МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

СЕКЦІЯ ІКТ

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Порівняльний аналіз алгоритмів чисельного розв'язання задач нелінійного програмування. Алгоритм Монте-Карло»

Завідувач

випускаючої кафедри **Довбиш А. С.**

Керівник роботи **Шаповалов С. П.**

Студент групи ІНмз – 91С **Войтович І. М.**

СУМИ 2020

Сумський державний університет

(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Войтович Ігорю Миколайовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *«Порівняльний аналіз алгоритмів чисельного розв'язання задач нелінійного програмування. Алгоритм Монте-Карло»*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми. Постановка задачі дослідження. 2) Інформаційний огляд. 3) Математична модель та вибір методу рішення 4) Розробка інформаційного та програмного забезпечення системи*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми. Постановка задачі дослідження*		
2.	*Інформаційний огляд*		
3.	*Математична модель та вибір методу рішення*		
4.	*Розробка інформаційного та програмного забезпечення системи*		
5.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____

(підпис)

Керівник проекту _____

РЕФЕРАТ

Записка: 35 стр.,9 рис., 2 додатка, 10 джерел інформації.

Об’єкт дослідження – задачі нелінійного програмування.

Мета роботи – застосувати алгоритм Монте-Карло ля рішення задач нелінійного програмування.

Методи дослідження – методи математичного моделювання, алгоритм Монте-Карло, програмування на алгоритмічній мові.

Результати – виконані дослідження щодо застосування алгоритму Монте-Карло для рішення задач нелінійного програмування. Проведена комп’ютерна реалізація та її тестування на конкретних завданнях з застосуванням мови програмування С++.

ЗАДАЧІ НЕЛІНІЙНОГО ПРОГРАМУВАННЯ,
АЛГОРИТМ МОНТЕ_КАРЛО, КОМП’ЮТЕРНА РЕАЛІЗАЦІЯ
ТЕСТОВІ РОЗРАХУНКИ

ЗМІСТ

ВСТУП

Оптимізаційні розв'язання проблем - це методика виділення одного або кількох найбільш бажаних рішень з великого або нескінченного числа допустимих рішень на основі використання математичного моделювання та застосування до одержаних моделей алгоритмів та методів, здебільшого комп'ютерного спрямування [1-5]. Серед моделей найбільш часто зустрічається однокритеріальна (скалярна) оптимізація - єдиний критерій описує переваги вибору. Така властивість завдання зустрічається далеко не завжди: є велика кількість завдань, в яких є кілька критеріїв, значення кожного з яких бажано збільшити (або зменшити) при незмінних інших [1-5].

До проблем такого напряму відносяться, наприклад, задачі проектування технічних систем або систем, що впливають на навколишнє середовище.

При прийнятті фінансових рішень поряд з доходом потрібно враховувати ризики. У таких випадках потрібно використовувати багатокритеріальні методи.

Стратегічні рішення - це рішення, що мають серйозне довгострокове вплив (проектні та планові рішення, рішення про форми і параметрах регулювання економіки, про стратегію бізнесу, фінансові рішення, рішення про вибір місця навчання або роботи і т.д.). Тому стратегічні рішення заслуговують ретельного аналізу, в тому числі математичного моделювання і розробки систем підтримки прийняття рішень (СППР).

Завдання нелінійної оптимізації зустрічаються в прикладних сферах набагато частіше ніж випадок лінійний і вимагають більш складніших алгоритмів для їх рішення. Алгоритмів точного рішення таких проблем майже не існує, а якщо і існує, то вони стикаються з труднощами, наприклад нелінійної оптимізації в рамках математичної постановки, що вимагає диференційованість [1].

Алгоритм Монте-Карло вважається одним з затребуваних чисельних алгоритмів, що має застосовність в задачах, що потребують оптимального рішення. Його точність можна корегувати кількістю статистичних випробувань, що для сучасної обчислювальної техніки не є проблемою [6-10].

1 ІНФОРМАЦІЙНИЙ ОГЛЯД ПРОБЛЕМИ

Система підтримки прийняття рішень - це система комп'ютерних програм і процедур, призначена для того, щоб допомогти особам, які приймають рішення, проаналізувати складні проблеми прийняття рішень на основі використання знань, даних, моделей [1-2].

Прийняття стратегічних рішень зазвичай здійснюється за участю людей; цим прийняття стратегічних рішень принципово відрізняється від автоматичного вибору рішень, скажімо, автопілотом або штучним інтелектом. Тому при аналізі методів підтримки прийняття стратегічних рішень доводиться враховувати складні психічні (а при колективному виборі - і соціальні) процеси.

Важливість наукового підходу для прийняття рішень полягає в тому, що рішення, які людина приймає інтуїтивно, не завжди є раціональними.

Науково обґрунтований вибір альтернатив базується на різних математичних постановках та відповідних методах, які залежать від змісту конкретної прикладної задачі прийняття рішень.

На рисунку 1.1 надано схематично розв’язання завдання прийняття рішень.

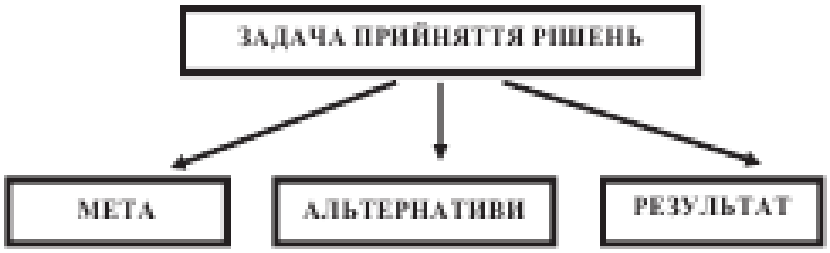


Рисунок 1.1 – Розв’язання задачі прийняття рішень

Математична модель цього процесу можна представити як описання категорійної моделі ⟨ Z, A, V ,D, P ⟩, в якій присутні:

- Z – загальний опис завдання, який повинен формалізувати проблему для рішення та необхідні модельні дані, наприклад, мету або цілі, яких потрібно досягти, а також який кінцевий результат потрібен;
- A – множина альтернативних варіантів, з котрих проводиться вибір. Їх може бути скінчена кількість або множина всіх теоретично можливих варіантів, яка може бути навіть нескінченою;

- V – множина ознак для описання варіантів та їх особливостей. За допомогою ознак характеризують альтернативи;
- D – сукупність умов, що обмежують область допустимих варіантів розв'язку задачі. Обмеження можуть бути описані як змістовним чином, так і формалізовані математично;
- P – переваги, які є основою для оцінювання та порівняння можливих варіантів рішення проблеми, відбору допустимих варіантів і пошуку найкращого або прийнятного варіанту.

Теорія прийняття рішень – це математично змодельована проблема, яка спрямована на вибір, в деякому розумінні, найкращого (найкращих) з можливих варіантів з урахуванням наявних обмежень.

1.1 Алгоритми та методи рішення задач нелінійного програмування

Задачі нелінійної оптимізації потребували розробки спеціальних методів та досліджень по їх розв'язанню, що привело до виникнення в галузі знань, що відноситься до програмування — нелінійного програмування.

Але всі ці дослідження не привели до винайдення єдиного універсального методу їх розв'язання (як наприклад, симплекс метод в лінійному програмуванні), то виникла роздільність всіх методів за зручністю їх використання для розв'язування певного класу завдань нелінійного програмування. Кожний практичний приклад обирав по-суті кращий метод (рисунок 1.2).

Клас задач нелінійного програмування ширше класу задач лінійного програмування. Докладне вивчення практичних завдань, які домовилися вважати лінійними, показує, що вони в дійсності є нелінійними. Існуючі методи дозволяють вирішувати вузький клас задач, обмеження яких мають вигляд, а цільова функція є сепарабельною (сумою n функцій), або квадратичною.

Нелінійне програмування (NLP, англ. NonLinear Programming) — випадок математичного програмування, у якому цільовою функцією чи обмеженнями є нелінійна функція.

Задача нелінійного програмування ставиться як задача знаходження оптимуму певної цільової функції $F(x_1,x_2,x_3,\ldots x_n)$ при виконанні умов

$G_j\ (x_1,x_2,x_3,\ldots x_n\ ) \geq 0,$

де $x_1,x_2,x_3,\ldots x_n$ — параметри, Gj — обмеження, n — кількість параметрів, j — кількість обмежень.

На відміну від задачі лінійного програмування в задачі нелінійного програмування оптимум не обов'язково лежить на границі області, визначеної обмеженнями.

Рисунок 1. 2 – Загальна класифікація методів розв’язку задач нелінійного програмування

Для розв’язування задач нелінійного програмування не існує універсального методу, а тому доводиться застосовувати багато методів і обчислювальних алгоритмів, які ґрунтуються, здебільшого, на теорії диференціального числення,

і вибір їх залежить від конкретної постановки задачі та форми економіко-математичної моделі.

Методи нелінійного програмування бувають прямі та непрямі. Прямими методами оптимальні розв'язки відшукують у напрямку найшвидшого збільшення (зменшення) цільової функції. Типовими для цієї групи методів є градієнтні. Непрямі методи полягають у зведенні задачі до такої, знаходження оптимуму якої вдається спростити. До них належать, насамперед, найбільш розроблені методи квадратичного та сепарабельного програмування.

Нижче наведені деякі постановки таких завдань.

Сепарабельна задача

$$f(x_1, x_2, ..., x_n) = \sum_{j=1}^{n} f_j(x_j) \to \max(\min)$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, (i = 1 \div m)$$

Квадратична задача НП

$$f(x_1, x_2, ..., x_n) = \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_i x_j \to \max(\min)$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, (i = 1 \div m)$$

Задача опуклого програмування

$$f(x_1, x_2, ..., x_n) \qquad g_i(x_1, x_2, ..., x_n)$$

В прикладній науці деякі прості аналізи даних можна зробити лінійними методами, але загалом ці проблеми також є нелінійними. Як правило, існує теоретична модель досліджуваної системи із змінними параметрами в ній та модель експерименту чи експериментів, які також можуть мати невідомі параметри.

Намагатись розв'язати задачі нелінійного характеру теоретичними методами по-перше потребує значного часу, а по-друге може привести до значного часу.

В звязку з цим, виникає ідея застосувань чисельних розв’язань, але тоді виникає проблема відповідності. У цьому випадку часто потрібна міра точності результату, а також найкраща відповідність.

Оптимізаційні задачі, на змінні яких не накладаються обмеження, розв’язують методами класичної математики. Оптимізацію з обмеженнями-рівностями виконують методами зведеного градієнта, скажімо методом Якобі, та множників Лагранжа. У задачах оптимізації з обмеженнями-нерівностями досліджують необхідні та достатні умови існування екстремуму Куна—Таккера.

Одним із методів, які дозволяють звести задачу нелінійного програмування до розв'язування системи рівнянь є метод невизначених множників Лагранжа.

Розглянемо метод множників Лагранжа на прикладі такої задачі нелінійного програмування:

$$Z = f(x_1, x_2, \ldots, x_n) \to \max(\min) \quad (1.1)$$

за умов

$$q_i(x_1\ x_2, \ldots, x_n) = b_i \quad (1.2)$$

$$(i = \overline{1, m}),$$

де функції $f(x_1, x_2, \ldots, x_n)$ і $q_i(x_1, x_2, \ldots, x_n)$ диференційовані.

Ідея методу множників Лагранжа полягає в заміні даної задачі простішою: на знаходження екстремуму складнішої функції, але без обмежень. Ця функція називається функцією Лагранжа і подається у вигляді:

$$L(x_1, x_2, \ldots, x_n;\ \lambda_1, \lambda_2, \ldots, \lambda_m) =$$

$$= f(x_1, x_2, \ldots, x_n) + \sum_{i=1}^{m} \lambda_i (b_i - q_i(x_1, x_2, \ldots, x_n)), \quad (1.3)$$

де λі — не визначені поки що величини, так звані множники Лагранжа.

Знайшовши частинні похідні функції L за всіма змінними і прирівнявши їх до нуля:

$$\begin{cases} \dfrac{\partial L}{\partial x_j} = 0 \quad (j = \overline{1, n}), \\ \dfrac{\partial L}{\partial \lambda_i} = 0 \quad (i = \overline{1, m}), \end{cases}$$

запишемо систему

$$\begin{cases} \dfrac{\partial f(x_1,\ x_2,\ ...,\ x_n)}{\partial x_j} + \sum\limits_{i=1}^{m} \lambda_i \dfrac{\partial q_i(x_1,\ x_2,\ ...,\ x_n)}{\partial x_j} = 0 & \left(j = \overline{1, n}\right), \\ b_i - q_i(x_1,\ x_2,\ ...,\ x_n) = 0 & \left(i = \overline{1, m}\right), \end{cases} \quad (1.4)$$

що є, як правило, нелінійною.

Розв'язавши цю систему, знайдемо $X^* = (x_1,\ x_2,\ ...,\ x_n)$ і $\lambda = (\lambda_1,\ \lambda_2,\ ...,\ \lambda_m)$ — стаціонарні точки. Оскільки їх визначено з необхідної умови екстремуму, то в них можливий максимум або мінімум. Іноді стаціонарна точка є точкою перегину (сідлова точка). Отже, для визначення достатніх умов екстремуму та діагностування його типу існує спеціальний алгоритм [1].

Недоліком цього методу є обов'язковість диференціювання, результатом якого є потім обчислення системи рівнянь. В разі нелінійності функції та обмежень ця система буде нелінійної й її аналітичне рішення може зовсім не існувати й ця система може потребувати пошуку свого рішення в області чисельних алгоритмів.

Актуальною стає проблема застосовності чисельних алгоритмів при розв'язані задач нелінійного програмування.

Навіть питання щодо існування розв'язку задачі нелінійного програмування потребує окремого дослідження.

Розглянемо основні труднощі розв'язування нелінійних задач.

Для лінійних задач можна завжди знайти оптимальний розв'язок універсальним методом — симплексним. При цьому не існує проблеми стосовно доведення існування такого розв'язку. Можливі варіанти:

а) отримали оптимальний розв'язок;

б) умови задачі суперечливі, тобто розв'язку не існує;

в) цільова функція необмежена, тобто розв'язку також не існує.

Для задач нелінійного програмування не існує універсального методу розв'язання, що зумовило розроблення значної кількості різних методів розв'язування окремих типів задач нелінійного програмування. Для кожного специфічного методу необхідно доводити існування розв'язку задачі, що також є досить складною математичною задачею.

Відомі точні методи розв’язування нелінійних задач, але в такому разі існують труднощі обчислювального характеру, тобто навіть для сучасних ЕОМ такі алгоритми є досить трудомісткими, тому здебільшого для розв’язування нелінійних задач виправданим є застосування наближених методів.

Існують труднощі в пошуку області допустимих рішень, бо взагалі ці області можуть бути кусочно-розривними (рисунок 1.3)

$$\begin{cases}(x_1-1)x_2 \le 1,\\ x_1+x_2 \ge 3.5,\end{cases}$$

$$x_1, x_2 \ge 0$$

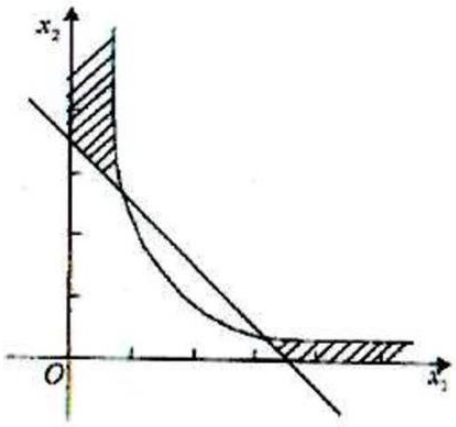


Рисунок 1. 3 – Кусочно розривна область пошуку рішення

Навіть якщо область допустимих рішень - опукла, то в ряді завдань цільова функція може мати декілька локальних екстремумів. За допомогою більшості ж обчислювальних методів можна знайти точку локального оптимуму, але не можна встановити, чи є вона точкою глобального (абсолютного) оптимуму чи ні. Якщо завдання містить нелінійні обмеження, то область допустимих рішень не є опуклою і крім глобального оптимуму можуть існувати точки локального оптимуму.

1.2 Постановка задачі

Прийдемо до наступної постановки задачі.

Для заданого завдання нелінійного програмування адаптувати метод Монте-Карло для його чисельного розв'язання.

. Для досягнення мети провести наступні дослідження:

1. *Створити комп'ютерну реалізацію алгоритму Монте-Карло для рішення задач нелінійного програмування.*
2. *Протестувати її працездатність на рішеннях відомих завдань.*
3. *Провести вдосконалення методу для звуження області пошуку рішень.*

2 МАТЕМАТИЧНА МОДЕЛЬ ТА ВИБІР МЕТОДУ ЇЇ РІШЕННЯ

2.1 Загальна математична модель

Поставимо загальну математичну постановку задачі нелінійного програмування:

Максимізувати(мінімізувати) функцію цілі F

$$\mathbf{F = f(x_1, x_2, x_3, \ldots, x_n) \rightarrow max(min),} \qquad (2.1)$$

в якій керовані змінні $\mathbf{(x_1, x_2, x_3, \ldots, x_n)}$ вибираються з області допустимих рішень **D**, що задається сукупністю обмежень

$$\begin{cases} G(\mathbf{x1, x2, x3, \ldots, xn}) \geq \mathbf{0}, \\ Q(\mathbf{x1, x2, x3, \ldots, xn}) \leq \mathbf{0}, \\ \mathbf{x1, x2, x3, \ldots, xn} \geq \mathbf{0} \end{cases} \qquad (2.2)$$

Нелінійність може проявлятись в аналітичному подання як самої функції цілі (2.1) , так і в обмеженнях (2.2), і взагалі й функціонально і в обмеженнях.

Розв'язанням (рішенням) поставленого завдання є пошук в області допустимих рішень точка (набору керованих параметрів), яка приведе функцію цілі до шуканого екстремуму.

Наприклад, Знайти максимальне й мінімальне значення функції

$$F = (x_1 - 3)^2 + (x_2 - 4)^2$$

При обмеженнях

$$\begin{cases} 3x_1 + 2x_2 \geq 7, \\ 10x_1 - x_2 \leq 8, \\ -18x_1 + 4x_2 \leq 12, \\ x_1, x_2 \geq 0. \end{cases}$$

В цій постановці нелінійність ввійшла до функції цілі.

Покажімо графічне розв'язання даного завдання (див. рис. 2. 1).

Аналітично доопрацюємо до відповіді:

F max = 65 в точці C з координатами (3, 11).

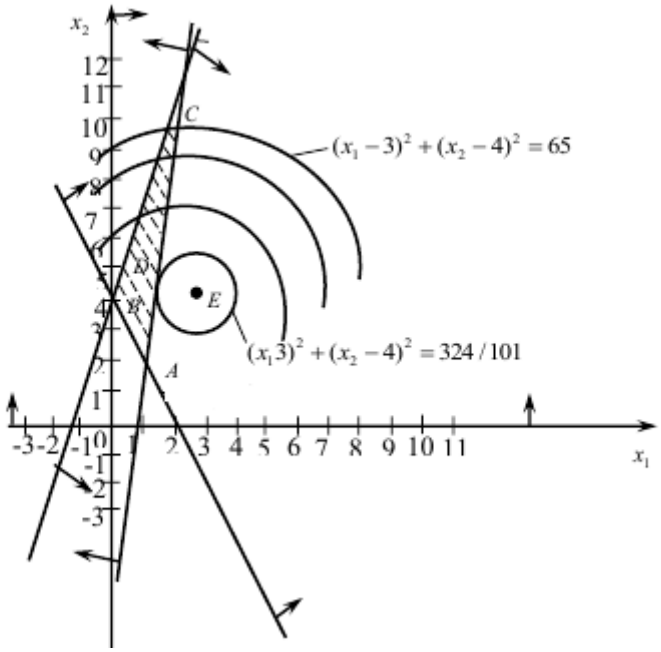


Рисунок 2.1 – Графічне розв’язання завдання нелінійного програмування

Перелічимо проблеми, з якими стикаються при розв’язанні задач нелінійного програмування:

1) Множина D допустимих рішень може бути неопуклою, незв'язною, мати нескінченне число крайніх точок.
2) Функція цілі може досягатися свого max(min) не тільки на границі D, але і всередині області допустимих рішень.
3) Цільова функція в області D може мати кілька локальних екстремумів.

Існують декілька методів для розв'язування неопуклих задач. Один підхід полягає у використанні спеціальних формулювань задач лінійного програмування. Інший метод передбачає використання методів гілок і меж, де задача поділяється на підкласи, щоби бути розв'язаною з опуклими (задача мінімізації) або лінійними апроксимаціями, які утворюють нижню межу загальної вартості у межах поділу.

При наступних поділах у певний момент буде отримано фактичний розв'язок, вартість якого дорівнює найкращій нижній межі, отриманій для будь-якого з наближених рішень. Цей розв'язок є оптимальним, хоча, можливо, не єдиним. Алгоритм можна також припинити на ранній стадії, з упевненістю, що оптимальний розв'язок знаходиться в межах допустимого відхилення від знайденої

кращої точки; такі точки називаються ε-оптимальними. Завершення біля ε-оптимальних точок, як правило, необхідне для забезпечення скінченності завершення.

Це особливо корисно для великих, складних задач і задач з невизначеними витратами або значеннями, де невизначеність може бути оцінена з відповідної оцінки надійності.

2.2 Вибір методу рішення

Алгоритм Мóнте-Ка́рло (названий за назвою однойменного міста в Монако, яке відоме своїми казино, в яких як відомо рулетка є основним інструментом) — загальна назва групи числових методів, заснованих на одержанні великої кількості реалізацій стохастичного (випадкового) процесу, який формується у той спосіб, щоб його ймовірнісні характеристики збігалися з аналогічними величинами задачі, яку потрібно розв'язати.

Методи Монте-Карло можна розглядати як сукупність обчислювальних методів для (зазвичай чисельного наближеного спрямування) рішення математичних задач, в яких використовуються фундаментальні випадкові вибірки. В рамках цієї системи найчастіше вирішуються два класи статистичних проблем: інтеграція і оптимізація.

Вже з початку електронних обчислень деяка частина досліджень відводилась проведенням випадкових експериментів на комп’ютері. На сьогодні такі методи Монте-Карло (МСМ) є важливим компонентом багатьох кількісних досліджень [7-10]. Які причини спокутали МСМ перетворити у провідну інформаційно-комунікаційну технологію, що в сьогоденні увійшла до сучасної науки?

Важливим моментом в цьому є розвиток самої комп’ютерної техніки, коли швидкодія сучасних комп’ютерів вимірюється вже десятками-сотнями мільярдів операцій в секунду. Тому провести комп’ютерний експеримент не представляє труднощів.

Додаймо ще до цього низку причин наступним чином:

- Алгоритми Монте-Карло не складні та ефективні;
- Алгоритми Монте-Карло, як правило, прості, гнучкі та масштабовані;

- ❖ Застосовуючись до фізичних систем, методи Монте-Карло можуть звести складні моделі до набору основних подій та взаємодій, відкриваючи можливість кодування поведінки моделі за допомогою набору правил, які можуть ефективно впроваджуватися на комп’ютері;
- ❖ Й на завершення, алгоритми Монте-Карло є надзвичайно паралелізабельними, зокрема, коли вони різні деталі можна запускати самостійно. Це дозволяє запускати деталі на різних комп'ютерах та / або процесорах, отже істотно скорочуючи час обчислень.

Моделювання Монте-Карло, по суті, є генерацією випадкових об'єктів або процесів за допомогою комп'ютера. Ці об'єкти можуть виникнути "природним шляхом" як частина моделювання системи реального життя, наприклад складна мережа доріг, транспорт нейтронів, або еволюція фондового ринку. У багатьох випадках однак випадкові об'єкти в техніках Монте-Карло вводяться "штучно" для розв'язування суто детерміновані проблеми.

Застосовність ідей МСМ широко застосовується й для рішення життєвих проблем, наприклад, в спорті при жеребкуванні (див. рис. 2.2)

Рисунок 2. 2 – Застосовність МСМ при жеребкуванні

В галузі освіти методи Монте-Карло найбільш цікаві як обчислювальний пристрій для виконання статистичного висновку. Багато цікавих моделі мають надзвичайно складну структуру і не можуть бути легко вирішені з використанням традиційних методів. В рамках байєсівської парадигми вся інформація, на якій може грунтуватися висновок, кодується в межах апостеріорного розподілу ймовірностей. Використовуючи методи Монте-Карло, ми можемо

охарактеризувати ці розподілу і розрахувати очікування по ним: основний метод логічного висновку.

В основі застосування для рішення будь-якого завдання алгоритмом Монте-Карло лежить генератор випадкових чисел: процедура, яка створює нескінченний потік випадкових величин, які є незалежними та однаково розподіленими відповідно до деякого розподілу ймовірностей.

Коли цей розподіл є рівномірним розподілом на інтервалі (0,1) (тобто Dist = U (0, 1)), генератор називається рівномірним генератором випадкових чисел. Більшість комп'ютерних мов програмування вже містять в собі вбудований єдиний генератор випадкових чисел. Як правило, користувачеві пропонується лише ввести початкове число, яке називається насінням, і після виклику генератор випадкових чисел створює послідовність незалежних рівномірних випадкових величин на інтервалі (0, 1).

Генератори засновані на простих алгоритмах, які можна легко реалізувати на комп'ютері. Такі алгоритми зазвичай можна представити у вигляді кортежу (S, f, μ, U, g), де

• S - скінченна сукупність станів,

• f - функція від S до S,

• μ - розподіл ймовірностей на S,

• U - вихідний простір; для рівномірного генератора випадкових чисел U дорівнює інтервал (0, 1), і ми будемо вважати це відтепер, якщо не інше вказано,

• g - це функція від S до U.

Тоді генератор випадкових чисел має таку структуру:

Алгоритм 1.1 (загальний генератор випадкових чисел)

1. Ініціалізація: Витягніть насіння S0 з розподілу μ на S. Встановіть t = 1.

2. Перехід: встановити St = f (St − 1).

3. Вихід: встановіть Ut = g (St).

4. Повторення: встановіть t = t + 1 і поверніться до кроку 2.

Алгоритм створює послідовність U1, U2, U3,. . . псевдовипадкових чисел - ми будемо називати їх просто випадковими числами. Починаючи з а певного

насіння, послідовність станів (а отже, і випадкових чисел) повинна повторюватися, оскільки простір станів скінченний. Найменша кількість зроблених кроків перед входом у раніше відвіданий стан називається тривалість періоду генератор випадкових чисел.

Намагаючись оцінити вплив якості випадкових чисел на результати моделювання в Монте-Карло, астрофізичні дослідники протестували криптографічно захищені псевдовипадкові числа, що генеруються за допомогою набору інструкцій RDRAND від Intel, порівняно з тими, що отримані з алгоритмів, таких як Mersenne Twister, в моделюванні Монте-Карло. радіосигнали від коричневих карликів. RDRAND є найближчим генератором псевдовипадкових чисел до справжнього генератора випадкових чисел. Статистично значущої різниці між моделями, створеними за допомогою типових генераторів псевдовипадкових чисел, та RDRAND для випробувань, що складаються з генерації 107 випадкових чисел, не виявлено.

Часто для обчислень за методом Монте-Карло застосовують датчики псевдовипадкових чисел. Головна особливість таких генераторів - наявність деякого періоду.

Метод Монте-Карло можна використовувати при значеннях N що не перевищують (краще багато менших) період вашого генератора псевдовипадкових чисел. Останній факт випливає з умови незалежності випадкових величин, які використовуються при моделюванні.

При проведенні великих розрахунків потрібно переконатися, що властивості генератора випадкових чисел дозволяють вам провести ці розрахунки. У стандартних генераторах випадкових чисел (в більшості мов програмування) період найчастіше не перевищує 2 певною мірою розрядності операційної системи, а то і ще менше. При використанні таких генераторів потрібно бути надзвичайно обережним.

2.3 Адаптація алгоритму Монте-Карло для рішення задач нелінійного програмування

Застосуємо алгоритм Монте-Карло для рішення завдання нелінійного програмування. Блок схема адаптації МСМ наведена на рисунку 2.3.

Опишемо основні його складові.

1. Задаємо вхідну інформацію : функцію цілі та обмеження.
2. Генеруємо за допомогою генератора випадкових величин точки.
3. Виконаємо процедуру, визначаючу чи входять генеровані точки до області допустимих рішень. З цією метою ми проходимо «сито» обмежень, що задає область допустимих рішень.
4. Для кожної з точок, що ввійшли в область допустимих рішень (пройшли «сито» обмежень) знаходимо значення функції цілі.
5. Запускаємо процедуру пошуку точки, що забезпечує шуканий екстремум.
6. Запускаємо процедуру звуження області генерування випадкових точок, якщо це потрібно.
7. Проводимо кількість статистичних випробувань, збільшуючи їх до досягнення заданої точності розрахунків ε.
8. Точність розрахунку перевіряємо за формулою
   $F_{n+1} - F_n < \varepsilon$. Ця нерівність одночасно служить виходом із циклу .
9. При досягненні заданої точності обчислень закінчуємо роботу алгоритму.

   Наприклад, при розв’язані наведеного 2.1 прикладу, генерація точок випадковим чином буде відбуватись в області

   $X1 \in [0\ldots3]$; $x2 \in [0\ldots13]$.

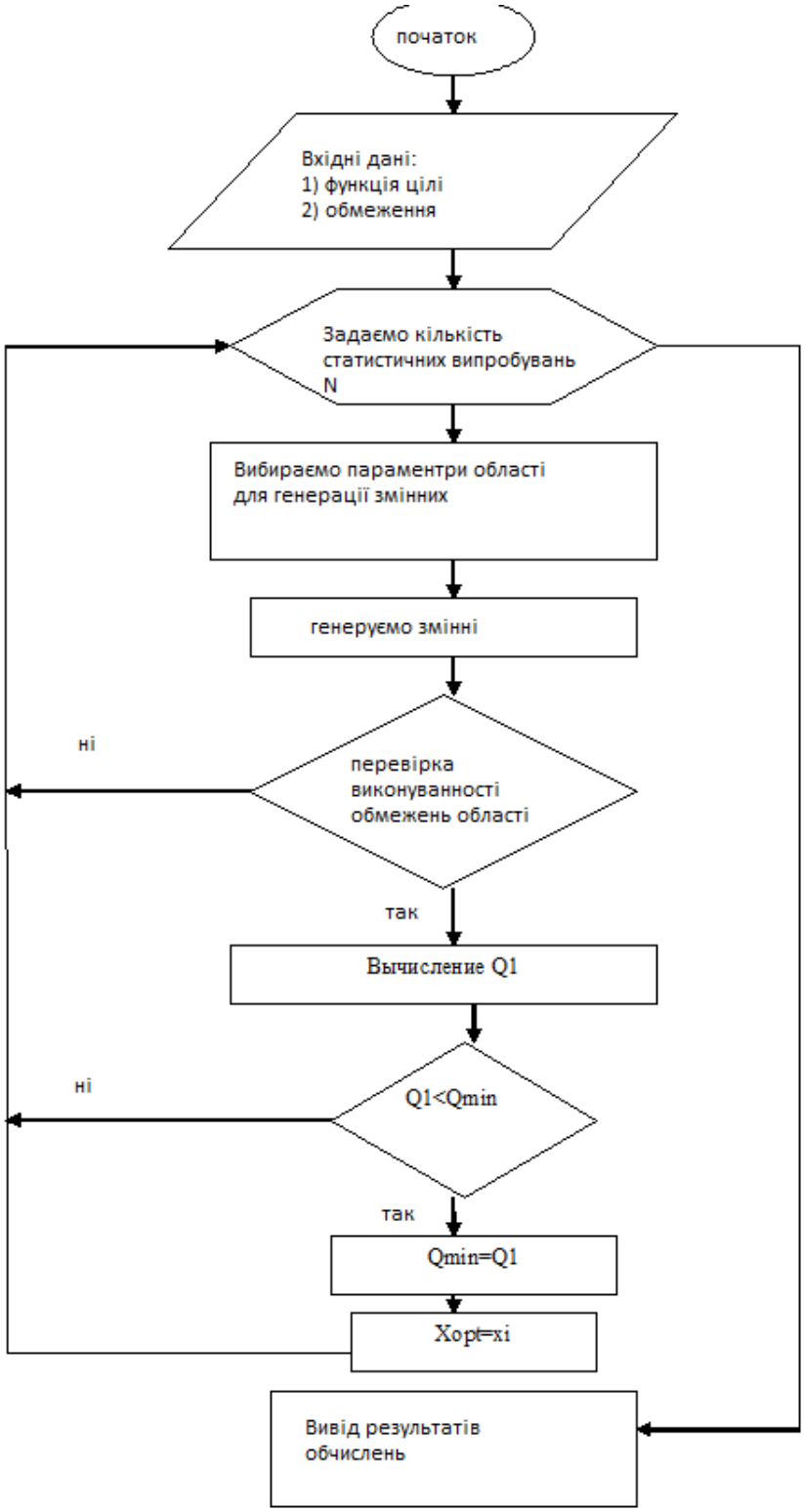


Рисунок 2. 3 – Схема загального алгоритму Монте-Карло в його адаптації для рішення задачі нелінійного програмування

3 КОМПЮТЕРНА РЕАЛІЗАЦІЯ ТА ТЕСТОВІ РОЗРАХУНКИ

3.1 Опис комп'ютерної реалізації

Нехай задано нелінійну задачу програмування в загальній математичній постановці

$$Max(min)\ F = f(x_1, x_2, x_3, \ldots x_n) \qquad (3.1)$$

при наступних обмеженнях:

$$\begin{cases} g_1(x_1, x_2, x_3, \ldots, x_n)\{\leq, =, \geq\} b_1; \\ g_2(x_1, x_2, x_3, \ldots, x_n)\{\leq, =, \geq\} b_2; \\ g_3(x_1, x_2, x_3, \ldots, x_n)\{\leq, =, \geq\} b_3; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ g_m(x_1, x_2, x_3, \ldots, x_n)\{\leq, =, \geq\} b_m; \\ x_j \geq 0\ (j = \overline{1,n}); \quad (3) \end{cases} \qquad (3.2)$$

Математично це означає, що потрібно знайти такі значення компонент n-вимірного вектору, що задовольняє системі обмежень, які задають область допустимих рішень щоб функція цілі набула оптимального значення.

Для знаходження використаємо адаптацію алгоритму Монте-Карло, блок схема, якого приведена в розділі 2.2 та мову програмування С ++ - сучасну, гнучку, мобільну мову програмування високого рівня, що використовує об'єктно-орієнтований підхід з його величезними можливостями. Код програми приведено в додатку. Він складається із двох частин А та В. Додаток А робить пошук точки оптимуму без звуження області допустимих рішень, додаток В призначений для звуження зони пошуку.

Тестування проводилися на комп'ютері з наступною конфігурацією:

- ПРОЦЕССОР Intel Core i3 3240 CPU@3.40 GHz;
- ОПЕРАТИВНА ПАМ'ЯТЬ (ОРЕ) 4.0 ГБ (3.11 ГБ досяжно);
- ОПЕРАЦІЙНА СИСТЕМА Windows 10.

Вхідною інформацією є аналітичний опис функції цілі та аналітичний опис області допустимих рішень. В коді відображені всі блоки адаптованого алгоритму Монте-Карло для рішення задач нелінійного програмування й їх реалізація представлена для рішення конкретного тестового завдання, що надається в пункті 3.2.

Для генерації випадкових координат точок використовувався генератор випадкових чисел

```
#include <iostream>
#include <math.h>
#include <time.h>
using namespace std;

double random(double min, double max) {
   return (double)(rand()) / RAND_MAX * (max - min) + min;

}
```

Перевірка входження точок в область допустимих значень відбувалася шляхом проходження «сита» нерівностей. Якщо координати точки задовольняли умовам, то точка використовувалась при знаходженні функції цілі

```
// перевірка входження точки в задану область
if ((2*x1 + 3*x2 >= 6) && (3*x1 - 2*x2 <= 18) && (-x1 + 2*x2 <=8)) {
   f = pow(x1 - 4, 2) + pow(x2 - 3, 2);
```

3.2 Тестові розрахунки

Приклад.

Поставимо завдання. Застосовуючи метод Монте-Карло знайти рішення наступної задачі нелінійного програмування:

Функція цілі $\mathbf{F = x_1^2+x_2^2+25 - 8x_1-6x_2 \rightarrow max}$

Область допустимих рішень:

$$\begin{cases} 2x_1 + 3x_2 \geq 6 \\ 3x_1 - 2x_2 \leq 18 \\ -x_1 + 2x_2 \leq 8 \\ x_1, x_2 \geq 0 \end{cases}$$

Рішення. Область допустимих рішень представлена на рисунку 3.1.

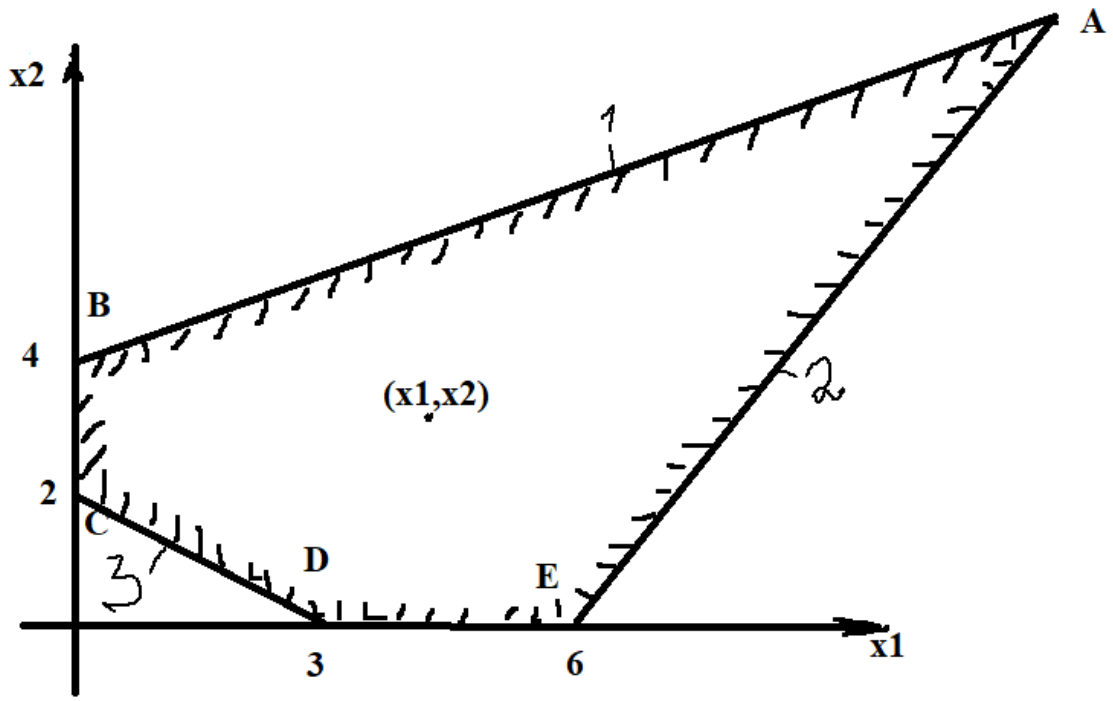


Рисунок 3.1 – Область допустимих рішень

На рисунку пряма 1 задає рівняння $-x_1+2x_2=8$, пряма 2 описується функцією $3x_1-2x_2=18$, а пряма 3 представляє рівняння $2x_1+3x_2=6$.

Згідно адаптації алгоритму Монте-Карло для рішення задач нелінійного програмування (див. розділ 2) маємо виконання по крокам:

Крок 1. Генерування випадкових точок з координатами (x1,x2) типу double :

```
double random(double min, double max){
    return (double)(rand())/RAND_MAX*(max - min) + min;
}
```

Крок 2. Перевірка входження згенерованих точок до області допустимих рішень (на рисунку ці точки вказані в області пошуку).

```
// перевірка входження точки в задану область
if ((2*x1 + 3*x2 >= 6) && (3*x1 - 2*x2 <= 18) && (-x1 + 2*x2 <=8)) {
   f = pow(x1 - 4, 2) + pow(x2 - 3, 2);
```

Крок 3. Перевірка на максимум та пошук точки, в кій відбудеться рішення.

```
// перевірка на макимум
if (f >= f_max) {

   // запам'ятовування максимальних значень
```

```
                max_x1 = x1;
                max_x2 = x2;
                f_max = f;
            }
        }
    }
    cout << "max : " << endl;
    cout << "При x1: " << max_x1 << " x2: " << max_x2 << " F = " << f_max <<
endl;
  }
};
```

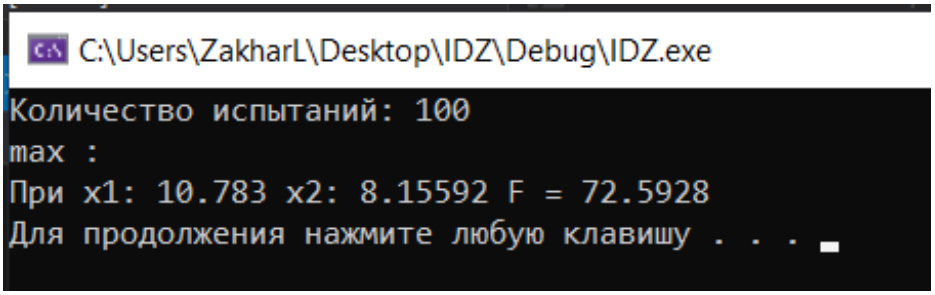
Крок 4. Звуження області пошуку рішення

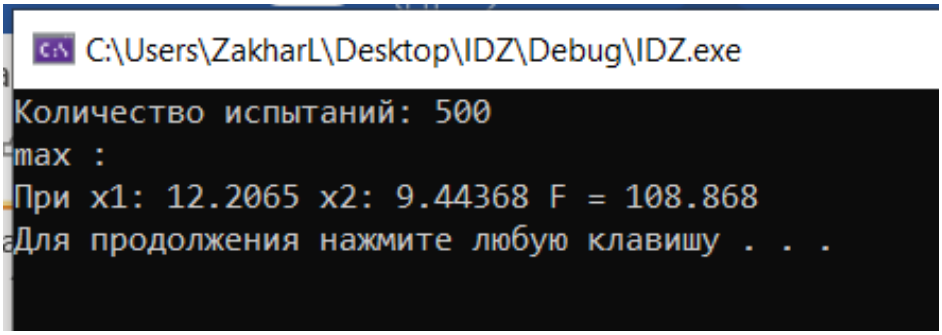
```
    for (int i = 0; i < n; ++i) {
        x1 = random(max_x1, 13);
        x2 = random(max_x2, 11);

        // перевірка входження точки в задану область
        if ((2*x1 + 3*x2 >= 6) && (3*x1 - 2*x2 <= 18) && (-x1 + 2*x2 <=8)) {
            f = pow(x1 - 4, 2) + pow(x2 - 3, 2);

            // перевірка на макимум
            if (f >= f_max) {

                // запам'ятовування максимальних значень
                max_x1 = x1;
                max_x2 = x2;
                f_max = f;
            }
        }
    }
    cout << "max : " << endl;
    cout << "При x1: " << max_x1 << " x2: " << max_x2 << " F = " << f_max <<
endl;
```

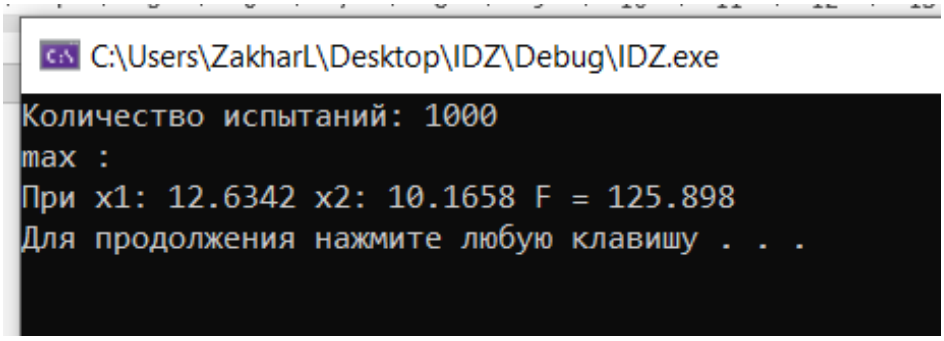
```
    }
};
```

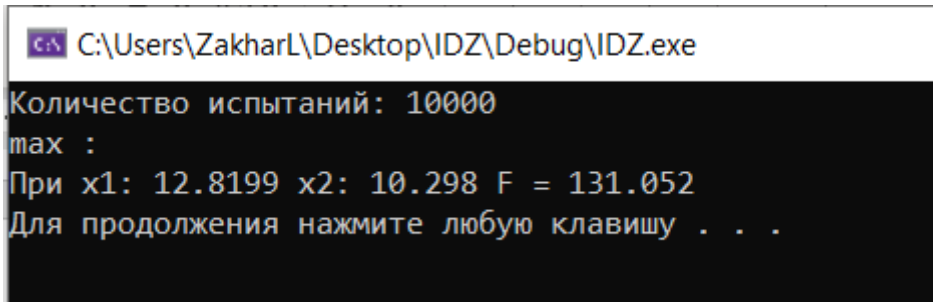
Проведемо дослідження про впотв кількості точок генерації на точність обчислення. Отримаємо наступний результат.

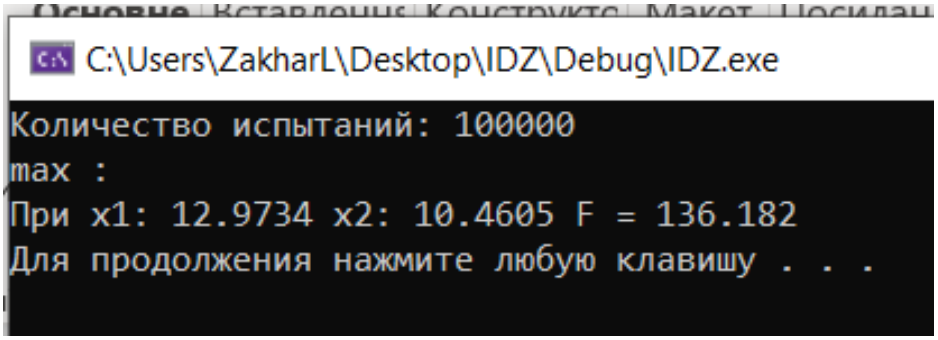
Результати

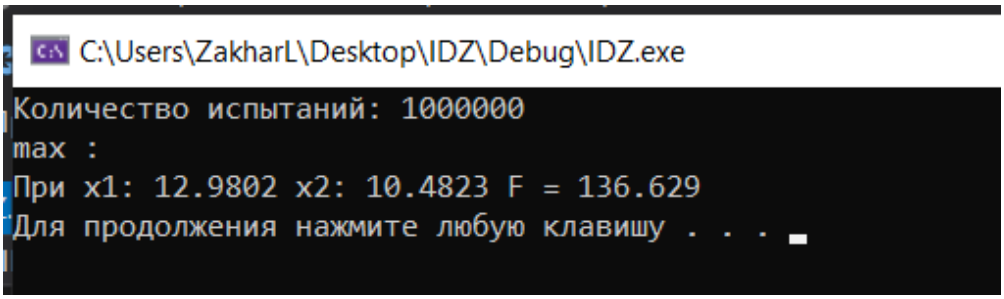










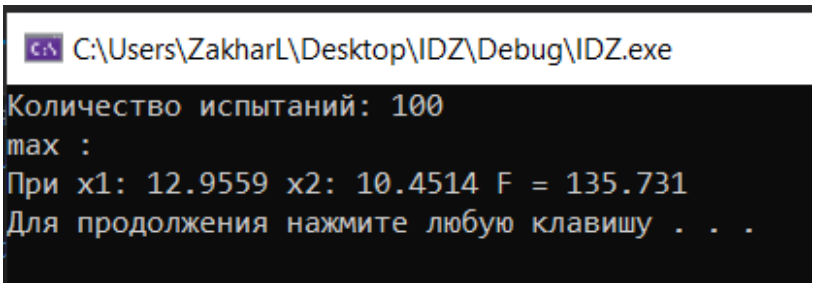


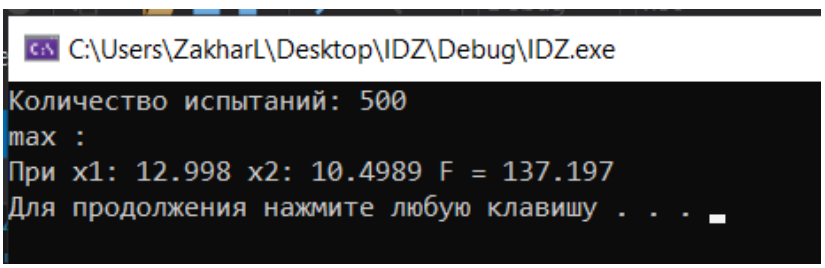
За результатами дослідження побудуємо графік залежності точності одержаного результату в залежності від кількості генерованих точок методу Монте-Карло. На рисунку 3.2 по осі Ох відкладено кількість точок генерації, по осі Оу значення функції цілі. Помічаємо закономірну тенденцію – із збільшенням кількості генерованих точок методу Монте-Карло збіжність методу збільшується.

Це завдання має точне розв’язання й воно становить:

F = 137, 25 й досягає в точці А (див. на рисунку 3.1) з координатами А(13, 10.5).

Виконання кроку 4. Приведе нас до точного рішення, як показано нижче





```
C:\Users\ZakharL\Desktop\IDZ\Debug\IDZ.exe
Количество испытаний: 1000
max :
При x1: 12.9992 x2: 10.4993 F = 137.226
Для продолжения нажмите любую клавишу . . .
```

```
C:\Users\ZakharL\Desktop\IDZ\Debug\IDZ.exe
Количество испытаний: 10000
max :
При x1: 12.9989 x2: 10.4994 F = 137.222
Для продолжения нажмите любую клавишу . . .
```

```
C:\Users\ZakharL\Desktop\IDZ\Debug\IDZ.exe
Количество испытаний: 100000
max :
При x1: 13 x2: 10.5 F = 137.249
Для продолжения нажмите любую клавишу . . .
```

```
C:\Users\ZakharL\Desktop\IDZ\Debug\IDZ.exe
Количество испытаний: 1000000
max :
При x1: 13 x2: 10.5 F = 137.25
Для продолжения нажмите любую клавишу . . .
```

На рисунку 3.3 показано досяжність точного результату за рахунок зуження області пошуку.

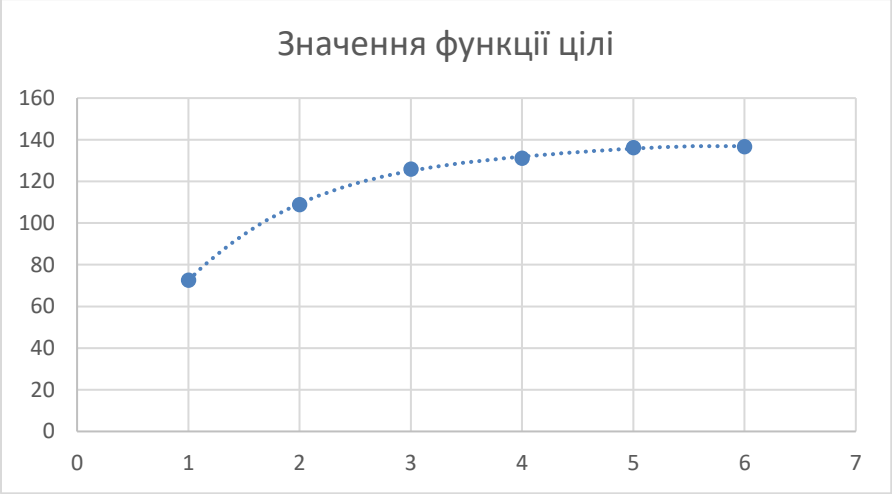


Рисунок 3.2 – Значення функції цілі в залежності від кількості точок генерації без звуження поля пошуку

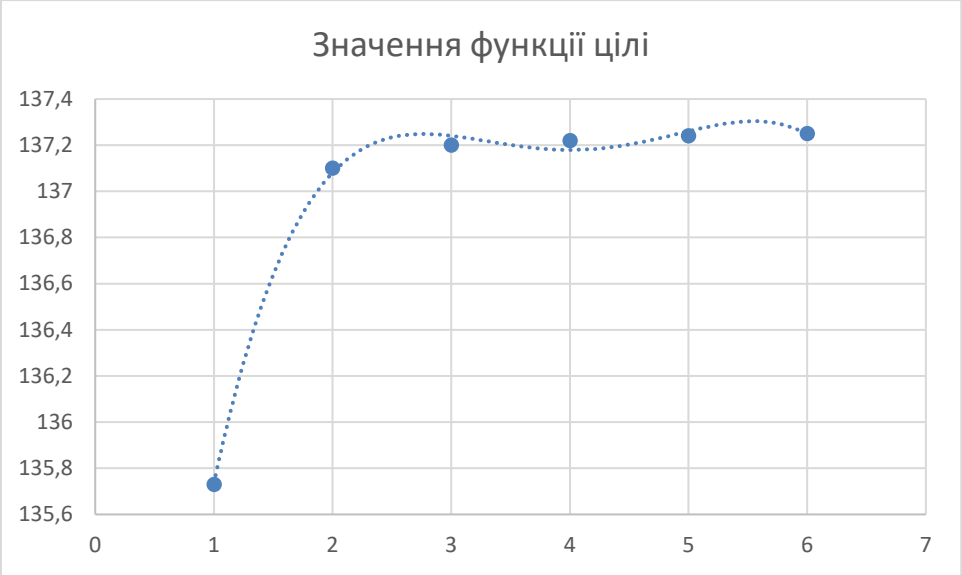


Рисунок 3.3 – Значення функції цілі в залежності від кількості точок генерації після звуження поля пошуку

Робимо висновок, що в алгоритмі Монте-Карло існує можливість звуження області пошуку рішення, що приведе до кращого результату.

ВИСНОВКИ

В магістерській роботі проведені дослідження на предмет адаптації методу Монте-Карло для рішення задач нелінійного програмування. Результати досліджень мають наступне:

1. Метод Монте-Карло може бути застосовним для рішення задач нелінійної оптимізації і являє собою чисельний метод її розв’язання.
2. Тестові розрахунки показують як за рахунок збільшення точок генерування можна поліпшити точність обчислення.
3. Метод Монте-Карло має в собі можливість звуження області пошуку допустимих рішень і тим самим збільшує точність обчислень.

СПИСОК ЛІТЕРАТУРИ

1. Математичні методи дослідження операцій : підручник / Є. А. Лавров, Л. П. Перхун, В. В. Шендрик та ін. – Суми : Сумський державний університет, 2017. – 212 с.
2. Файнзільберг Л. С., Жуковська О. А., Якимчук В. С. Теорія прийняття рішень. – Київ: Освіта України, 2018. – 246 с.
3. Стивенс Р. Алгоритмы. Теория и практическое применение. – Москва: Изд. «Э», 2016. – 544 с.
4. Stephens R. Essential Algorithms: A Practical Approach to Computer Algorithms Using Python and C#. - Wiley, 2019. — 800 p.
5. David G. Luenberger, Yinyu Ye Linear and Nonlinear Programming. - Springer, 2015. - 546 p.
6. Шаповалов С. П. Застосування методу Монте-Карло в обчислювальних моделях // МА:: ІМА – 2020. Матеріали та програма науково-технічної конференції СумДУ. 2020. - с. 84.
7. Mazhdrakov M. The Monte Carlo Method: Engineering Applications/ Mazhdrakov M. , Benov D. , Valkanov N. – ACMO Academic Press, 2018. – 250 p.
8. Lemire D. Fast Random Integer Generation in an Interval, 2019. - 15 p. [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/pdf/1805.10941.pdf
9. Krauth W. Introduction to Monte Carlo algorithms, 2006. – 41 p. [Електронний ресурс] – Режим доступу до ресурсу: https://cel.archives-ouvertes.fr/cel-00092936/document
10. Dirk P. Kroese, Tim Brereton, Thomas Taimre, Zdravko I. Botev Why the Monte Carlo method is so important today// WIREs Comput Stat, 2014, № 6. – p. 386–392.

ДОДАТОК А

```
#include <iostream>
#include <math.h>
#include <time.h>
using namespace std;

double random(double min, double max) {
   return (double)(rand()) / RAND_MAX * (max - min) + min;

}

class Point {
   double x1;
   double x2;

public:
   void fun(int n) {
      double f;
      double f_max = 0;
      double max_x1 = 0, max_x2 = 0;
      int k = 0;

      for (int i = 0; i < n; ++i) {
         x1 = random(0, 13);
         x2 = random(0, 11);

         // перевірка входження точки в задану область
         if ((2*x1 + 3*x2 >= 6) && (3*x1 - 2*x2 <= 18) && (-x1 + 2*x2 <=8)) {
            f = pow(x1 - 4, 2) + pow(x2 - 3, 2);

            // перевірка на макимум
            if (f >= f_max) {

               // запам'ятовування максимальних значень
               max_x1 = x1;
```

```
                max_x2 = x2;
                f_max = f;
            }
        }
    }
    cout << "max : " << endl;
    cout << "При x1: " << max_x1 << " x2: " << max_x2 << " F = " << f_max << endl;
  }
};

int main() {
  setlocale(LC_ALL, "ru");
  srand((unsigned int)time(0));
  int n;
  Point z;

  cout << "Количество испытаний: ";
  cin >> n;

  z.fun(n);

  system("pause");
  return 0;
}
```

ДОДАТОК В

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <random>
using namespace std;

double random(double min, double max) {
    return (double)(rand()) / RAND_MAX * (max - min) + min;
}

class Point {
    double x1;
    double x2;

public:
    void fun(int n) {
        double f;
        double f_max = 0;
        double max_x1 = 0, max_x2 = 0;
        int k = 0;

        for (int i = 0; i < n; ++i) {
            x1 = random(max_x1, 13);
            x2 = random(max_x2, 11);

            // перевірка входження точки в задану область
            if ((2*x1 + 3*x2 >= 6) && (3*x1 - 2*x2 <= 18) && (-x1 + 2*x2 <=8)) {
                f = pow(x1 - 4, 2) + pow(x2 - 3, 2);
```

```
            // перевірка на макимум
            if (f >= f_max) {

                // запам'ятовування максимальних значень
                max_x1 = x1;
                max_x2 = x2;
                f_max = f;
            }
        }
    }
    cout << "max : " << endl;
    cout << "При x1: " << max_x1 << " x2: " << max_x2 << " F = " << f_max <<
endl;
  }
};

int main() {
  setlocale(LC_ALL, "ru");
  srand((unsigned int)time(0));
  int n;

  cout << "Количество испытаний: ";
  cin >> n;

  Point z;
  z.fun(n);

  system("pause");
  return 0;
}
```