, які є рівнобіжними координатним осям. Частинні похідні характеризують змінення функції f за кожним аргументом окремо. Утворений за їх допомогою вектор градієнта дає загальне уявлення про поведінку функції в околі точки (x_1, x_2). Напрямок цього вектора є напрямком найбільш швидкого зростання функції в даній точці. Протилежний йому напрямок, що часто називають антиградієнтним, являє собою напрямок найбільш швидкого убування функції. Модуль градієнта

$$|\,\text{grad}\, f(x_1, x_2)\,| = \sqrt{\left[\partial f(x_1, x_2)/\partial x_1\right]^2 + \left[\partial f(x_1, x_2)/\partial x_2\right]^2}$$

визначає швидкість зростання й убування функції в напрямку градієнта й антиградієнта. Для всіх інших напрямків швидкість зміни функції в точці (x_1, x_2) менше модуля градієнта. При переході від однієї точки до іншої як напрямок градієнта, так і його модуль, як правило, змінюються.

Ідея використати градієнт у пошуку мінімуму функції реалізується так. Виберемо будь-яким способом початкову точку з області, де визначена функція, обчислимо в ній градієнт цієї функції й зробимо невеликий крок у зворотному, антиградієнтному напрямі. У результаті ми потрапимо в точку, у якій значення функції буде менше початкового. У цій точці повторимо процедуру: знову обчислимо градієнт функції й зробимо крок у зворотному напрямі. Продовжуючи цей процес, ми будемо рухатися у бік зменшення функції. Спеціальний вибір напрямку руху на кожному кроці дозволяє сподіватися на те, що в цьому випадку наближення до найменшого значення функції буде більш швидким, ніж у методі покоординатного спуску.

У практиці змінення значень аргументів для нового кроку пошуку мінімуму функції на величини, зворотні значенням складових градієнта, є не завжди зручним. Часто потрібно при визначенні чергових значень координат нової точки використовувати множник *h*, який регулює швидкість спуску. Тоді операцію обчислення координат точки для k+1-го кроку процесу пошуку можна задати формулами:

$$x_1^{(k+1)} = x_1^{(k)} - h \cdot \frac{\partial f\left(x_1^{(k)}, x_2^{(k)}\right)}{\partial x_1^{(k)}}, \quad x_2^{(k+1)} = x_2^{(k)} - h \cdot \frac{\partial f\left(x_1^{(k)}, x_2^{(k)}\right)}{\partial x_2^{(k)}},$$

де $x_1^{(k)}$, $x_2^{(k)}$ — координати k-го кроку процесу.

Щоб зупинити процес пошуку мінімуму функції на основі методу градієнтного спуску, можна скористатися тими ж критеріями, які застосовуються при організації пошуку мінімуму функції методом покоординатного спуску. Крім того, для завершення процесу пошуку мінімуму можливо відстеження кроку, на якому функція почне зростати.

Метод градієнтного спуску вимагає обчислення градієнта цільової функції на кожному кроці. Якщо вона задана аналітично, то для частинних похідних, що визначають градієнт, доцільно отримати явні формули. У противному випадку частинні похідні в потрібних точках доводиться обчислювати приблизно, заміняючи їх відповідними різницевими відношеннями:

$$\frac{\partial f(x_1, x_2)}{\partial x_1} \approx \frac{f(x_1 + \Delta x_1, x_2) - f(x_1, x_2)}{\Delta x_1}; \quad \frac{\partial f(x_1, x_2)}{\partial x_2} \approx \frac{f(x_1, x_2 + \Delta x_2) - f(x_1, x_2)}{\Delta x_2}.$$

Зазначимо, що при таких розрахунках Δx_1, Δx_2 не можна брати занадто малими, а значення функції потрібно обчислювати з досить високим ступенем точності, інакше при обчисленні різниць

$$f(x_1 + \Delta x_1, x_2) - f(x_1, x_2), \quad f(x_1, x_2 + \Delta x_2) - f(x_1, x_2)$$

будуть допущені великі помилки.

На рисунку 7.5 зображені лінії рівня тієї ж функції двох змінних $u = f(x_1, x_2)$, що й на рисунку 7.1, і також наведена траєкторія пошуку її мінімуму за допомогою методу градієнтного спуску.

Порівняння рисунків 7.1 і 7.5 показує, наскільки більш ефективним є метод градієнтного спуску.

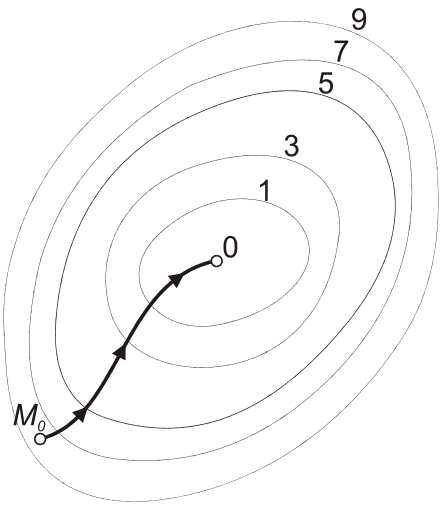


Рисунок 7.5

Схема алгоритму пошуку мінімуму функції двох змінних методом градієнтного спуску наводиться на рисунку 7.6. За своєю структурою вона подібна до схеми рисунка 7.2: блоки 2 – 9 визначають дії з уведення початкових даних і підготовки до роботи основних змінних, у блоках 10 – 28 визначені усі обчислення, що відносяться безпосередньо до проведення пошуку мінімуму функції відповідно методу градієнтного спуску, блок 29 визначає виведення результатів пошуку.

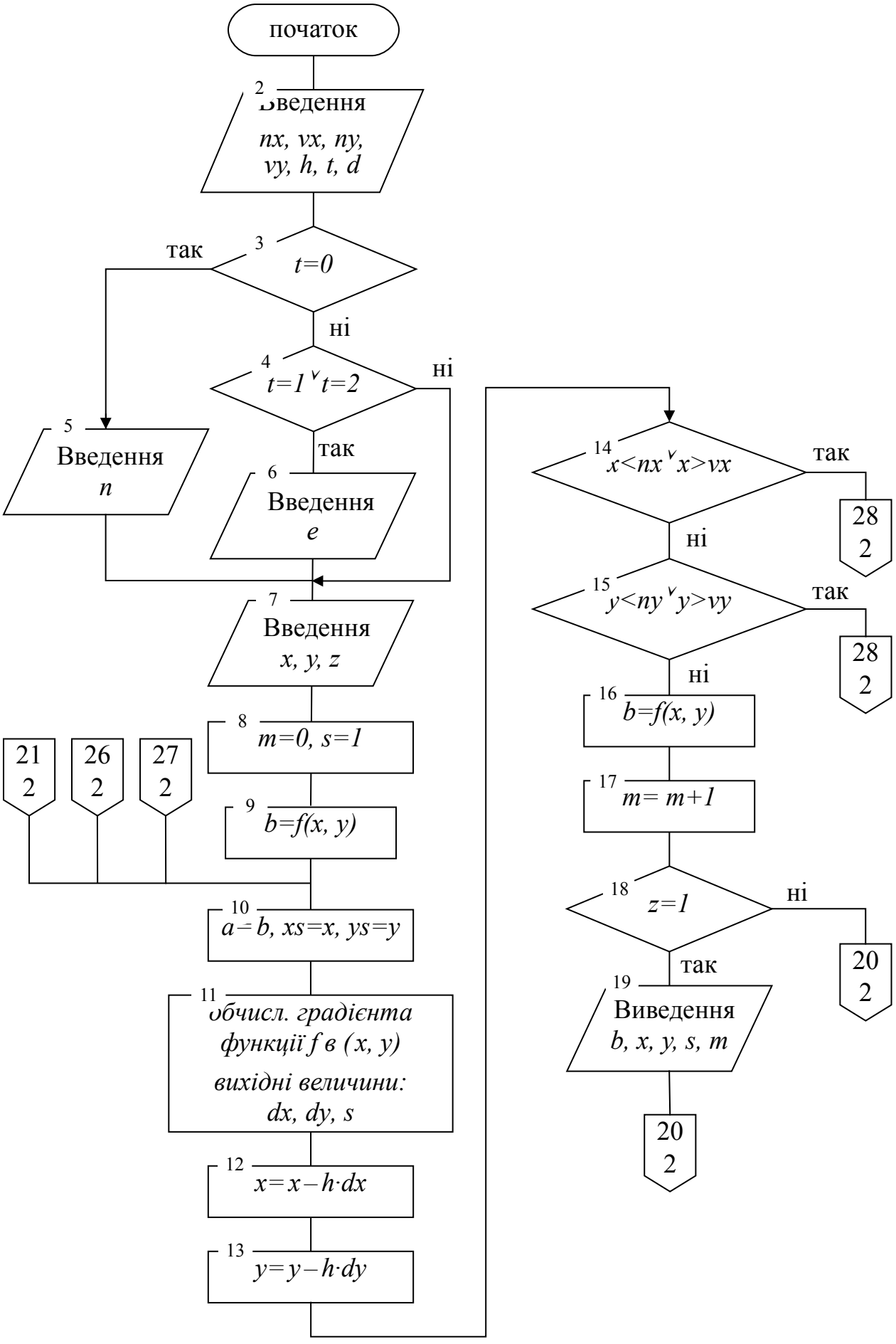


Рисунок 7.6 (аркуш 1)

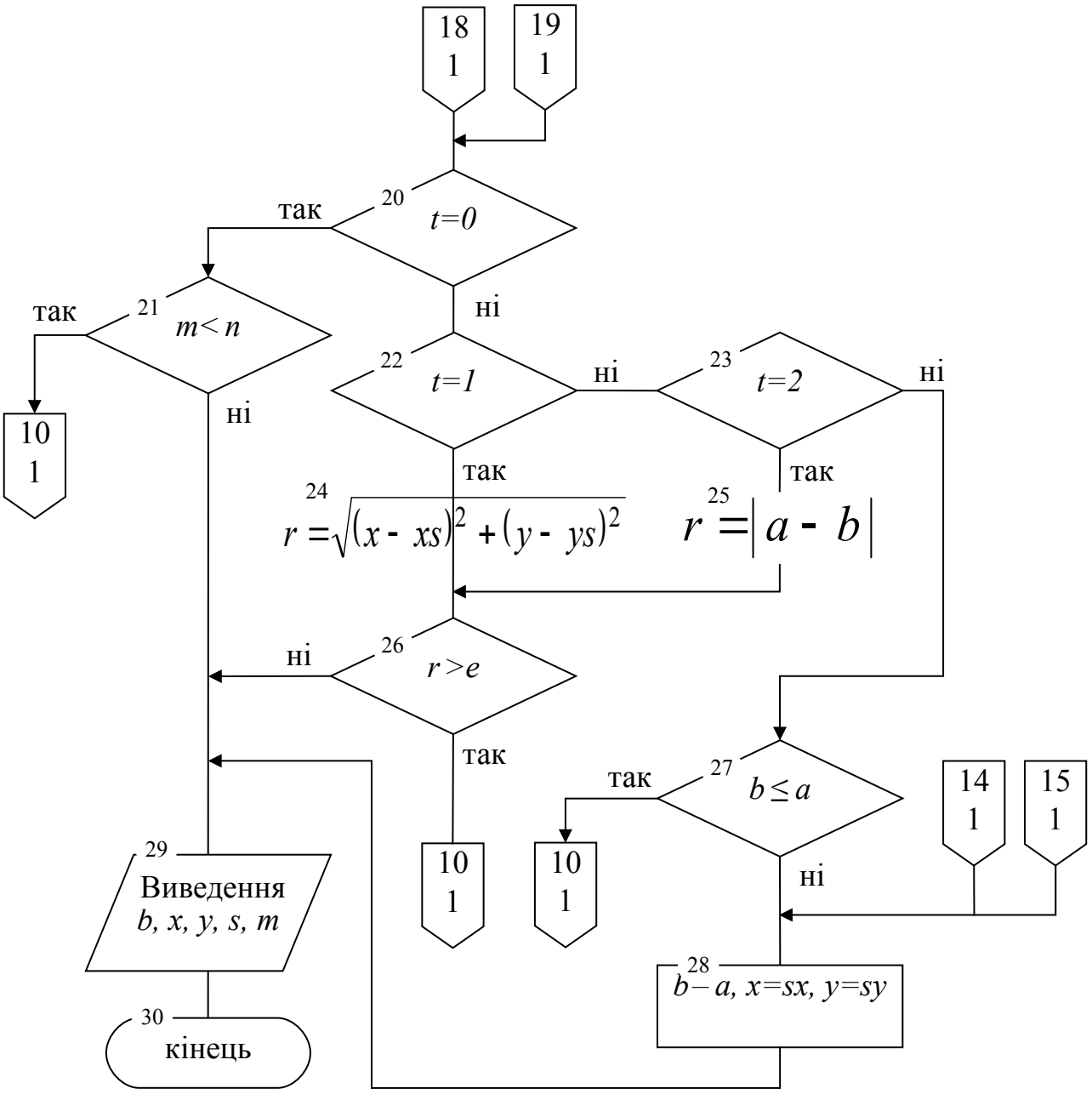


Рисунок 7.6 (аркуш 2)

Для зручності дослідження процесу пошуку тут, також само як і для випадку пошуку за методом покоординатного спуску, у схему алгоритму уведені дії зі змінними-лічильниками, які дозволяють визначити кількість обчислених значень функції (операції зі змінною *s*) і число ітерацій по уточненню мінімуму (операції зі змінною *m*), що мали місце в конкретному випадку пошуку. Уведені в схему алгоритму й змінні-перемикачі, змінюючи значення яких можна управляти процесом обчислень:

зміна значення змінної *t* приводить до зміни критерію зупинки алгоритму (див. операції в блоках 3 – 6 і 20 – 28), зміна значення змінної *z* перемикає режими виводу даних (див. операції в блоках 7, 18, 19).

Блок 11 визначає собою операцію виклику до виконання відокремленої процедури обчислення градієнта функції в заданій точці області зміни аргументів. Схема алгоритму цієї процедури подана на рисунку 7.7. Програмно вона оформлена як підпрограма типу *Sub* і має ім'я *gradvtoch*.

У програмній реалізації цього процесу, також само як і в програмі мінімізації функції за методом покоординатного спуску, для обчислення значень досліджуваної функції використовується підпрограма типу *Function*. Її ім'я, завдання й особливості застосування такі ж, як і при програмуванні попереднього варіанта процесу пошуку.

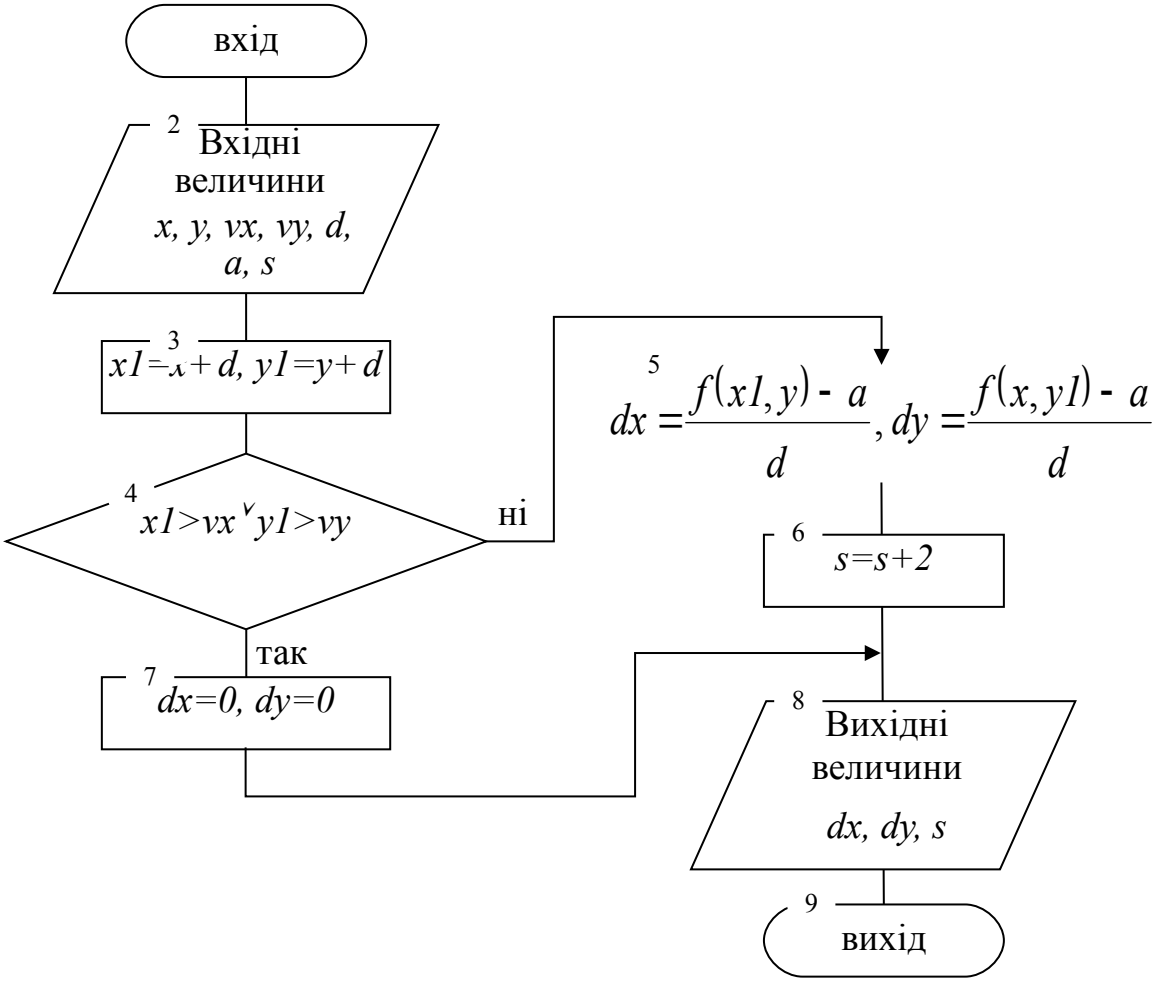


Рисунок 7.7

Basic-програма, яка реалізує алгоритм, що наведений на рисунку. 7.6

```
'Поиск минимума функции методом градиентного спуска
CLS
INPUT "Нижний и верхний пределы изменения координаты
                                    ↳ X функции:", nx, vx
INPUT "Нижний и верхний пределы изменения координаты
                                    ↳ Y функции:", ny, vy
INPUT "Укажите множитель шаг исследования функции:",
                                                    ↳ h
INPUT "Укажите величину приращения аргумента:", d
PRINT "Укажите критерий остановки поиска минимума:"
PRINT "0 - по количеству итераций"
PRINT "1 - по близости точек аргумента значений
                    ↳ функции между итерациями"
PRINT "2 - по близости значений функции между
                                    ↳ итерациями"
PRINT "3 - по началу возрастания текущего значения
                    ↳ функции между итерациями"
INPUT "Номер критерия:", t
IF t = 0 THEN
INPUT "Введите количество допустимых итераций:", n
ELSEIF t = 1 OR t = 2 THEN
INPUT "Введите точность поиска минимума:", e
END IF
INPUT "Введите координаты начальной точки поиска -
                                    ↳ X,Y:", x, y
INPUT "Выводить ли промежуточные результаты? (0-нет,
                                    ↳ 1-да) ", z
m = 0
b# = DvePerFun#(x, y)
s = 1
vozvrat: a# = b#
xs = x: ys = y
CALL gradvtoch(a#, s, x, y, vx, vy, d, dx#, dy#)
x = x - h * dx#
y = y - h * dy#
IF x < nx OR x > vx THEN GOTO vivod
IF y < ny OR y > vy THEN GOTO vivod
b# = DvePerFun#(x, y)
m = m + 1
IF z = 1 THEN
PRINT "x = "; x; " y = "; y; " текущий min функции =
                                                    ↳ "; b#
```

```
PRINT "количество вычисленных значений функции = "; s
PRINT "Конец "; m; "-й итерации"
INPUT "Для продолжения нажмите Enter", zz
END IF
IF t = 0 THEN
IF m < n THEN GOTO vozvrat
ELSEIF t = 1 THEN
r = SQR((x - xs) ^ 2 + (y - ys) ^ 2)
IF r > e THEN GOTO vozvrat
ELSEIF t = 2 THEN
r# = ABS(a# - b#)
IF r# > e THEN GOTO vozvrat
ELSE
IF b# <= a# THEN GOTO vozvrat
vivod: b# = a#: x = xs: y = ys
END IF
PRINT "Результат поиска:"
PRINT "     минимальное значение функции = "; b#
PRINT "     при x = "; x; "  и  y = "; y
PRINT "          количество вычисленных значений
                                ↳ функции = "; s
PRINT "          количество итераций = "; m
END
```

Basic-програма, яка реалізує алгоритм,
що наведений на рисунку. 7.7

```
SUB gradvtoch (a#, s, x, y, vx, vy, d, dx#, dy#)
x1 = x + d: y1 = y + d
IF x1 > vx OR y1 > vy THEN
dx# = 0: dy# = 0
GOTO konec
END IF
dx# = (DvePerFun#(x1, y) - a#) / d
dy# = (DvePerFun#(x, y1) - a#) / d
s = s + 2
konec: END SUB
```

Лабораторна робота № 8

РОЗВ'ЯЗАННЯ ЗАДАЧІ ЛІНІЙНОГО ПРОГРАМУВАННЯ СИМПЛЕКСНИМ МЕТОДОМ

Задачами лінійного програмування називаються оптимізаційні задачі, у яких обмеження зображуються у вигляді рівностей або нерівностей і цільова функція лінійна. Методи лінійного програмування (ЛП) широко використовуються для розв'язання промислових, економічних і організаційних задач. Головними причинами такого широкого застосування методів ЛП є доступність математичного забезпечення для розв'язання задач ЛП великої розмірності й можливість аналізу розв'язання задач ЛП при варіації вихідних даних.

Задача ЛП у стандартній (канонічній) формі із m обмеженнями й n змінними має такий вигляд:

максимізувати функцію

$$F = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \qquad (8.1)$$

при системі обмежень

$$\begin{cases} a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n = b_1, \\ a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n = b_2, \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n = b_m, \end{cases} \qquad (8.2)$$

та умові

$$\begin{array}{llll} x_1 \geq 0, & x_2 \geq 0, & \cdots & x_n \geq 0, \\ b_1 \geq 0, & b_2 \geq 0, & \cdots & b_m \geq 0. \end{array} \qquad (8.3)$$

При розв'язанні задачі ЛП симплекс-методом потрібно, щоб задача була зображена в стандартній формі. Тому, як правило, першим етапом розв'язання задачі ЛП є приведення її до стандартної форми. Для цього застосовуються такі прийоми.

Обмеження у вигляді нерівностей можна перетворити в рівності за допомогою введення так званих залишкових або надлишкових змінних. Наприклад, нерівність вигляду

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \le b_i \qquad (8.4)$$

можна перетворити в рівність за допомогою введення залишкової змінної x_j :

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n + x_j = b_i .$$

Змінна x_j не є від'ємною і відповідає різниці правої та лівої частин нерівності (8.4). Аналогічно нерівність вигляду

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \ge b_i$$

можна перетворити в рівність шляхом введення надлишкової змінної x_t :

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n - x_t = b_i .$$

Якщо ж метою задачі ЛП виступає мінімізація функції F, то для переходу до задачі пошуку максимуму функції вихідну функцію F достатньо помножити на -1.

Зараз ми не будемо розглядати теоретичну базу симплекс-методу. Наведемо тільки алгоритм послідовності перетворення числової таблиці (матриці), значення елементів якої на першому кроці процесу фіксують умови конкретної задачі лінійного програмування, а наприкінці останнього — містять результати розв'язання задачі. У пошуку докладних пояснень цього алгоритму варто звернутися до відповідної літератури [6, 18].

Нехай задача лінійного програмування подана у стандартній формі. Треба максимізувати цільову функцію F з n невідомими, яка описується формулою (8.1), при наявності системи із m обмежень (8.2) і умови невід'ємності невідомих та вільних членів (8.3).

Крок 1 Оголошується й заповнюється даними умови задачі таблиця (матриця) S, що містить $m+1$ рядок і $n+2$ стовпці (рисунок 8.1).

S	*0*	*1*	*2*	…	*n*	*n+1*
0		c_1	c_2	…	c_n	F
1	*n-m+1*	a_{11}	a_{12}	…	a_{1n}	b_1
2	*n-m+2*	a_{21}	a_{22}	…	a_{2n}	b_2

⋮	⋮	…	⋮	…	⋮	⋮
m	n	a_{m1}	a_{m2}	…	a_{mn}	b_m

Рисунок 8.1

У нульовий рядок таблиці S, починаючи з елемента s_{01}, вносяться послідовно коефіцієнти при аргументах функції F: $c_1, c_2, \ldots, c_n, \ldots$ Останній елемент цього рядка $s_{0\ m+1}$ одержує нульове значення. Далі після кожного чергового перетворення таблиці S йому буде надаватися значення функції F при новому знайденому наборі значень її аргументів. Підматриця (мінор) $\{ s_{ij}, i = \overline{1, m}, j = \overline{1, n} \}$ заповнюється значеннями коефіцієнтів системи обмежень (8.2), при цьому присутня повна погодженість у індексах, тобто елемент s_{11} отримує значення коефіцієнта a_{11}, елемент s_{12} отримує значення a_{12} і т.д. … , елемент s_{mn} отримує значення a_{mn}. У нульовий стовпець матриці S заносяться індекси базисних змінних. У стовпці $n+1$, починаючи з рядка 1, містяться постійні із системи обмежень: елемент s_{1n+1} одержує значення b_1, елемент s_{2n+1} одержує значення b_2 і т.д. …, елемент s_{mn+1} одержує значення b_m.

Крок 2 У нульовому рядку таблиці S серед елементів $\{ s_{0j}, j = \overline{1, n} \}$ шукається елемент із максимальним додатним значенням. Якщо серед зазначених елементів немає додатних, то подальша оптимізація неможлива. Поточний базисний припустимий план вважається оптимальним, і процес переходить до виконання кроку 5. Якщо ж елемент з максимальним додатним значенням існує, то фіксується індекс стовпця матриці S, де він перебуває. Позначимо цей індекс через k. Стовпець із таким індексом називають ключовим.

Крок 3 Значення кожного елемента стовпця $n+1$, починаючи з елемента, що належить першому рядку, ділиться на значення відповідного за індексом рядка елемента із ключового стовпця, тобто одержуємо набір дробів: s_{1n+1} / s_{1k}, s_{2n+1} / s_{2k}, s_{3n+1} / s_{3k}, …, s_{mn+1} / s_{mk} ... У ньому шукається найменше додатне число. Якщо додатних чисел у наборі не виявилося, то задача знаходження оптимуму не має розв'язку, алгоритм завершується. Якщо таке число знайдене, то індекс рядка матриці S, елементи якої дали

ділене й дільник для його обчислення, фіксується. Позначимо цей індекс через *r*. Рядок, що визначений таким способом, прийнято називати провідним. Якщо в наборі дробів виявилося кілька однакових додатних найменших чисел, тобто кілька рядків матриці *S* можна вибрати як провідні, то вибирається провідним рядком з них той, індекс якого найбільший.

Елемент s_{rk} матриці *S* називають генеральним (ще — провідним, або розв'язуючим) елементом перетворення плану.

Крок 4 Здійснюється перерахунок таблиці *S* за допомогою виконання ланцюга таких дій:

4.1 Значення всіх елементів підматриці $\{ s_{ij}, i = \overline{0, m}, j = \overline{1, n+1} \}$, крім значення останнього елемента нульового рядка й тих, які належать елементам провідного рядка, змінюються відповідно до формули:

$$s_{ij}^{*} = s_{ij} - \frac{s_{rj} \cdot s_{ik}}{s_{rk}},$$

де s_{ij} — попереднє значення елемента під матриці; s_{ij}^{*} — перераховане значення того ж елемента; *k* — індекс ключового стовпця; *r* — індекс провідного рядка; s_{rk} — значення генерального елемента.

4.2 Значення всіх елементів провідного рядка, крім значення елемента нульового стовпця, перераховуються відповідно до формули: $s_{rj}^{*} = s_{rj} / s_{rk}$, де s_{rj} — попереднє значення *j*-го елемента провідного рядка, s_{rj}^{*} — перераховане значення того ж елемента.

4.3 Елемент s_{r0} отримує значення індексу *k*.

4.4 Обчислюється значення функції *F* відповідно до формули (8.1). Результат обчислення заноситься в елемент $s_{0\,n+1}$. Значення невідомих, які шукаються, вибираються із елементів стовпця *n+1* у діапазоні рядків від першого до *m*-го. Індекси невідомих, яким належать ці значення, визначаються за індексами, що зберігаються у нульовому стовпці таблиці *S*.

Наведена послідовність процедур цього кроку дозволяє вивести із припустимого базисного розв'язання змінну з індексом *r* і ввести замість неї в базис змінну з індексом *k*. Далі після

виконання дій кроку 4 повинно відбутися повернення до виконання кроку 2.

Крок 5 Оголошується оптимальне припустиме базисне розв'язання. Індекси й значення змінних, що входять до оптимального плану, читаються з нульового й *n+1*-го стовпців матриці *S* відповідно в діапазоні між першим й *m*-м рядками. Оптимальне значення функції *F* читається з останнього елемента нульового рядка матриці *S*.

Укрупнена схема цього алгоритму зображена у відповідному розділі [18]. Нижче, на рисунку 8.2, наводиться розширена схема алгоритму розв'язання задачі ЛП із застосуванням симплекс-методу, програмна реалізація якого відповідає цілям лабораторної роботи. Відповідність між блоками схеми з [18] і блоками схеми рисунка 8.2 можна визначити так: блоку “Заповнення таблиці” у схемі із [18] відповідають блоки 2 – 17 у схемі рисунка 8.2; блокам “Базисний припустимий план (БПП) оптимальний?” і “Вибір ключового стовпця” відповідають дії, що задані блоками 19 – 25 у схемі рисунка 8.2; блоку “ Чи є рішення?” відповідають блоки 26 – 30; блоку “Вибір провідного рядка” відповідають блоки 32 – 39; “Перерахунок таблиці” здійснюється в блоках 42 – 58; для виводу повідомлення про відсутність рішення задачі служить блок 31; вивід проміжних і кінцевих результатів задається блоками 60, 62 – 67.

Дії, що визначені іншими блоками розширеної схеми алгоритму з рисунка 8.2, служать для управління процесом рішення задачі та для фіксації його проміжних обчислювальних кроків.

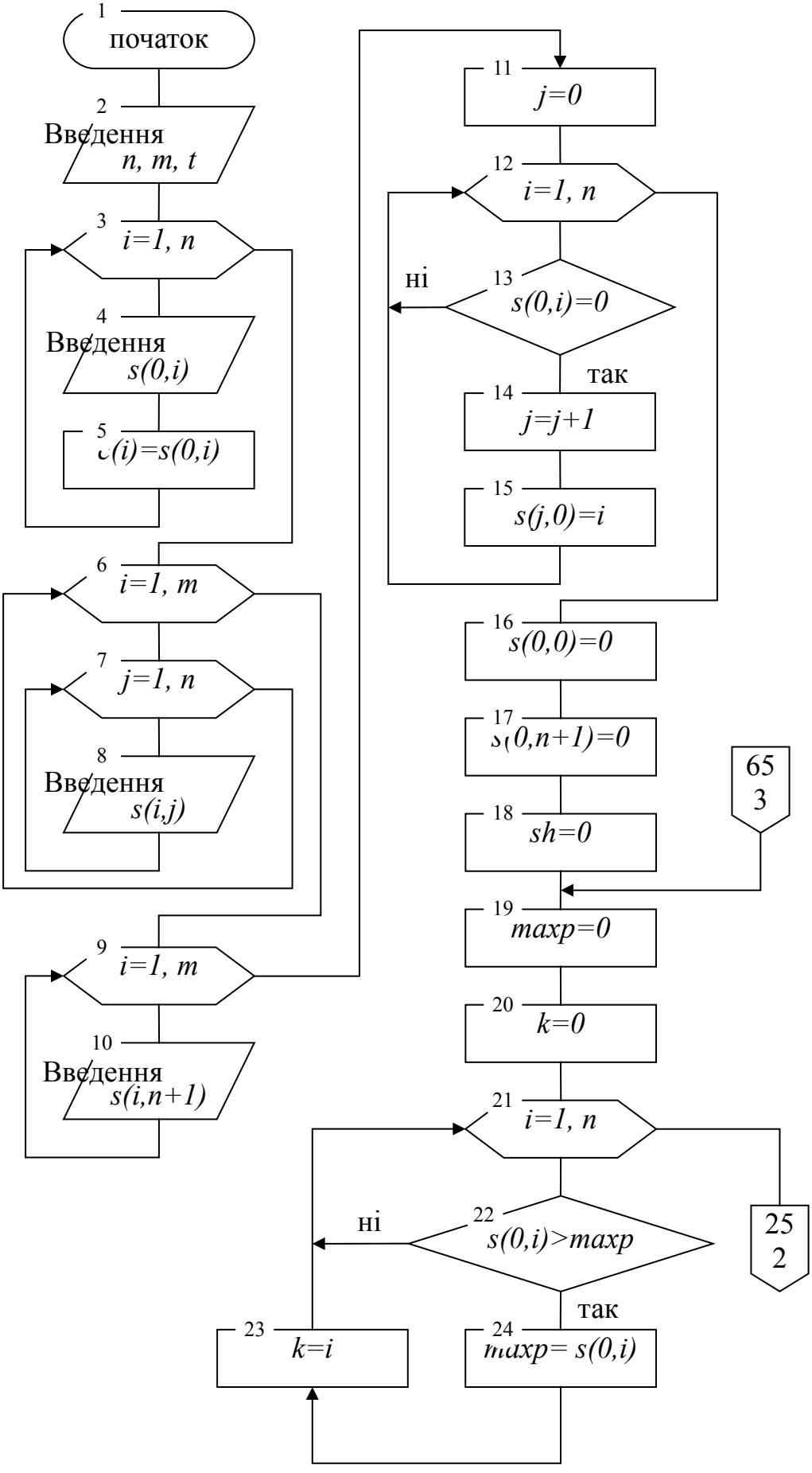


Рисунок 8.2 (аркуш 1)

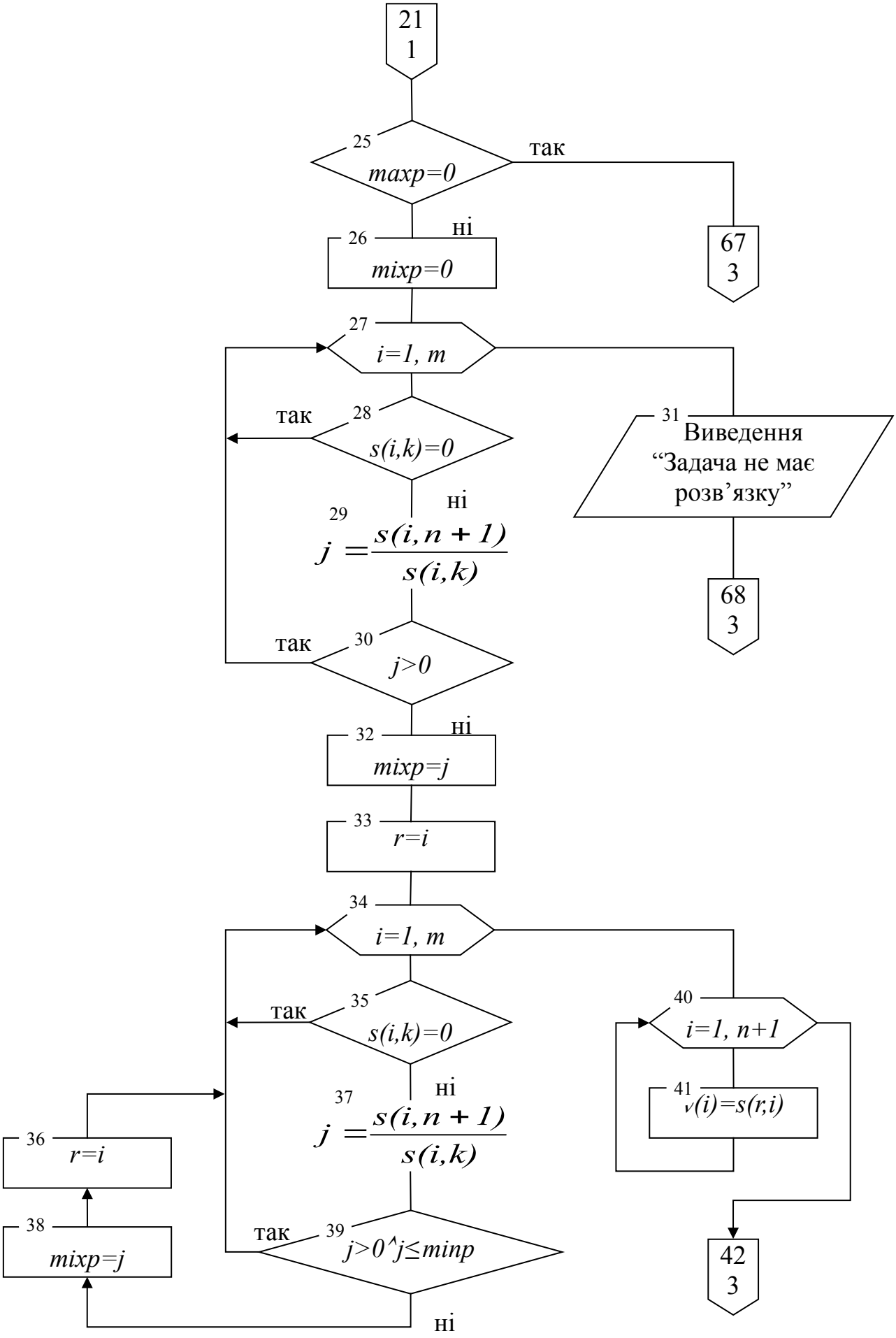


Рисунок 8.2 (аркуш 2)

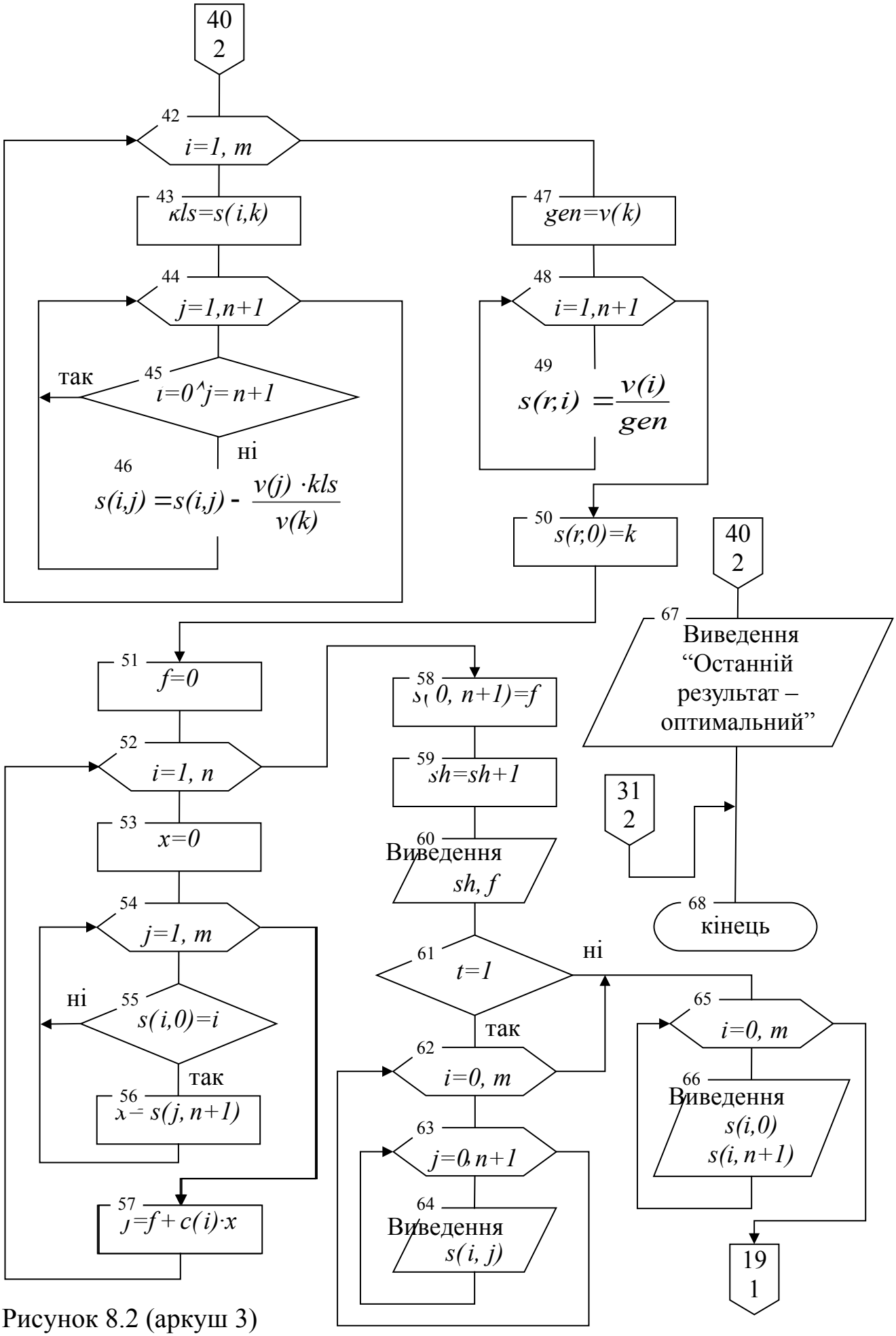


Рисунок 8.2 (аркуш 3)

Basic-програма алгоритму на рисунку 8.2

```
' Решение задач линейного программирования симплекс-
' методом
CLS
INPUT "Введите количество переменных(n): ", n
INPUT "Введите количество ограничений(m): ", m
DIM s(0 TO m, 0 TO n + 1), v(1 TO n + 1), c(1 TO n)
PRINT "Введите для целевой функции значение -"
FOR i = 1 TO n
PRINT i; "-го коэффициента: ";
INPUT s(0, i)
c(i) = s(0, i)
NEXT i
PRINT "Введите матрицу коэффициентов ограничений -"
FOR i = 1 TO m
FOR j = 1 TO n
PRINT "a("; i; ", "; j; ") = ";
INPUT s(i, j)
NEXT j, i
PRINT "Введите вектор свободных членов ограничений -"
FOR i = 1 TO m
PRINT "b("; i; ") = ";
INPUT s(i, n + 1)
NEXT i
t = 0
PRINT "Надо ли выводить преобразованную матрицу ";
INPUT "(да - 1, нет -0) ? ", t
PRINT
j = 0
FOR i = 1 TO n
IF s(0, i) = 0 THEN
j = j + 1
s(j, 0) = i
END IF
NEXT i
s(0, 0) = 0: s(0, n + 1) = 0: sh = 0
vozvrat: maxp = 0
k = 0
FOR i = 1 TO n
IF s(0, i) > maxp THEN
maxp = s(0, i)
k = i
END IF
NEXT i
```

```
IF maxp = 0 THEN GOTO yesreshenie
minp = 0
FOR i = 1 TO m
IF s(i, k) = 0 THEN GOTO prodcikl
j = s(i, n + 1) / s(i, k)
IF j > 0 THEN
minp = j
r = i
GOTO dalee
END IF
prodcikl: NEXT i
PRINT
PRINT "Задача не имеет решения !!!"
PRINT
GOTO konec
dalee: FOR i = 1 TO m
IF s(i, k) = 0 THEN GOTO prodcikl1
j = s(i, n + 1) / s(i, k)
IF j > 0 AND j <= minp THEN
minp = j
r = i
END IF
prodcikl1: NEXT i
FOR i = 1 TO n + 1
v(i) = s(r, i)
NEXT i
FOR i = 0 TO m
kls = s(i, k)
FOR j = 1 TO n + 1
'IF s(i, j) = 0 THEN GOTO dalee1
IF i = 0 AND j = n + 1 THEN GOTO dalee1
s(i, j) = s(i, j) - v(j) * kls / v(k)
dalee1: NEXT j
NEXT i
gen = v(k)
FOR i = 1 TO n + 1
s(r, i) = v(i) / gen
NEXT i
s(r, 0) = k
f = 0
FOR i = 1 TO n
x = 0
FOR j = 1 TO m
IF s(j, 0) = i THEN x = s(j, n + 1)
NEXT j
```

```
f = f + c(i) * x
NEXT i
s(0, n + 1) = f
sh = sh + 1
PRINT "Шаг N"; sh
PRINT "Значение функции F = "; s(0, n + 1)
IF t = 0 THEN GOTO perestup
PRINT "Вывод основной матрицы:"
FOR i = 0 TO m
FOR j = 0 TO n + 1
PRINT USING "###.### "; s(i, j);
NEXT j
PRINT
NEXT i
perestup: PRINT "Значения базисных переменных:"
FOR i = 1 TO m
PRINT "X("; s(i, 0); ") = "; s(i, n + 1)
NEXT i
PRINT
INPUT "Для продолжения нажмите Enter", sss : PRINT
GOTO vozvrat
yesreshenie: PRINT "Последний результат - ";
PRINT "оптимальное допустимое решение!"
konec: END
```

Розглянемо на прикладі основні моменти роботи із програмою.

Необхідно знайти $x_1 \geq 0$, $x_2 \geq 0$, при яких функція $F = -7x_1 - 5x_2$ досягає мінімуму та виконуються обмеження

$$\begin{cases} 2x_1 + 3x_2 \leq 19, \\ 2x_1 + x_2 \leq 13, \\ 3x_2 \leq 15, \\ 3x_1 \leq 18. \end{cases} \qquad (8.5)$$

Перш ніж запустити програму на виконання та вводити вхідні дані цієї задачі, треба її умову зобразити у стандартній формі. Для цього система нерівностей (8.5) перетворюється шляхом уведення додаткових змінних у систему рівностей

$$\begin{cases} 2x_1 + 3x_2 + x_3 = 19, \\ 2x_1 + x_2 + x_4 = 13, \\ 3x_2 + x_5 = 15, \\ 3x_1 + x_6 = 18. \end{cases} \qquad (8.6)$$

Крім того, задача мінімізації функції F заміняється задачею максимізації: функція F помножується на -1.

Після початку роботи програми послідовно пропонується увести кількість змінних (для цього прикладу — 6), кількість обмежень (для цього прикладу — 4). Далі потрібно вводити значення коефіцієнтів при змінних функції, яку треба максимізувати. Для даного прикладу цю функцію з усіма її коефіцієнтами (тобто коефіцієнти із нульовими значеннями теж ураховуються) можна записати так:

$$F = 7x_1 + 5x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 .$$

Завершується завдання вхідних даних уведенням значень коефіцієнтів системи обмежень (8.6):

$$\begin{cases} 2x_1 + 3x_2 + 1x_3 + 0x_4 + 0x_5 + 0x_6 = 19, \\ 2x_1 + 1x_2 + 0x_3 + 1x_4 + 0x_5 + 0x_6 = 13, \\ 0x_1 + 3x_2 + 0x_3 + 0x_4 + 1x_5 + 0x_6 = 15, \\ 3x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 1x_6 = 18. \end{cases}$$

Коефіцієнти вводяться по рядках без значень правих частин рівностей. Стовпець останніх уводиться окремо після значень всіх коефіцієнтів матриці системи.

Після введення вхідних даних програма користувачеві ставить запитання: чи треба після знаходження кожного припустимого плану виводити значення розрахункової матриці S (див. рисунок 8.1). Це виведення можна заборонити або дозволити.

Якщо користувач відмовився від виведення значень матриці S, то результати обчислень будуть мати такий вигляд:

```
Шаг N 1
```

```
Значение функции F =  42
Значения базисных переменных:
X( 3 ) =  7
X( 4 ) =  1
X( 5 ) =  15
X( 1 ) =  6

Шаг N 2
Значение функции F =  47
Значения базисных переменных:
X( 3 ) =  4
X( 2 ) =  1
X( 5 ) =  12
X( 1 ) =  6

Шаг N 3
Значение функции F =  50
Значения базисных переменных:
X( 6 ) =  3
X( 2 ) =  3
X( 5 ) =  6
X( 1 ) =  5
```

Ці результати показують, що для розглянутого прикладу за три ітерації алгоритму був знайдений оптимальний припустимий план: $x_1 = 5,\ x_2 = 3,\ x_3 = 0,\ x_4 = 0,\ x_5 = 6,\ x_6 = 3.$ Мінімальне значення функції F, що досягається при цьому плані, є –50.

Лабораторна робота № 9

МЕТОД МОНТЕ-КАРЛО В ЗАДАЧАХ ОПТИМІЗАЦІЇ

Способи розв’язання задач, що використовують випадкові величини, дістали загальну назву методу Монте-Карло. Більш точно під методом Монте-Карло розуміється сукупність прийомів, що дозволяють отримувати розв’язання математичних, фізичних, технічних і інших задач за допомогою випадкових випробувань [18].

У задачах оптимізації, наприклад, треба мінімізувати деяку функцію, що описує залежність деякого параметра моделі від інших. На практиці такі функції є, як правило, багатовимірними, але в даній роботі будуть розглядатися лише одновимірний і двовимірний випадки.

Нехай є функція $F(x)$. Треба знайти

$$x^* = \arg\operatorname*{Min}_{x \in D} F(x),$$

де D — припустима область зміни значення x.

Метод випадкового пошуку в загальному випадку складається з таких дій. В області пошуку величини x^* відповідно до заданої щільності розподілу генеруються випадкові точки $x_1, x_2, x_3, \ldots$ При цьому для кожного значення x_i обчислюється значення функції $F(x)$ із запам’ятовуванням мінімального.

Алгоритм пошуку записується у вигляді рекурентної формули для x і $F(x)$:

$$x_i^* = \begin{cases} x_i, & \text{якщо } F(x_i) < F(x_{i-1}), \\ x_{i-1}, & \text{якщо } F(x_i) \geq F(x_{i-1}), \end{cases} \qquad (9.1)$$

де x_i^* — аргумент мінімального значення функції $F(x)$, який знайдено за i реалізацій її обчислення від випадково обраних значень із області D. Якщо розглядається одновимірний варіант задачі, то D, звичайно, є деякий числовий відрізок $[a, b]$, $a < b$.

Схема алгоритму пошуку мінімуму у випадку одновимірної функції F показана на рисунку 9.1.

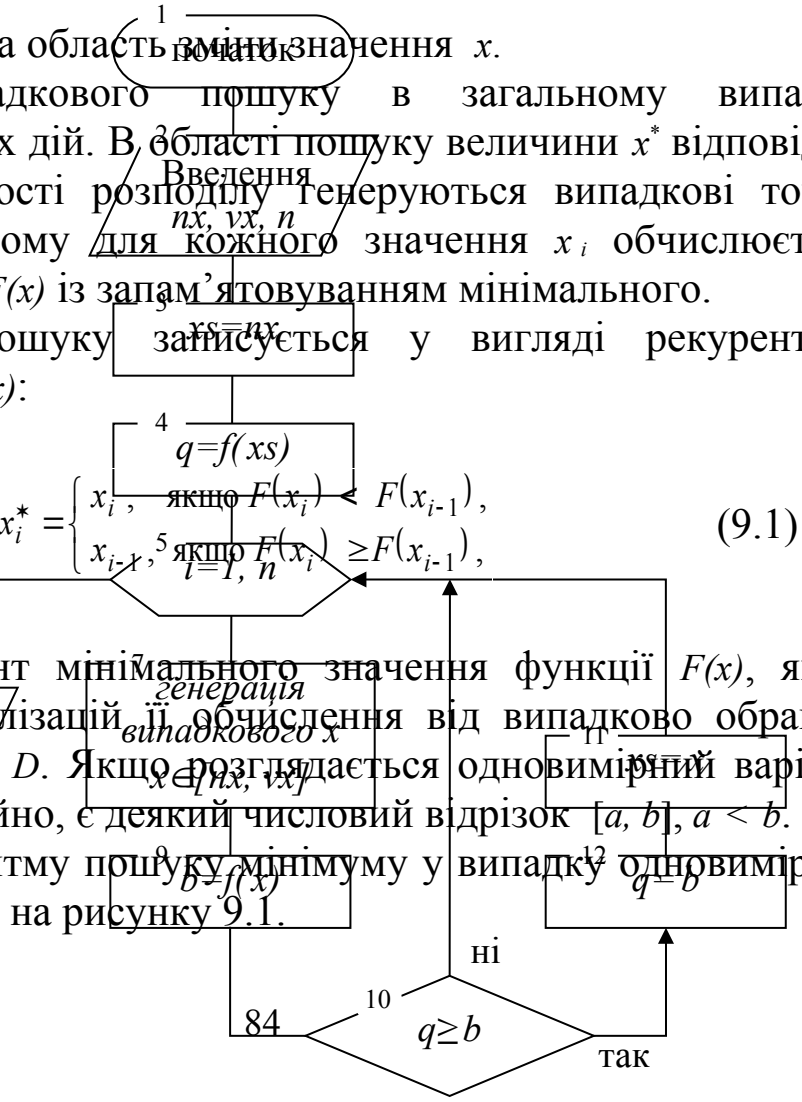


Рисунок 9.1

Оскільки кожна операція вибору значення x з D має випадковий характер і ніяк не залежить від інших операцій, то, очевидно, слідом за значенням аргументу x можна також визначити значення аргументу y, якщо функція F залежала б від двох аргументів. А потім таким же способом можна було б знайти й значення аргументу z, якби функція F залежала від трьох аргументів, і т.д. Статистична незалежність цих операцій дозволяє легко задавати процес пошуку мінімуму функції F, що визначена в гіперпаралелепіпеді D із n-вимірного простору. Причому описуватися процес пошуку буде тією ж формулою (9.1), звісно, у ній під x_i, x_{i-1}, x_i^* вже слід розуміти n-вимірні вектори.

На рисунку 9.2 наводиться схема алгоритму пошуку мінімуму у випадку двовимірної функції $F(x, y)$ в області $D = \{ a_i \le x \le b_i, i = 1, 2 \}$.

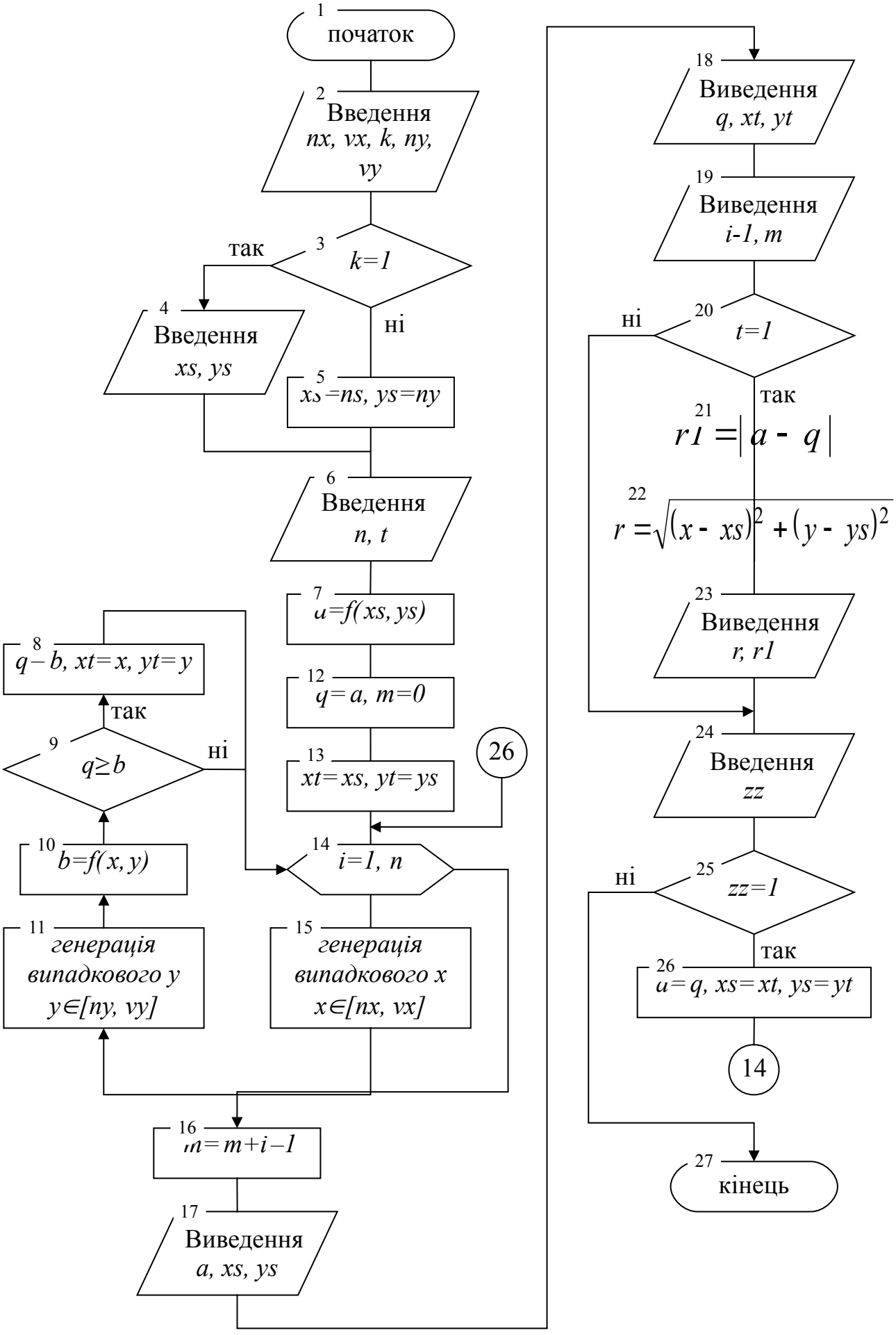

1
початок
2
Введення
nx, vx, k, ny, vy
3
k=1
так
ні
4
Введення
xs, ys
5
xs=ns, ys=ny
6
Введення
n, t
7
a=f(xs, ys)
8
q=b, xt=x, yt=y
так
9
q≥b
ні
10
b=f(x, y)
11
генерація випадкового y
y∈[ny, vy]
12
q=a, m=0
13
xt=xs, yt=ys
26
14
i=1, n
15
генерація випадкового x
x∈[nx, vx]
16
m=m+i–1
17
Виведення
a, xs, ys
18
Виведення
q, xt, yt
19
Виведення
i-1, m
20
t=1
ні
так
21
r1 = | a - q |
22
r = √((x - xs)² + (y - ys)²)
23
Виведення
r, r1
24
Введення
zz
25
zz=1
ні
так
26
a=q, xs=xt, ys=yt
14
27
кінець

Рисунок 9.2

Якщо не враховувати блоки, де обчислю-ються й виводяться поточні значення характеристик процесу, то вона майже цілком повторює схему рисунка 9.1. Обидва алгоритми мають частини, у яких визначається початкова точка пошуку й відповідне значення функції (блоки 3, 4 на рисунку 9.1 і блоки 5, 7 на рисунку 9.2). Кожна схема має схожі один на одного фрагменти, де визначені дії з проведення випробування, його аналізу й обліку (блоки 7, 9 – 12 на рисунку 9.1 і блоки 8 – 11, 15 на рисунку 9.2). На обох схемах є блок, що організовує циклічне повторення цих дій (блок 5 на рисунку 9.1 і блок 14 на рисунку 9.2). На обох схемах можна бачити операції з виведення результатів пошуку (блок 6 на рисунку 9.1 і блоки 17, 18, 23 на рисунку 9.2).

Програма, що реалізує мовою Basic схему алгоритму з рисунка 9.2, наведена нижче. Обчислення функції, що досліджується, задається в ній за допомогою оператора-функції (див. рядок 3 від початку тексту програми). Як приклад досліджуваної функції в програмі описується функція $f(x,y) = (1-x)^2 + (1-y)^2$. Щоб задати іншу досліджувану функцію, рекомендується в зазначеному рядку програми праворуч від знака рівності внести необхідні корективи без зміни імені функції та імен формальних параметрів.

Випадкові величини у програмі дістаються за допомогою стандартної функції RND, яка може утворювати послідовність псевдовипадкових чисел із діапазону між 0 та 1 з розподілом, близьким до рівномірного. Щоб із новим запуском програми така числова послідовність не повторювалася, використовується оператор RANDOMIZE, який може змінювати стартові константи, чим змушує вказану послідовність будуватися з іншого відліку.

Basic-програма, яка реалізує алгоритм із рисунка 9.2 – пошук мінімуму функції від двох змінних за методом Монте-Карло

```
'Поиск минимума функции от двух переменных
'методом Монте-Карло
DEF fnFun (x, y) = (1 - x) ^ 2 + (1 - y) ^ 2
CLS
RANDOMIZE TIMER
PRINT "Нижний и верхний пределы изменения ";
```

```
INPUT "координаты X функции:", nx, vx
PRINT "Нижний и верхний пределы изменения ";
INPUT "координаты Y функции:", ny, vy
PRINT "Надо ли задавать начальную точку поиска? ";
INPUT "( да - 1, нет - 0): ", k
IF k = 1 THEN
PRINT "Введите координаты начальной точки ";
INPUT "поиска - X,Y: ", xs, ys
ELSE
xs = nx: ys = ny
END IF
INPUT "Введите количество независимых испытаний:", n
PRINT "Сопоставлять ли результат с начальными ";
INPUT "данными?( да - 1, нет - 0): ", t
a = fnFun(xs, ys)
q = a: xt = xs: yt = ys: m = 0
vozvrat: PRINT
FOR i = 1 TO n
x = nx + (vx - nx) * RND
y = ny + (vy - ny) * RND
b = fnFun(x, y)
IF q >= b THEN
q = b: xt = x: yt = y
END IF
NEXT i
m = m + i - 1
PRINT
PRINT
PRINT "*** ОТЧЕТ о ПОИСКЕ МИНИМУМА ФУНКЦИИ ***"
PRINT "Предыдущий результат поиска:"
PRINT "     минимальное значение функции = "; a
PRINT "     при x = "; xs; "  и  y = "; ys
PRINT
PRINT "Последний результат поиска:"
PRINT "     минимальное значение функции = "; q
PRINT "     при x = "; xt; "  и  y = "; yt
PRINT "     количество испытаний = "; i - 1
PRINT "Общее количество испытаний с момента ";
PRINT "запуска = "; m
PRINT
IF t = 1 THEN
r = SQR((xt - xs) ^ 2 + (yt - ys) ^ 2)
r1 = ABS(a - q)
PRINT "Уменьшение значения минимума = "; r1
PRINT "Расстояние между точками минимумов ";
```

```
PRINT "(<<старого>> и <<нового>>) = "; r
END IF
PRINT
INPUT "Продолжить поиск? ( да - 1, нет - 0): ", zz
IF zz = 1 THEN
a = q: xs = xt: ys = yt
GOTO vozvrat
END IF
END
```

Алгоритм розв'язання задачі одновимірної оптимізації за методом Монте-Карло можна також використати для випадку багатовимірної оптимізації, узявши за його основу ітераційні кроки в алгоритмі пошуку мінімуму функції за методом покоординатного спуску (див. опис до лабораторної роботи № 7). Нижче наведена програмна реалізація такого алгоритму мовою Basic, що дозволяє шукати мінімум у функції від двох змінних. Вона складається з головної програми й двох підпрограм. Схеми їхніх алгоритмів тут не наводяться, оскільки алгоритм головної програми відрізняється від алгоритму, що показаний схемою на рисунку 6.2, лише незначними деталями, а алгоритм *Sub*-підпрограми такий же, як показаний схемі на рисунку 9.1. Текст *Function*-підпрограми *DvePerFun*, яка задає обчислення функції, що досліджується, в заданій точці, тут не наводиться також. Як складати таку підпрограму, описано у лабораторній роботі № 7.

Basic-програма, яка реалізує алгоритм пошуку мінімуму функції від двох змінних при об'єднанні методів покоординатного спуску та Монте-Карло

```
'Поиск минимума функции путем покоординатного спуска
'с использованием метода Монте-Карло
CLS
RANDOMIZE TIMER
PRINT "Нижний и верхний пределы изменения ";
INPUT "координаты X функции:", nx, vx
PRINT "Нижний и верхний пределы изменения ";
INPUT "координаты Y функции:", ny, vy
PRINT "Надо ли задавать начальную точку поиска?
INPUT "( да - 1, нет - 0): ", k
IF k = 1 THEN
PRINT ""Введите координаты начальной точки ";
```

```
INPUT "поиска - X,Y:", x, y
ELSE
x = nx: y = ny
END IF
PRINT "Укажите количество вычислений функции за шаг";
INPUT " в итерации:", h
PRINT "Укажите критерий остановки поиска минимума:"
PRINT "0 - по количеству итераций"
PRINT "1 - по близости точек аргумента минимума";
PRINT " функции между итерациями"
PRINT "2 - по близости значений минимума функции";
PRINT " между итерациями"
INPUT "Номер критерия:", t
IF t = 0 THEN
INPUT "Введите количество допустимых итераций:", n
ELSE
INPUT "Введите точность поиска минимума:", e
END IF
PRINT "Выводить ли результаты после каждого шага ";
INPUT "итерации?( да - 1, нет - 0): ", z
q = DvePerFun(x, y)
vozvrat1: m = 0
l = 0
vozvrat: a = q: xs = x: ys = y
IF l = 0 THEN
CALL minpokoor(1, x, y, nx, vx, h, a, q)
IF z = 1 THEN
PRINT "x = "; x; " y = "; y;
PRINT " текущий min функции = "; q
END IF
l = 1
GOTO vozvrat
ELSE
CALL minpokoor(2, x, y, ny, vy, h, a, q)
l = 0
m = m + 1
IF z = 1 THEN
PRINT "x = "; x; " y = "; y;
PRINT " текущий min функции = "; q
PRINT "Конец "; m; "-й итерации"
INPUT "Для продолжения нажмите Enter", zz
END IF
IF t = 0 THEN
IF m < n THEN GOTO vozvrat
ELSEIF t = 1 THEN
```

```
r = SQR((x - xs) ^ 2 + (y - ys) ^ 2)
IF r > e THEN GOTO vozvrat
ELSE
r = ABS(a - q)
IF r > e THEN GOTO vozvrat
END IF
END IF
PRINT "Результат поиска:"
PRINT "     минимальное значение функции = "; q
PRINT "     при x = "; x; "  и  y = "; y
PRINT "     количество итераций = "; m
INPUT "Продолжить поиск? ( да - 1, нет - 0): ", zz
IF zz = 1 THEN GOTO vozvrat1
END
```

Basic-підпрограма `minpokoor`, що реалізує алгоритм випадкового пошуку мінімуму функції від двох змінних за однією із них при фіксації значення іншої

```
SUB minpokoor (i, x, y, nk, vk, h, a, q)
q = a: xs = x: ys = y
FOR j = 1 TO h
z = nk + (vk - nk) * RND
IF i = 2 THEN y = z ELSE x = z
b = DvePerFun(x, y)
IF q >= b THEN
q = b: xs = x: ys = y
END IF
NEXT j
x = xs: y = ys
END SUB
```

Лабораторна робота № 10

ВИЗНАЧЕННЯ ШЛЯХУ МІНІМАЛЬНОЇ ВАРТОСТІ МЕТОДОМ ДИНАМІЧНОГО ПРОГРАМУВАННЯ

Залежно від специфіки задачі, виду цільової функції і обмежень існують різні методи розв’язання таких задач, об'єднані під загальною назвою - методи математичного програмування. Одним з них є метод динамічного програмування (ДП).

Цей метод призначений для пошуку оптимальних розв’язків у задачах оптимізації багатокрокових процесів прийняття рішень.

ДП визначає оптимальний розв’язок **n**-вимірної задачі шляхом її декомпозиції на **n** етапів, кожний з яких являє собою підзадачу щодо однієї змінної. Обчислювальна перевага такого підходу полягає в тому, що ми займаємося розв’язанням одно-вимірних оптимізаційних задач замість великої **n**-вимірної задачі.

Процес розв’язання задачі методом **ДП** містить у собі два основних етапи:

1 Оптимізація кроків, при цьому визначаються умовно-оптимальні керування для кожного кроку.

2 Оптимізація процесу в цілому; при цьому з оптимальних кроків складається оптимальний процес шляхом вибору оптимальних керувань із умовно-оптимальних.

Розглянемо метод **ДП** на прикладі розв’язання такої задачі:

Поряд з іншими роботами для забезпечення надійності колісних пар ваконів у колісно-роликовому цеху здіснюїться такі задачі:

- обточування поверхні кочення колісних пар (вид 1)
- дефектоскопія деталей буксового вузла (вид 2)

Необхідно визначити послідовність реалізації заходів, при якій задана надійність буде досягнута з мінімальним витратами.

Заходів 1 - виду – 4; заходів 2 виду – 5.

Оскільки задача двопараметрична, то як модель доцільно використовувати граф, а процедуру оптимізації зобразимо як визначення шляху найменшої вартості з початкової вершини в кінцеву. Цей шлях проходить через ряд проміжних точок.

Нехай задані 20 точок, які в абстрагованому видгляді можна розташувати в такий спосіб (рисунок 10.1).

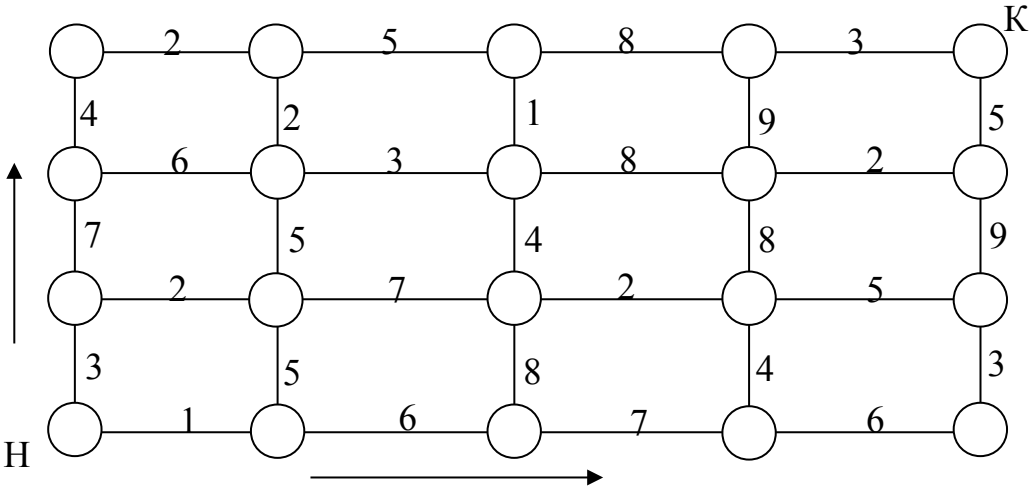


Рисунок 10.1 – Графічне зображення вихідних даних

Маємо прямокутний граф, ребра якого відповідають проведеним заходам: вертикальні - заходам виду 1; горизонтальні - заходам виду 2.

Оскільки точки розташовані у вигляді матриці, то зручно позначити їх номерами рядка й стовпця, у яких вони перебувають (рисунок 10.2).

Відома вартість шляху між сусідніми точками, вона показана у вигляді чисел на лініях, що з'єднують точки. Ліва нижня точка є початковою, а права верхня - кінцевою. Шлях складається із кроків - відрізків ліній, що з'єднують суміжні точки.

Для запису вищевикладених дій у вигляді алгоритму введемо позначення:

Елементи матриці *A* - a(i, j)- вартість горизонтального кроку з вершини з індексами i, j;

Елементи матриці *B* - b(i, j) -вартість вертикального кроку з вершини з індексами i, j;

Елементи матриці *C*- c(i, j) -вартість оптимального шляху із точки (i, j) до кінцевої точки.

Ці величини зручно зобразити у вигляді двовимірних масивів: a - розміром 4 на 4, b - розміром 3 на 5 з - розміром 4 на 5.

Вихідними даними є масиви *A* і *B,* результат – масив *C*. Нумерацію рядків і стовпців у цих масивах починаємо з нуля, рухаючись від початкової точки до кінцевої.

$A_{0,0}=1$, $A_{0,1}=6$ $A_{0,2}=7$ $A_{0,3}=6$ $B_{0,0}=3$, $B_{0,1}=5$ $B_{0,2}=8$ $B_{0,3}=4$ $B_{0,4}=3$

$A_{1,0}$=2, $A_{1,1}$=7 $A_{1,2}$=2 $A_{1,3}$=5 $B_{1,0}$=7, $B_{1,1}$=5 $B_{1,2}$=4 $B_{1,3}$=8 $B_{1,4}$=9
$A_{2,0}$=6, $A_{2,1}$=3 $A_{2,2}$=8 $A_{2,3}$=2 $B_{2,0}$=4, $B_{2,1}$=2 $B_{2,2}$=1 $B_{2,3}$=9 $B_{2,4}$=5
$A_{3,0}$=2, $A_{3,1}$=5 $A_{3,2}$=8 $A_{3,3}$=3

У такий спосіб весь шлях буде складатися із семи кроків: трьох вертикальних і чотирьох горизонтальних. Необхідно знайти таку послідовність кроків, вартість якої буде мінімальною.

1 На першому етапі оптимізацію кроків починаємо з кінця. Рухаючись від кінцевої точки, записуємо у вершини графа мінімальні вартості переміщення з даного стану в кінцеву вершину. Рухатися по шляху можна тільки вліво й униз.

У кінцевій точці (3,4) записуємо нуль, тому що вартість шляху із цієї точки в саму себе дорівнює нулю. Далі переходимо до точки (3,3). Із цієї точки в кінцеву є єдиний шлях - крок вправо, його вартість дорівнює 3, цей крок і буде оптимальним. Записуємо 3 у точці (3,3) і позначаємо оптимальний крок стрілкою. Наступна точка (3,2), з неї шлях у кінцеву точку також єдиний - крок вправо до точки (3,3) плюс крок від точки (3,3) до точки (3,4). Вартість оптимального шляху із точки (3,3) у кінцеву точку відома й дорівнює 3, отже, вартість оптимального шляху від точки (3,2) до кінцевої точки дорівнює вартості шляху від точки (3,3) до точки (3,4) плюс вартість кроку із точки (3,2) у точку (3,3), тобто 3+8=11. Записуємо в точці (3,2) 11 і позначаємо стрілкою оптимальний крок. Аналогічно визначаються оптимальні кроки й вартості шляху для всіх точок, що перебувають у верхньому рядку. Для точок, розташованих у крайньому правому стовпці, дії аналогічні, тому що для них існує єдиний шлях у кінцеву точку, що складається з вертикальних кроків нагору.

Для інших точок шлях у кінцеву точку не є єдиним, тому що з них можна робити два кроки - нагору або вправо. Очевидно, що оптимальним буде крок, при якому вартість шляху до кінцевої точки буде менше. Наприклад, із точки (2,3) можна рухатися через точку (2,4) і через точку (3,3). У першому випадку вартість шляху становить 2+5=7, а в другому - 9+3=12, отже, оптимальним є крок вправо. Позначаємо цей крок стрілкою, а в точці (2,3) записуємо 7.

Аналогічно визначаються оптимальні кроки й вартості шляху для всіх інших точок. На рисунку 10.2 показаний результат оптимізації всіх кроків.

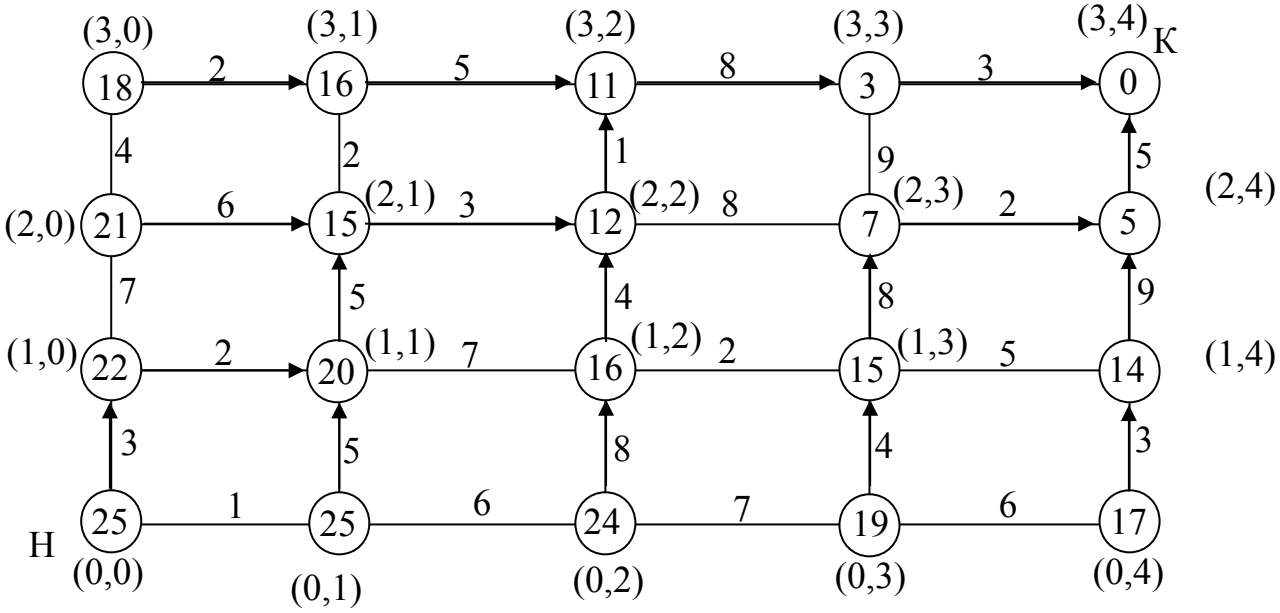


Рисунок 10.2 – Результат оптимізації кроків

2 На другому етапі оптимизируется план у цілому - з оптимальних кроків становимо оптимальний шлях, рухаючись по стрілках з початкової точки в кінцеву вправо й нагору (рисунок 10.3).

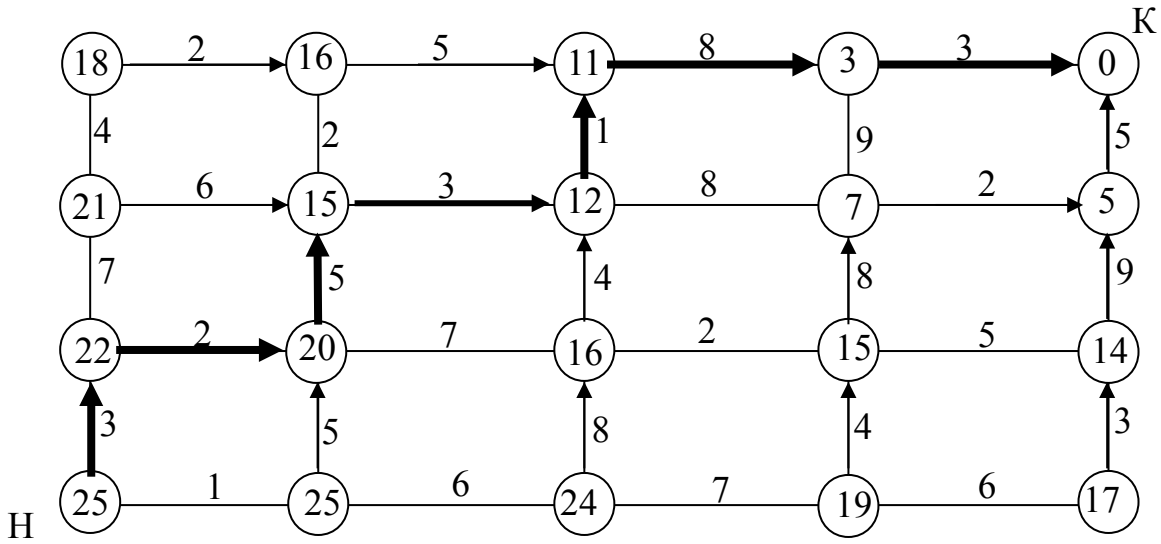


Рисунок 10.3 – Оптимальний план

На рисунках 10.4 та 10.5 наведені алгоритми описаного методу, а далі – відповідна програма.

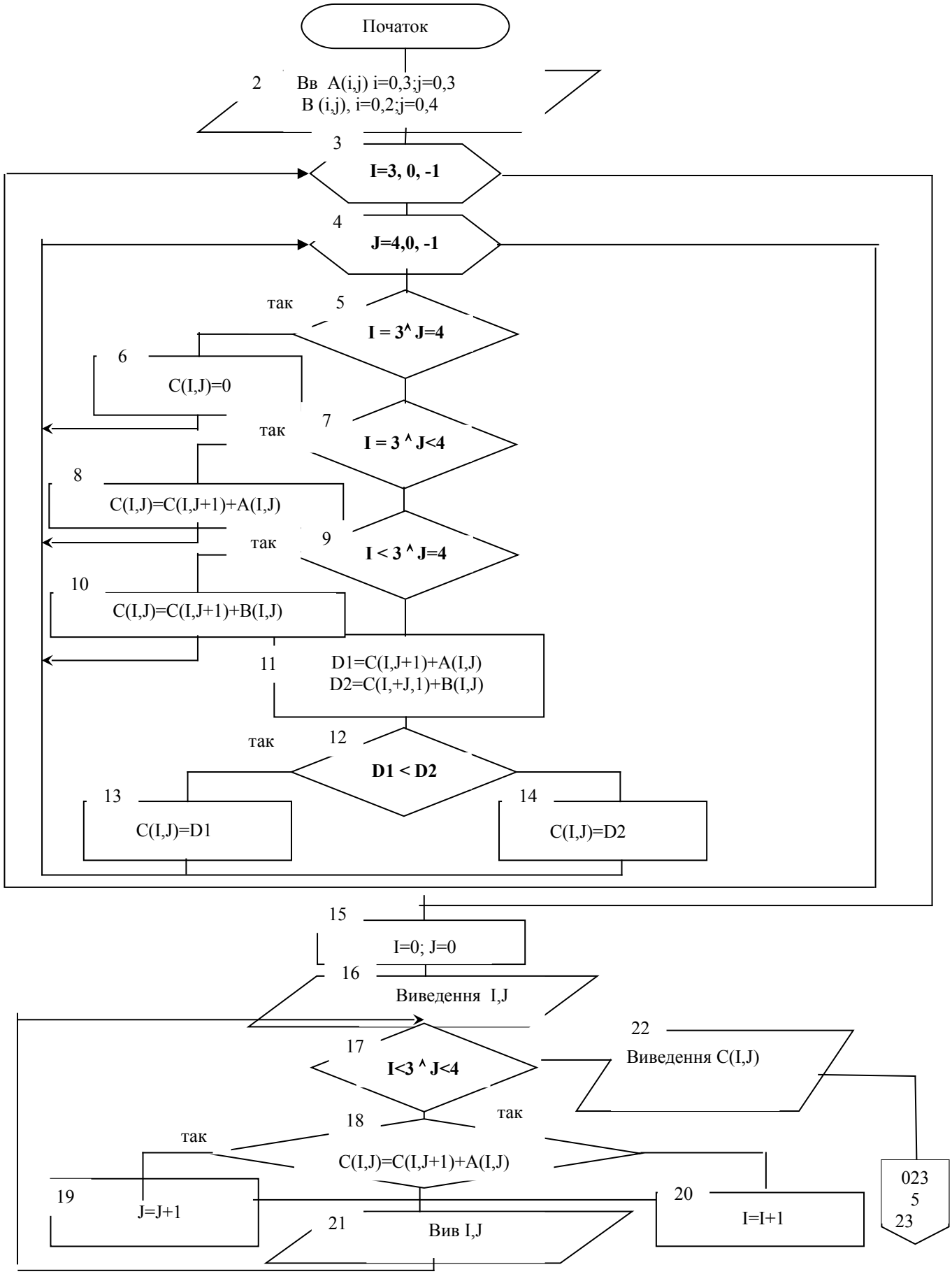


Рисунок 10.4 – Алгоритм оптимізації кроків та обчислення c(i,j)

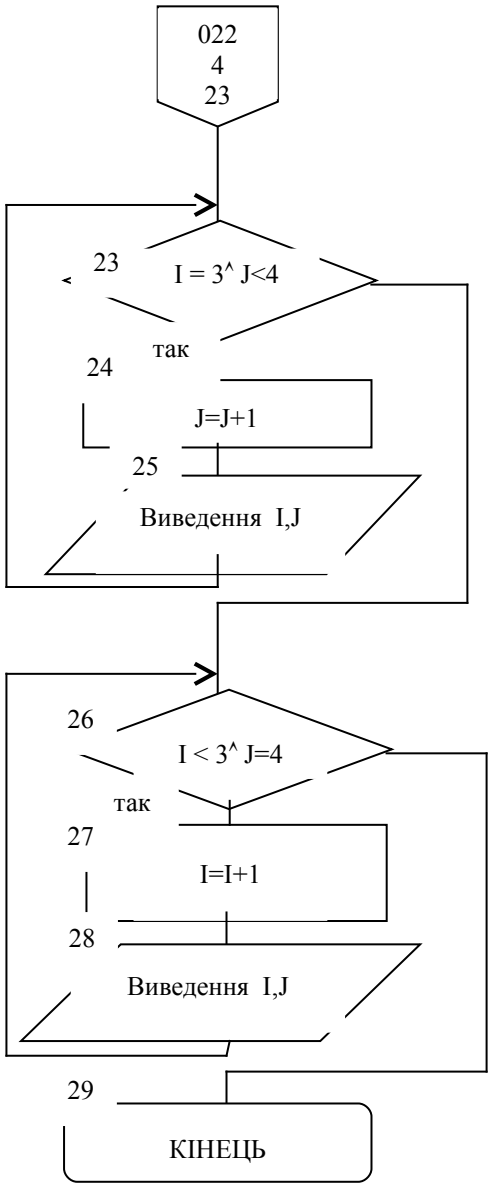


Рисунок 10.5 – Алгоритм визначення оптимального шляху

Basic-програма (алгоритм на рисунку 10.5)

```
CLS
PRINT "Задайте розмір матриці вартості"
INPUT "Число рядків "; N
povt: INPUT "Число стовпців "; M
IF M > 7 OR M < 1 THEN
        PRINT "Число стовпців поза діапазоном.
Повторіть введення"
        GOTO povt
```

```
END IF
DI C(N - 1, M - 1) AS INTEGER
DI B(N - 2, M - 1) AS INTEGER
DI A(N - 1, M - 2) AS INTEGER
Begin: CLS
PRINT "   Вихідний граф"
FOR I = N - 1 TO 0 STEP -1
    H = CSRLI + 1
    FOR J = 0 TO M - 1
       LOCATE (H), (1 + 10 * J)
       PRINT "("; I; ","; J; ")";
       IF J <> M - 1 THEN INPUT "", A(I, J)
       IF I <> 0 THEN
          LOCATE (H + 1), (5 + 10 * J)
          INPUT "", B(I - 1, J)
       END IF
    NEXT J
NEXT I
PRINT
CLS
'перевірка
PRINT
PRINT "  Заповнений вихідний граф"
FOR I = N - 1 TO 0 STEP -1
        H = CSRLI + 1
        FOR J = 0 TO M - 1
                LOCATE (H), (1 + 15 * J)
                PRINT "("; I; ","; J; ")";
                IF J <> M - 1 THEN PRINT USING "###
"; A(I, J);
                IF I <> 0 THEN
                  LOCATE (H + 1), (3 + 15 * J)
                  PRINT USING "### "; B(I - 1, J);
                END IF
        NEXT J
NEXT I
PRINT
INPUT "Повторити введення?(Y/N) "; flag$
IF UCASE$(flag$) = "Y" THEN GOTO Begin
FOR I = N - 1 TO 0 STEP -1
       FOR J = M - 1 TO 0 STEP -1
                IF I = N - 1 AND J = M - 1 THEN
                        C(I, J) = 0
                ELSEI I = N - 1 AND J < M - 1 THEN
                       C(I, J) = C(I, J + 1) + A(I, J)
```

```
                    ELSEI I < N - 1 AND J = M - 1 THEN
                            C(I, J) = C(I + 1, J) + B(I, J)
                    ELSE
                            d1 = C(I, J + 1) + A(I, J)
                            d2 = C(I + 1, J) + B(I, J)
                            IF d1 < d2 THEN C(I, J) = d1
ELSE C(I, J) = d2
                     END IF
        NEXT J
NEXT I
CLS
PRINT "     Таблиця вартості"
FOR I = N - 1 TO 0 STEP -1
        H = CSRLI + 1
        FOR J = 0 TO M - 1
          LOCATE (H), (1 + 10 * J): PRINT "("; I;
","; J; ")"
          LOCATE (H + 1), (3 + 10 * J): PRINT USING
"### "; C(I, J);
        NEXT J
NEXT I
PRINT
PRINT "      Шлях оптимальної вартості"
I = 0
J = 0
PRINT "("; I; ","; J; ")";
DO WHILE I < N - 1 AND J < M - 1
        IF C(I, J) = C(I, J + 1) + A(I, J) THEN J = J
+ 1 ELSE I = I + 1
        PRINT "("; I; ","; J; ")";
LOOP
IF I = N - 1 AND J < M - 1 THEN J = J + 1: PRINT "(";
I; ","; J; ")";
IF I < N - 1 AND J = M - 1 THEN I = I + 1: PRINT "(";
I; ","; J; ")"
```

Список літератури

1 Бережная Е.В., Бережной В.И. Математические методы моделирования экономических систем. – М.: Финансы и статистика, 2003. – 368 с.

2 Боглаев Ю.П. Вычислительная математика и программирование. – М.: Высш. школа, 1990.

3 Вентцель Е.С. Исследование операций .– М.: Наука, 1972.

4 Глухов В.В., Медников М.Д., Коробко С.Б. Математика в бизнесе: Учебн. для вузов. Специальная литература. – М.: Лань, 2005. – 528 с.

5 Джашитов В.Э., Панкратов В.М. Математические модели теплового дрейфа гироскопических датчиков инерциальных систем / Под общ. ред. акад. РАН В.Г.Пешехонова. – СПб.: ГНЦ РФ - ЦНИИ "Электроприбор", 2001. - 150 с.

6 Катулев А.Н., Северцев Н.А. Математические методы в системах поддержки принятия решений. – М.: Высшая школа, 2005. – 311 с.

7 Комп’ютерна техніка і програмування. Методичні вказівки до розрахунково-графічної роботи з дисципліни «Комп’ютерна техніка і програмування» /С.Є.Бантюков, О.Є.Пєнкіна, С.О.Бантюкова. – Харків: УкрДАЗТ, 2006. – 50 с.

8 Красс М.С. Математика в экономике. Математические модели и методы. – М.: Финансы и статистика, 2007. – 544 с.

9 Маликов В.Т., Кветный Р.Н. Вычислительные методы и применение ЭВМ. – К.: Вища школа, 1989.

10 Крылов В.И., Бобков В.В., Монастырный П.И. Вычислительные методы. – М.: Наука, 1976. – т. 1.

11 Методические указания к лабораторно-практическим работам N1-4 по дисциплине "Математические методы и модели в расчетах на ЭВМ" (Исследование детерминированных моделей). - Харьков: ХарГАЖТ, 1995.

12 Методы и модели оптимизации. Методические указания к лаб.работам. – Х.: ХИИТ, 1993.

13 Методы классической и современной теории автоматического управления: В 5 томах. Том 1: Математические модели, динамические харатеристики и анализ систем автоматического управления: Учебник / Под ред. К.А. Пупкова. – М.: Высш. шк., 2004. – 322 с.

14 Невежин В. Сборник задач по курсу "Экономико-математическое моделирование". – М.: Городец, 2005. – 320 с

15 Петров А.В. и др. Вычислительная техника в инженерных и экономических расчетах. – М.: Высшая школа, 1984.

16 Советов Б.Я., Яковлев С.А. Моделирование систем. - М.: Высшая школа, 1985.

17 Філіппенко І.Г., Гончаров В.О. Меркулов В.С. Основи проектування технічних систем. Класифікація моделей. Основи теорії оптимізації: Конспект лекцій з дисципліни "Математичні методи і моделі в розрахунках на ЕОМ" – Ч. 1. – Харків: ХарДАЗТ, 2004.

18 Філіппенко І.Г., Гончаров В.О. Меркулов В.С. Методи оптимізації: Конспект лекцій з дисципліни "Математичні методи і моделі в розрахунках на ЕОМ" – Ч. 2. – Харків: ХарДАЗТ, 2004.

19 Шелобаев С.И. Математические методы и модели. – М.: ЮНИТИ, 2001. – 367 с.

20 Шенон Р. Имитационое моделирование систем – искусство и наука. – М., 1978.

21 Щуп Т. Решение инженерных задач с использованием ЭВМ. Практическое руководство. – М.: Мир, 1982.