МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Дипломна магістерська робота

на тему Експериментальне обґрунтування якості градієнтних методів оптимізації

Виконав: студент групи МгІТ-2-20
спеціальності
122 комп'ютерні науки
освітньої програми
комп'ютерні науки

Сергєєв Денис Дмитрович

Керівник к.т.н., доц. Володимир Яхно

Рецензент к.т.н., доц. Борис Шрамченко

Київ 2021

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

факультет мехатроніки та комп'ютерних технологій

кафедра комп'ютерних наук

Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

_____ Володимир Щербань

"____"_____2021 року

З А В Д А Н Н Я

НА ДИПЛОМНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Сергєєву Денису Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Експериментальне обґрунтування якості градієнтних методів оптимізації
Науковий керівник роботи Яхно Володимир. Михайлович, к. т .н., доц. затверджений наказом вищого навчального закладу від "04"жовтня 2021 року № 286**.**

2. Строк подання студентом роботи 11.12.2021р.

3. Вихідні дані до роботи Розробка кафедри інформаційних технологій проектування. Математичні методи дослідження операцій. Засоби проектування та побудови програмних засобів.

4. Зміст дипломної роботи (перелік питань, які потрібно розробити) : РОЗДІЛ 1 (Теоретичний. Постановка задачі і предмет дослідження); РОЗДІЛ 2 (обґрунтування інформаційних, математичних моделей, принципових

моделей даних та методів); РОЗДІЛ 3 (алгоритмічне і програмне забезпечення системи).

5. Консультанти розділів дипломної магістерської роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ1	Володимир Яхно, к.т.н., доц		
Розділ 2	Володимир Яхно, к.т.н., доц.		
Розділ3	Володимир Яхно, к.т.н., доц.		
Висновки	Володимир Яхно, к.т.н., доц		

6. Дата видачі завдання 10.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	10.01.2921	
2	Розділ 1 (Постановка задачі та методи дослідження)	10.10.2021	
3	Розділ 2 (Обґрунтування моделей та методів)	30.10.2021	
4	Розділ 3 (Алгоритмічне та програмне забезпечення)	10.11.2021	
5	Висновки	25.11.2021	
6	Оформлення дипломної магістерської роботи (чистовий варіант)	30.11.2021	
7	Здача дипломної магістерської роботи на кафедру для рецензування (за 14 днів до захисту)	4.12.2021	
8	Перевірка дипломної магістерської роботи на наявність ознак плагіату (за 10 днів до захисту)	8.12.2021	
9	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (з 7 днів до захисту)	11.12.2021р.	

Студент _____ Денис СЕРГЄЄВ
(підпис)

Науковий керівник роботи _____ **Володимир** Яхно
(підпис)

Директор НМЦУПФ _____ **Олена ГРИГОРЕВСЬКА**
(підпис)

АНОТАЦІЯ

Сергєєв Д .Д . Експериментальне обґрунтування якості
градієнтних методів оптимізації
– Рукопис.

Дипломна магістерська робота за спеціальністю 122 комп'ютерні науки та інформаційні технології – Київський національний університет технологій та дизайну, Київ, 2021 рік.

Розглянуті практичні аспекти реалізації широкого кола градієнтних методів мінімізації функцій. На прикладі еліптичних та не опуклих функцій проведено експериментальне дослідження основних параметрів збіжності модифікацій методу градієнтного спуску. Модифікації визначали вибір кроку та напрямку спуску, технології з розширенням просторів у створенні градієнтів та методи, що будують напрямки спуску з допомогою градієнтів. Ефективні методи враховують особливості двох послідовних напрямків спуску, які теоретично забезпечують прискорену збіжність, але це потребує експериментального підтвердження.

Існує велика кількість ефективних чисельних методів оптимізації опуклих негладких функцій, що використовують градієнт функції для визначення напрямку знаходження нової точки. Ці технології використовуються також як основа побудови багатьох інших чисельних методів оптимізації, що мають практичне застосування. Наприклад, навчання нейронних мереж. Методи цієї групи є основою не оптимізаційних алгоритмів для задач які часто можуть бути інтерпретовані як задачі оптимізації (методи штрафних функцій, рішення систем лінійних та нелінійних рівнянь та інше). Для цієї групи методів розрахунки виконуються відповідно до наступного рекурентного співвідношення

$x_{k+1} = x_k - \text{step}_{k1}\, v_{k1} -- \text{step}_{k2}\, v_{k2}$, $k = 0, 1 \ldots x_{k+1}$, $x_k \in E^n$, step_{k1} , $\text{step}_{k2} \in E^1$

Якісні характеристики обчислювальних алгоритмів, що будуються за цією схемою залежить від вибору напрямків v_k спуску та кроків step_{k1} , $\text{step}_{k2} \in E^1$. Можливість отримання результату також залежать від гладкості, опуклості та ступеню яровості функцій, що мінімізуються. У дипломній роботі розглянуті практичні проблеми, що пов'язані з можливостями застосування градієнтних методів. Ці можливості пов'язані з технологією обчислення важливих параметрів - напрямків $v_{k1,}$ v_{k2}, спуску та кроку. Запропонована програма надає можливість визначити залежність швидкості збіжності алгоритмів від вибору параметрів $v_{k1,}$ v_{k2}, $\in E^n$, step_{k1} , $\text{step}_{k2} \in E^1$ та властивостей функції, що мінімізується.

Ключові слова: дослідження операцій, методи оптимізації, моделі даних, автоматизована система

АННОТАЦИЯ

Сергеев Д.Д. Экспериментальное обоснование качества градиентных методов оптимизации
- Рукопись.

Дипломная магистерская работа по специальности 122 компьютерные науки и информационные технологии - Киевский национальный университет технологий и дизайна, Киев, в 2020 год.

Рассмотрены практические аспекты реализации градиентных методов безусловной минимизации. На примере эллиптических и не выпуклых функций проведено экспериментальное исследование основных параметров сходимости. модификаций метода. Модификации определяли выбор шага и направления спуска, технологии с расширением пространств в создании градиентов и методы, строящие направления спуска с помощью градиентов. Эффективные методы учитывают особенности двух последовательных направлений спуска, которые теоретически обеспечивают ускоренную сходимость, но это требует экспериментального подтверждения.

Существует множество эффективных методов оптимизации выпуклых негладких функций, использующих градиент функции для определения направления нахождения новой точки. Эти технологии используются также в качестве основы построения многих других многочисленных методов оптимизации, имеющих практическое применение. К примеру, обучение нейронных сетей. Методы этой группы являются основой не оптимизационных алгоритмов для задач, которые часто могут быть интерпретированы как задачи оптимизации (методы штрафных функций, решения систем линейных и нелинейных уравнений и др.). Для этой группы

методов расчеты выполняются согласно следующему рекуррентному соотношению

$$x_{k+1} = x_k - step_{k1}\, v_{k1} \text{ -- } step_{k2}\, v_{k2},\ k = 0, 1 \ldots\ x_{k+1}, x_k \in E^n,\ step_{k1}, step_{k2} \in E^1$$

Качественные характеристики вычислительных алгоритмов, строящихся по этой схеме, зависит от выбора направлений v_k спуска и шагов $step_{k1}$, $step_{k2} \in E^1$. Возможность получения результата также зависит от гладкости, выпуклости и степени овражности функций. В дипломной работе рассмотрены практические проблемы, связанные с возможностями применения градиентных методов. Эти возможности связаны с технологией вычисления важных параметров – направления v_k спуска и шага. Предложенная программа позволяет определить зависимость скорости сходимости алгоритмов от выбора параметров v_{k1}, v_{k2}, $\in E^n$, $step_{k1}$, $step_{k2} \in E^1$ и свойств функции.

Ключевые слова: исследование операций, управления запасами, модели данных, автоматизированная система

SUMMARY

Sergeev D.D. Experimental justification of quality gradient optimization methods -- Manuscript.

Master's thesis in the specialty 122 computer science and information technology - Kyiv National University of Technology and Design, Kyiv, 2020.

Practical aspects of realization of a wide range of gradient methods of function minimization are considered. An experimental study of the main parameters of the convergence of modifications of the gradient descent method was performed on the example of elliptical and nonconvex functions. Modifications determined the choice of step and direction of descent, technologies with the expansion of spaces in the creation of gradients and methods that build directions of descent with the help of gradients. Effective methods take into account the features of two consecutive descent directions, which theoretically provide accelerated convergence, but this requires experimental confirmation.

There are a large number of effective numerical methods for optimizing convex nonsmooth functions that use a gradient function to determine the direction of finding a new point. These technologies are also used as a basis for building many other numerical optimization methods that have practical applications. For example, learning neural networks. The methods of this group are the basis of non-optimization algorithms for problems that can often be interpreted as optimization problems (methods of penalty functions, solutions of systems of linear and nonlinear equations, etc.). For this group of methods, the calculations are performed according to the following recurrent ratio

$x_{k+1} = x_k - step_{k1}\, v_{k1} -- step_{k2}\, v_{k2}$, $k = 0, 1 \ldots$ x_{k+1} , $x_k \in E^n$, $step_{k1}$, $step_{k2} \in E^1$

Qualitative characteristics of computational algorithms built according to this scheme depend on the choice of descent directions v_k and $step_{k1}$, $step_{k2} \in E^1$. The possibility of obtaining the result also depends on the smoothness, convexity and degree of fierceness of the functions that are minimized. Thesis deals with practical problems related to the possibility of applying gradient methods. These capabilities are related to the technology of calculating important parameters - the direction vk descent and step. The proposed program provides an opportunity to determine the dependence of the rate of convergence of algorithms on the choice of parameters v_k and $step_{k1}$, $step_{k2} \in E^1$ and the properties of the function that is minimized.

Зміст

Вступ……..……11

Розділ 1. Задача та методи дослідження……..... 14

1.1. Постановка задачі……...14

1.2. Методи дослідження……..18

Висновки до розділу……...22

Розділ 2. Обґрунтування застосованих моделей та алгоритмів25

2.1. Моделі задачі…….……25

2.2. Порівняльний аналіз алгоритмів…….30

Висновки до розділу ……..40

Розділ 3. Програмне забезпечення……..41

3.1. Обґрунтування принципів програмної реалізації……...41

3.2. Керівництво користувача …….49

Висновки до розділу ……..55

Висновки…….56

Література…….. 57

Додатки…… 61

Вступ

Актуальність теми. Чисельні методи мінімізації є важливою складовою базових основ наукової, інженерної та проектної діяльності. Таких як навчання нейронних мереж та вирішення багатовимірних диференціальних рівнянь еліптичного типу - дуже важлива проблема чисельних методів. Побудова чисельних схем для таких рівнянь призводить до систем лінійних рівнянь дуже великої розмірності до методів аналізу яких відносяться методи якнайшвидшого спуску, мінімальних неув'язок, тощо. Вони дуже прості та стійкі. Але якісні характеристики обчислювальних алгоритмів, що будуються за цією схемою залежить від вибору напрямків v_k спуску та кроків $step_{k1}$, $step_{k2} \in E^1$. Можливість отримання результату також залежать від гладкості, опуклості та ступеню яровості функцій, що мінімізуються. Можливості застосування алгоритмів пов'язані з технологією обчислення важливих параметрів - напрямкіів v_{k2} ,v_k спуску та кроку. Тому є актуальною можливість визначити практичну залежність швидкості збіжності алгоритмів від вибору параметрів $v_{k1,}$ v_{k2}, $\in E^n$, $step_{k1}$, $step_{k2} \in E^1$.

Мета дослідження. Мета дослідження – є розробка рекомендацій до від вибору параметрів $v_{k1,}$ v_{k2}, $\in E^n$, $step_{k1}$, $step_{k2} \in E^1$ алгоритмів, що для отримання розв'язку задачі мінімізації будують послідовності

$x_1 ,\ldots, x_{k,} .x_{k+1}\ldots \in E^n$,

з допомогою співвідношення

$x_{k+1} = x_k - step_{k1}\, v_{k1} - step_{k2}\, v_{k2}$, $k = 0, 1 \ldots x_{k+1}$, $x_k \in E^n$, $step_{k1}$, $step_{k2} \in E^1$

Для досягнення мети - обґрунтований вибір параметрів $v_{k1,}$ v_{k2}, $\in E^n$, $step_{k1}$, $step_{k2} \in E^1$, необхідно розробити програмні засоби, які дозолять дослідити експериментально та порівняти ефективність вибору параметрів $v_{k1,}$ v_{k2}, $\in E^n$, $step_{k1}$, $step_{k2} \in E^1$.

Для досягнення цієї мети необхідно вирішити наступні проблеми:

- розробити формальні та об’єктні програмні моделі для програмних засобів які використані для отримання реальних параметрів, що характеризують складність задачі..
- для реалізації відповідних програмних засобів необхідно виконати аналіз та порівняння видів програмної архітектури сучасних та найбільш поширених. Обґрунтовано обрати зручну для реалізації сформульованої проблеми магістерської роботи архітектуру програмного забезпечення
- Обґрунтувати вибір та використати доцільні інструментальні засоби програмування з локалізацією до української мови.
- Розробити комплекс моделей для наочного відображення результатів – бажано для графічного підтвердження результатів

Завдання дослідження. По перше це розробка інформаційних, математичних та програмних моделей. Необхідним є повний набір моделей для задач дослідження. Програмна реалізація алгориттмів та моделей дозволить дослідити проаналізувати і підтвердити експериментально ефективність стратегій, що перевіряються. Бажаним є графічний інтерфейс, що демонструє результати і відповідає потребам дослідження.

Необхідними є звичайні стандартні вимоги, що є обов’язковими для всіх програмних засобів:

- мінімальні витрати використання та нескладність використання та розгортання ;
- інтуїтивна зрозумілість та практична наочність представлення результатів.

Об’єкт дослідження. Принципи побудови оптимізаційних алгоритмів за схемою

$$x_{k+1} = x_k - step_{k1}\, v_{k1} \text{ -- } step_{k2}\, v_{k2},\ k = 0, 1 \ldots\ x_{k+1}, x_k \in E^n,\ step_{k1}, step_{k2} \in E^1$$

Сучасні графічні методи згортки отриманої в обчисленнях інформації для представлення результатів. Методи і технологія розробки та проектування програмного забезпечення та інформаційних систем.

Предмет дослідження. Предметом дослідження є експериментальне дослідження швидкості збіжності алгоритмів, що для отримання розв'язку задачі мінімізації негладких опуклих функцій будують послідовності

$$x_1, \ldots, x_k, x_{k+1} \ldots \in E^n,$$

з допомогою співвідношення

$$x_{k+1} = x_k - step_{k1}\, v_{k1} - step_{k2}\, v_{k2},\ k = 0, 1 \ldots\ x_{k+1}, x_k \in E^n,\ step_{k1}, step_{k2} \in E^1$$

Послідовність

$$x_1, \ldots, x_k, x_{k+1} \ldots \in E^n,$$

повинна збігатись до розв'язку задачі мінімізації, що задана.

Питання дослідження пов'язані з алгоритмами, що відповідають релаксаційній схемі

$$x_{k+1} = x_k - step_{k1}\, v_{k1} - step_{k2}\, v_{k2}\ k = 0, 1. \ldots\ , f'(x_k) > f'(x_{k+1})$$

Теоретично якість цих алгоритмів визначають можливістю визначення наступних співвідношень

$$\| x^{k+1} - x^* \| \le g_k \| x^k - x^* \| \quad \text{або} \quad \| x^{k+1} - x^* \| \le g \| x^k - x^* \|^2 \quad x_k, \in E^n$$

x* рішення задачі. Залежність визначає можливі теоретичні класи швидкості збіжності алгоритмів.

Необхідно дослідити також можливі не релаксаційні алгоритми, вони не вимагають виконання $f'(x_k) > f'(x_{k+1})$ на кожній ітерації. Практична оцінка швидкість збіжності дозволить визначити реальне коло задач для аналізу. В наш час, ця проблема не має обґрунтованого теоретичного розв'язку.

Теоретична оцінка не визначає реальну якість алгоритмів. Часто не збігається з досягнутими практичними результатами. Найпростіший приклад наступний. Не складна задача з мінімізацією функції $f(x,y) = x^2 + y^2$. Для знаходження мінімуму цієї функції незалежно від початкової точки

необхідна лише одна ітерація методу Ньютона, методу спряжених градієнтів та методу найскорішого градієнтного спуску, для якого крок визначається мінімізацією функції вздовж напрямку антиградієнту. Для майже подібної функції $f(x,y) = x^2 + 10*y^2$ необхідна, як і раніше, одна ітерація методу Ньютона, але вже два кроку методу спряжених градієнтів. Кількість ітерацій найшвидшого градієнтного спуску, в залежності від необхідних вимог точності, наближається до нескінченності.

Методи дослідження. Основними методами дослідження, що застосовані і потребує сформульована задача є програмування і математичні методи дослідження операцій.

Практична цінність - дозволяє покращити якість виконання поширених та актуальних задач оптимізації.

Елементи наукової новизни. Результати дозволяють отримати обґрунтований набір вимог до параметрів значного класу алгоритмів. Автору роботи програми, що базуються на подібних принципах і мають наведені характеристики не відомі.

Практична значущість роботи. Чисельні методи мінімізації є важливою складовою базових основ наукової, інженерної та проектної діяльності. Таких як навчання нейронних мереж та вирішення багатовимірних диференціальних рівнянь еліптичного типу - дуже важлива проблема чисельних методів.

Апробація результатів роботи. Результати роботи програми перевірені на прикладах відповідних задач чисельної оптимізації.

Розділ 1. Задача та методи дослідження

Постановка задачі

В роботі експериментально досліджуються методи вирішення задач математичної оптимізації, що ґрунтуються на використанні градієнта функції. Основна мета – дослідити найпоширеніші ідеї, які так чи інакше пов'язані з цим методом та його різноманітними модифікаціями.

Оптимізаційні задачі математичного програмування можна формулювати так:

Необхідно визначити знайти мінімум (або максимум) функції

$$f(\boldsymbol{x}), \boldsymbol{x} \in E^n, \tag{1.1}$$

за виконання необхідних умов

$$f_i(\boldsymbol{x}) \; \Pi_i \; 0, \;\; i=1,\ldots,m, \tag{1.2}$$

де $\Pi_i \in \{\leq, =, \geq\}$.

У тому випадку, коли одна функція із тих що досліджується

$$f(\boldsymbol{x}), f_i(\boldsymbol{x}), \; i=1,\ldots,m,$$

є нелінійною, наведена задача є задачею нелінійного програмування.

Функцію, $f(\boldsymbol{x})$, для якої необхідно знайти мінімум (або максимум) називають цільовою функцією, область

$$D = \{\boldsymbol{x} \in E^n : f_i(\boldsymbol{x}) \; R_i \; 0, \;\; i=1,\ldots,m\} \text{ —}$$

областю визначення аргументів функції а довільний елемент **x** із D — допустимим планом задачі (1.1) нелінійного програмування.

Допустимий план $\boldsymbol{x}^* \in D$ (їх може бути багато) такий, що

$$\mathbf{x}^* = \arg\min_{x \in D} f(\mathbf{x})$$

або

$$\mathbf{x}^* = \arg\max_{x \in D} f(\mathbf{x})$$

є розв'язком задачі (1.1)−(1.2).

Задача максимізації $f(\boldsymbol{x})$ еквівалентна задачі подібній задачі мінімізації $-f(\boldsymbol{x})$, а нерівність $f_i(\boldsymbol{x}) \geq 0$ можна замінити $-f_i(\boldsymbol{x}) \leq 0$, рівність $f_i(\boldsymbol{x}) = 0$ можна замінити

$$f_i(\boldsymbol{x}) \leq 0 \quad \text{і} \quad -f_i(\boldsymbol{x}) \leq 0.$$

Тому надалі обмежитись лише задачею мінімізації. Важливою компонентою задачі з обмеженнями є задача безумовної оптимізації

$$D = E^n.$$

Надалі будемо досліджувати лише цю задачу. Для функції $f(\boldsymbol{x})$ потрібно знайти точку $argmin\ f(\boldsymbol{x})$, $\boldsymbol{x} \in E^n$,

Окремим випадком задачі нелінійної оптимізації програмування є задача квадратичного програмування, в якій обмеження є лінійними, а функція є сумою лінійної та квадратичної функції (квадратичної форми).

$$f(\boldsymbol{x}) = \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_i x_j \to \min$$

Як і загальному випадку розв'язання задач нелінійного програмування, визначення глобального екстремуму завдання квадратичного програмування немає ефективного обчислювального методу, якщо не відомо, що будь-який локальний екстремум є одночасно і глобальним. Оскільки завдання квадратичного програмування безліч допустимих рішень опукло, то, якщо цільова функція увігнута, будь-який локальний максимум є глобальним; якщо ж цільова функція – опукла, то будь-який локальний мінімум також і глобальний.

Функція f є сумою лінійної функції (яка є і опуклою, і увігнутою) і квадратичної форми. Якщо остання є увігнутою (опуклою), то завдання

відшукання максимуму (мінімуму) цільової функції можуть бути успішно вирішені. Питання про те, чи буде квадратична форма увігнутою або опуклою, залежить від того, чи є вона негативно-визначеною, позитивно-визначеною, позитивно – напив визначеною або взагалі невизначеною.

Зазвичай передбачається, що $f(\boldsymbol{x})$ є диференційованою і опуклою функцією. Це дозволяє дати певні гарантії успіху застосування градієнтного спуску. На практиці градієнтний спуск успішно застосовується навіть тоді, коли проблема не має жодної з перерахованих вище властивостей (приклад далі). Для розв'язання задачі пошуку $argmin\, f(\boldsymbol{x}), \boldsymbol{x} \in E^n$ існує альтернатива.

Ще в 17 столітті П'єром Ферма був придуманий критерій, який дозволяв вирішувати прості завдання оптимізації, а саме, якщо $\boldsymbol{x}^*$ точка мінімуму $f(\boldsymbol{x})$, то.

$\nabla f(\boldsymbol{x}^*)=0$

Цей критерій базується на лінійному наближенні

$$f(x) \approx f(x^*) + f'(x^*)(x - x^*).$$

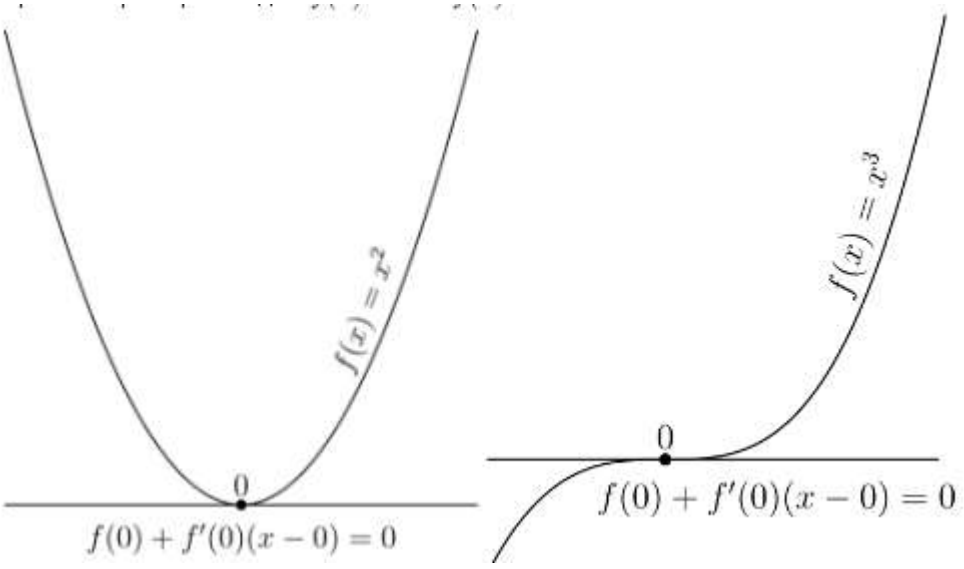


Рис 1.1

Чим ближче до $\boldsymbol{x}^*$ тим точніше це наближення. Також рівність градієнта нулю означає рівність всіх приватних похідних нулю, тому в багатовимірному випадку можна отримати цей критерій просто послідовно

застосувавши одновимірний критерій щодо кожної змінної окремо. Надалі ми будемо порівнювати алгоритми за ефективністю.

Частинні похідні $f(\boldsymbol{x})$ в точці $\boldsymbol{x0}$ визначаються як межі (якщо існують)

$$\frac{\partial f(\boldsymbol{x}^0)}{\partial x_j} = \lim_{h \to 0} \frac{f(\boldsymbol{x}^0 + h\,\boldsymbol{e}_j) - f(\boldsymbol{x}^0)}{h},$$

де $\boldsymbol{ej} = (0,...,0,1(1$ стоїть на j-у місці$),0,...,0)^T$ – *n-вимірний* одиничний вектор, що був транспонований у вектор-рядок.

Функція $f(\boldsymbol{x})$має назву *диференційовної* в точці $\boldsymbol{x}$, якщо вона визначена в деякому околі точки $\boldsymbol{x}$ і для $\boldsymbol{x}$ із цього околу $f(\boldsymbol{x})$ можна подати у вигляді

$$f(\boldsymbol{x}) = f(\boldsymbol{x}0) + \nabla mf(\boldsymbol{x}0)\,(\boldsymbol{x} - \boldsymbol{x}0) + o(||\boldsymbol{x} - \boldsymbol{x}0||).$$

Вектор

$$\nabla f(\boldsymbol{x}^0) = \left(\frac{\partial f(\boldsymbol{x}^0)}{\partial \boldsymbol{x}_1}, \frac{\partial f(\boldsymbol{x}^0)}{\partial \boldsymbol{x}_2}, ..., \frac{\partial f(\boldsymbol{x}^0)}{\partial \boldsymbol{x}_n} \right)^T$$

градієнт функції.

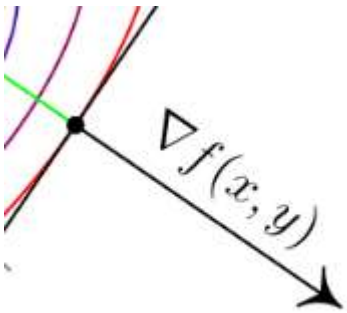


Рис 1.2

Як міру складності алгоритму найчастіше розглядають дві характеристики: час його роботи та обсяг використовуваної пам'яті, які виражаються як функції від розміру задачі. Оскільки час роботи алгоритму, як правило, не менше обсягу пам'яті, то надалі під складністю алгоритму, головним чином, ми розумітимемо його тимчасову складність (час роботи). Обчислювальні алгоритми, які ми вивчатимемо, вирішують деяке завдання,

виконуючи певну послідовність елементарних арифметичних та логічних операцій (складань, віднімань, творів, поділів та порівнянь). Тимчасова складність таких алгоритмів визначається як кількість елементарних операцій, що він виконує. Такий підхід до оцінки складності алгоритмів називають алгебраїчним. Алгебраїчний підхід ігнорує дискретність даних у пам'яті комп'ютера (там немає дійсних чисел, лише раціональні). У пам'яті комп'ютера під запис числа приділяється фіксована кількість бітів. Це обмежує розміри чисел, над якими арифметичні операції виконуються точно (без заокруглень) та з однаковою швидкістю. Якщо ж розміри чисел, над якими виконуються арифметичні операції, або результати цих операцій перевершують розміри комп'ютерної розрядної сітки, то для точного виконання арифметичних операцій потрібно використовувати бібліотеку алгоритмів для виконання арифметичних операцій з довгими (великими) числами

Використовуючи градієнтні методи, можна знайти рішення будь-якого завдання нелінійного програмування. Застосування цих методів у випадку дозволяє знайти точку локального екстремуму. Тому доцільно використовувати їх знаходження вирішення завдань опуклого програмування. Процес знаходження вирішення задачі за допомогою градієнтних методів полягає в тому, що починаючи з деякої точки здійснюється послідовний перехід до деяких інших точок доти, доки не буде знайдено прийнятне рішення вихідної задачі – точка, що задовольняє умовам зупинки обчислень. Градієнтні методи можуть бути поділені на дві групи.

До першої групи належать методи, при використанні яких досліджувані точки не виходять за межі області допустимих розв'язків задачі. В даному випадку найпоширенішим є метод Франка – Вульфа. До другої - методи, при використанні яких досліджувані точки можуть належати, так і не належати області допустимих рішень. Проте в результаті реалізації ітераційного процесу знаходиться точка області допустимих рішень, що визначає прийнятне рішення. Найчастіше використовуються метод штрафних

функцій та метод Ерроу – Гурвіца. При знаходженні рішення задачі градієнтними методами ітераційний процес триває доти, поки градієнт функції в черговій точці не стане рівним нулю або поки не виконається нерівність

Окрема технологія для випадку коли обмеження містять лише лінійні нерівності. Ця особливість є основою для заміни в околиці досліджуваної точки нелінійної цільової функції лінійної, у результаті рішення вихідної задачі зводиться до послідовного вирішення задач лінійного програмування.

Процес знаходження рішення починають із визначення точки, що належить області допустимих рішень. Обчислюють у цій точці градієнт функції :

$$\nabla f\left(X^{(k)}\right)=\left(\frac{\partial f\left(X^{(k)}\right)}{\partial x_1};\frac{\partial f\left(X^{(k)}\right)}{\partial x_2};\ldots;\frac{\partial f\left(X^{(k)}\right)}{\partial x_n}\right),$$

Та будують лінійну функцію, що базується на лінійному наближенні

$$F_k=\frac{\partial f\left(X^{(k)}\right)}{\partial x_1}x_1+\frac{\partial f\left(X^{(k)}\right)}{\partial x_2}x_2+\ldots+\frac{\partial f\left(X^{(k)}\right)}{\partial x_n}x_n.$$

Знаходять максимум функції при обмеженнях. Нехай вирішення цього завдання визначається точкою $X^{(k)}$. За нове допустиме вирішення $Z^{(k)}$ вихідного завдання приймають координати точки , які знаходять за формулами

$$X^{(k+1)}=X^{(k)}+\lambda_k\left(Z^{(k)}-X^{(k)}\right);$$

Замість того , щоб вирішувати завдання (1)-(3), знаходять максимум функції

$$F\left(x_1,x_2,\ldots,x_n\right)=f\left(x_1,x_2,\ldots,x_n\right)+H\left(x_1,x_2,\ldots,x_n\right),$$

де визначається системою обмежень і називається штрафною функцією Останню можна побудувати у різний спосіб. Найчастіше вона має вигляд.

Використовуючи штрафну функцію, послідовно переходять від однієї точки до іншої, доки не отримають прийнятне рішення.

Модифікації градієнтного спуску використовують співвідношення

$x_{k+1} = x_k - step_{k1}\, v_{k1}$ -- $step_{k2}\, v_{k2}$, $k = 0, 1 \ldots x_{k+1}$, $x_k \in E^n$, $step_{k1}$, $step_{k2} \in E^1$

в якому напрямки v_{k1} , v_{k2} будують з допомогою градієнту.

Останнє доданок характеризує цю «інерційність», алгоритм щокроку намагається рухатися проти градієнта, але за цьому по інерції частково рухається у тому напрямі, як і попередньої ітерації. Такі методи мають дві важливі властивості: Вони мало ускладнюють простий градієнтний спуск в обчислювальному плані. При акуратному підборі такі методи набагато швидше, ніж звичайний градієнтний спуск навіть з оптимально підібраним кроком. Вони мало ускладнюють простий градієнтний спуск в обчислювальному плані. При акуратному підборі такі методи набагато швидше, ніж звичайний градієнтний спуск навіть з оптимально підібраним кроком.

Скалярні - множники перед напрямками v_{k1} , v_{k2} , що будують з допомогою градієнту називають довжиною кроку. Очевидно, вони повинні бути визначеними як позитивними. Існує багато модифікацій цих методів, які відрізняються один від одного стратегією обчислення довжини кроку на кожній (к-тій) ітерації.

Найбільш важливі з них розглянуті в цій роботі.

1.Постійний крок. $step_{k1}\, step_{k2} = h > 0$, Для таких функцій, градієнт яких задовольняє умові Ліпшиця це можливо. Умова Ліпшиця - існує таке число L що нерівність

$$\| f'(x) - f'(y) \| < L \| x - y \|$$

має місце для всіх $f'(x)$, $f'(y)$, $x, y \in E^n$. В цьому випадку найкращим є максимальний крок $h = 1/L$[1].

2. З дробленням кроку на кожній ітерації. Крок, на кожній ітерації, зменшується с заданим постійним коефіцієнтом якщо не досягається зменшення функції за формулою $x_{k+1} = x_k - step_{k1}\, v_{k1}$

3. Найшвидший спуск (або метод повної релаксації)

$h_k = \arg\min\ x_k - step_{k1}\, v_{k1}$, мінімум визначається по $- step_{k1} > 0$.

4. З регулюванням кроку. Крок, на кожній ітерації, с заданим постійним коефіцієнтом якщо досягається зменшення функції за формулою $x_{k+1} = x_k - step_{k1}\, v_{k1}$

5. Рандомізований крок. Параметри $step_{k1}$ $step_{k2}$ обираються випадково

Збіжність послідовності

$$x_1, \ldots, x_k, x_{k+1} \ldots \in E^n,$$

гарантується якщо крок знаходиться відповідно до правила Голдштейна - Армійо: h такий, що мають місце нерівності

$x_{k+1} = x_k - hf'(x_k)$,

$\alpha^*(f'(x_k), x_k - x_{k+1}) \leq f(x_k) - f(x_{k+1})$,

$\beta^*(f'(x_k), x_k - x_{k+1}) \geq f(x_k) - f(x_{k+1})$,

де задані наперед α, β, $0<\alpha<\beta<1$, — деякі фіксовані параметри.

В останньому випадку якість алгоритму суттєво залежить від вибору фіксованих значень α, β.

Метод сполучених градієнтів. Цей спосіб пошуку абсолютного екстремуму поєднує у собі поняття градієнта цільової функції та пов'язаних напрямів.

Висновки 1.1

Розділ посвячений аналізу характеристик придатності алгоритмів градієнтної оптимізації для задач мінімізації. Досліджується вплив параметрів цих методів для функцій з ярами, або негладких функцій. Методи цієї групи є основою не оптимізаційних алгоритмів для задач які часто можуть бути інтерпретовані як задачі оптимізації (методи штрафних функцій, рішення систем лінійних та нелінійних рівнянь та інше). Для цієї групи методів розрахунки виконуються відповідно до наступного рекурентного співвідношення

$x_{k+1} = x_k - \text{step}_{k1}\, v_{k1} -- \text{step}_{k2}\, v_{k2}$, $k = 0, 1 \ldots x_{k+1}$, $x_k \in E^n$, step_{k1} , $\text{step}_{k2} \in E^1$

Якісні характеристики обчислювальних алгоритмів, що будуються за цією схемою залежить від вибору напрямків v_k спуску та кроків step_{k1} , $\text{step}_{k2} \in E^1$. Можливість отримання результату також залежать від гладкості, опуклості та ступеню яровості функцій, що мінімізуються. Якісні характеристики обчислювальних алгоритмів теоретично визначають швидкість збіжності.

У розділі роботі розглянуті практичні проблеми, що пов'язані з можливостями застосування градієнтних методів. Ці можливості пов'язані з технологією обчислення важливих параметрів - напрямку $v_{k1,}$ v_{k2}, спуску та кроку. Необхідна можливість визначити залежність швидкості збіжності алгоритмів від вибору параметрів $v_{k1,}$ v_{k2}, $\in E^n$, step_{k1} , $\text{step}_{k2} \in E^1$ та властивостей функції, що мінімізується. Якісні характеристики обчислювальних алгоритмів, що будуються за цією схемою залежить від вибору напрямків v_k спуску та кроків step_{k1} , $\text{step}_{k2} \in E^1$. Можливість отримання результату також залежать від гладкості, опуклості та ступеню яровості функцій, що мінімізуються. У дипломній роботі розглянуті практичні проблеми, що пов'язані з можливостями застосування градієнтних методів. Ці можливості пов'язані з технологією обчислення важливих параметрів - напрямків $v_{k1,}$ v_{k2}, спуску та кроку. Запропонована програма надає можливість визначити залежність швидкості збіжності алгоритмів від вибору параметрів $v_{k1,}$ v_{k2}, $\in E^n$, step_{k1} , $\text{step}_{k2} \in E^1$ та властивостей функції, що мінімізується.

Питання дослідження пов'язані з алгоритмами, що відповідають релаксаційній схемі

$x_{k+1} = x_k - step_{k1}\, v_{k1} -- step_{k2}\, v_{k2}$ $k = 0, 1. \ldots$, $f'(x_k) > f'(x_{k+1})$

Теоретично якість цих алгоритмів визначають можливістю визначення наступних співвідношень

$\| x^{k+1} - x^* \| <= g_k \| x^k - x^* \|$ або $\| x^{k+1} - x^* \| <= g \| x^k - x^* \|^2$ x_k , $\in E^n$

x* рішення задачі. Залежність визначає можливі теоретичні класи швидкості збіжності алгоритмів.

Необхідно дослідити також можливі не релаксаційні алгоритми, вони не вимагають виконання $f'(x_k) > f'(x_{k+1})$ на кожній ітерації. Практична, отримана експериментально, оцінка швидкість збіжності дозволить визначити реальне коло задач для аналізу. В наш час, ця проблема не має обґрунтованого теоретичного розв'язку.

Для розв'язання задачі необхідні наочні результати експериментів, наприклад наступний рисунок є результатом програми. Він наочно дозволяє порівняти стратегію, що базується на мінімізації функції цілі вздовж градієнта з іншою стратегією мінімізації норми градієнта цієї функції.

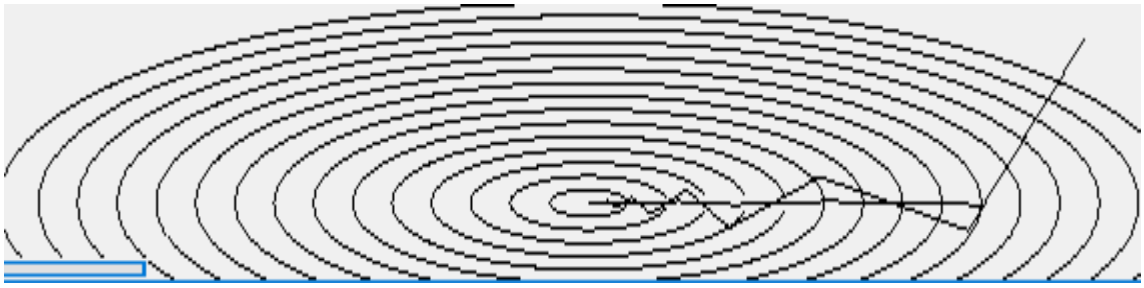


Рис 1.4.

Розділ 2. Обґрунтування моделей та методів

Найбільш важливі та поширені варіанти методів мінімізації досліджені експериментально в цій роботі. Для тестування методів використовувалось дві технології. Перша базувалась на наочному графічному зображенні траєкторії спуску для еліптичних та не опуклих функцій. В якості не опуклих функцій використовувалась функція Розенброка (англ. Rosenbrock function, Rosenbrock's valley, Rosenbrock's banana function) - не опукла функція , що використовується для оцінки продуктивності алгоритмів оптимізації , запропонована Ховардом Розенброком (англ.) в 1960 [5] . Вважається, що пошук глобального мінімуму для цієї функції є нетривіальним завданням.

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2.$$

В програмному коді це має вигляд

```
ссылок: 30
private  double f(double x, double y)
{
    //result:=100*sqr(y-sqr(x))+sqr(1-x)).
    double xn, yn;
   if(radioButton1.Checked) return Math.Pow(x, 2) + 5 * (y) * (y);
   if (radioButton2.Checked)
   {
       xn = x * Math.Cos(Math.PI / 4) - y * Math.Sin(Math.PI / 4);
       yn = x * Math.Sin(Math.PI / 4) + y * Math.Cos(Math.PI / 4);
       return Math.Pow(xn, 2) + 5 * (yn) * (yn);
   }
   if (radioButton3.Checked)
   {
       return 100*Math.Pow((y-x*x), 2) +  (1-x) * (1-x);
   }
   return Math.Pow(x, 2) + 5 * (y) * (y);
}
ссылок: 4
```

Графік функції наведений на малюнку нижче

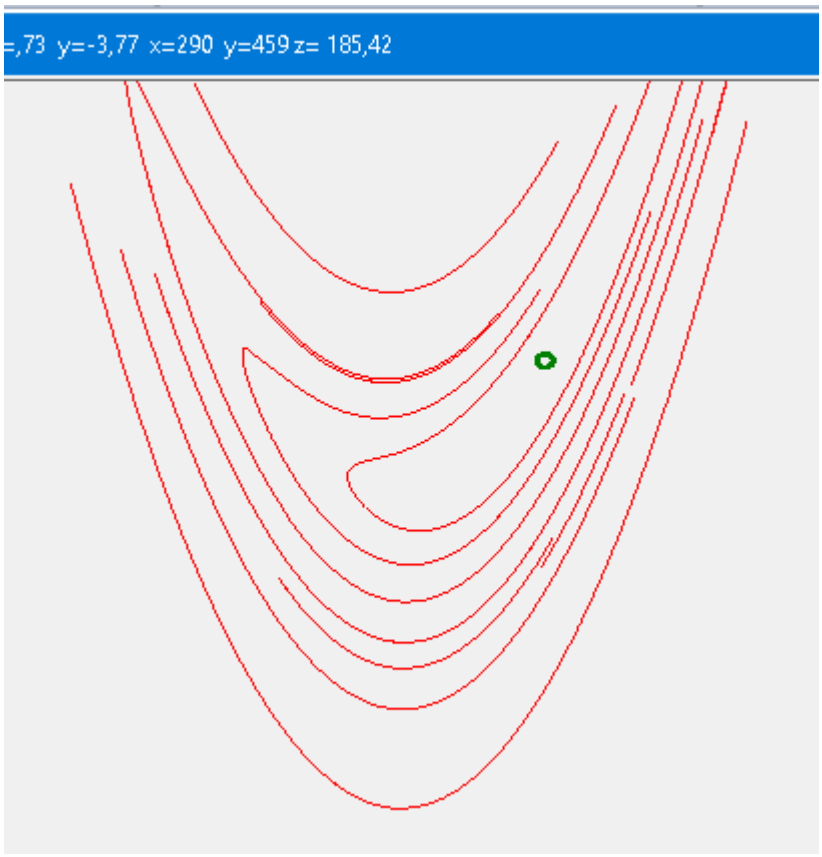


Рис 2.1 Функція Розенброка

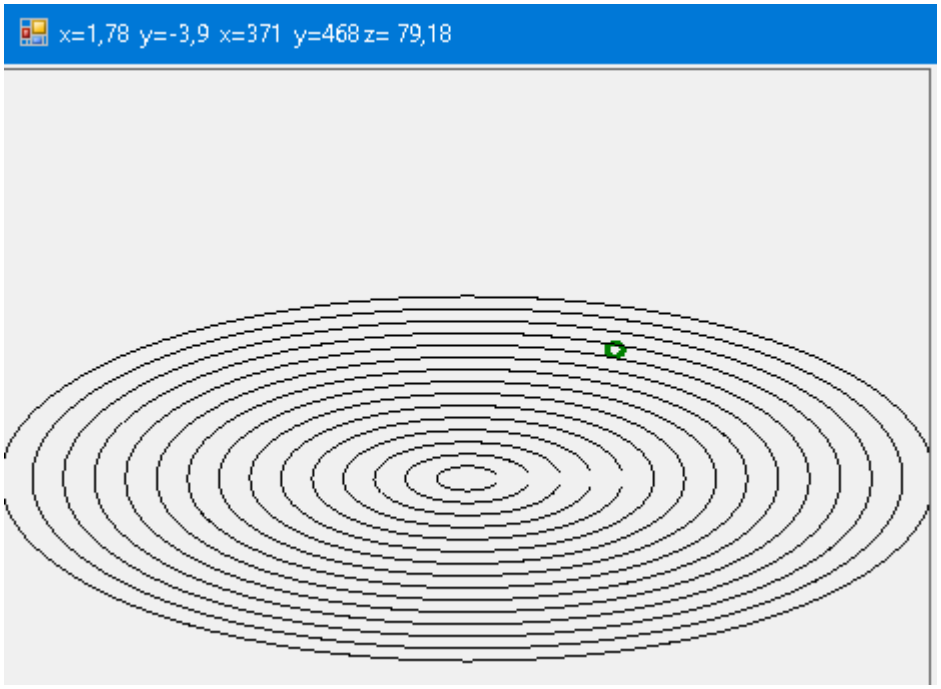


Рис 2.2 Еліптична функція.

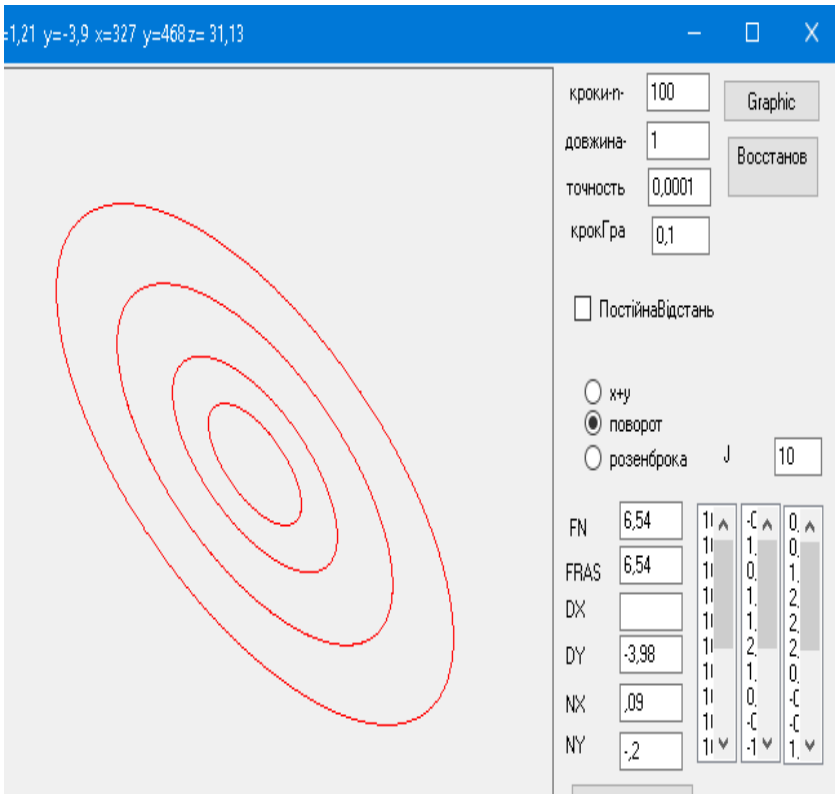


Рис 2.3 Повернена еліптична функція.

Для не графічного тестування методів використовувався наступний підхід: випадковим чином вибирався деякий n – вимірний вектор і з нього обчислювалася

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in D} f(\mathbf{x})$$

Мірою помилки у разі може бути норма різниці векторів. Випробування проводилися M різних початкових наближень; зазвичай M = 30. Компоненти вектора початкового наближення вибиралися випадково з відрізка [-2, 2]. Потім отримані результати усереднювалися. Серії випробувань проводилися для N= 12, 17, 24, 34, ... У цій послідовності відношення наступного числа до попереднього близько до кореня з 2, що в логарифмічному масштабі відповідає рівномірному кроку.

Залежність швидкості збіжності від параметра для тестових матриць розміром N = 30 демонструється на Рис. 2.4 де представлена залежність логарифму кількості ітерацій K від коефіцієнта релаксації для алгоритму з постійним кроком.

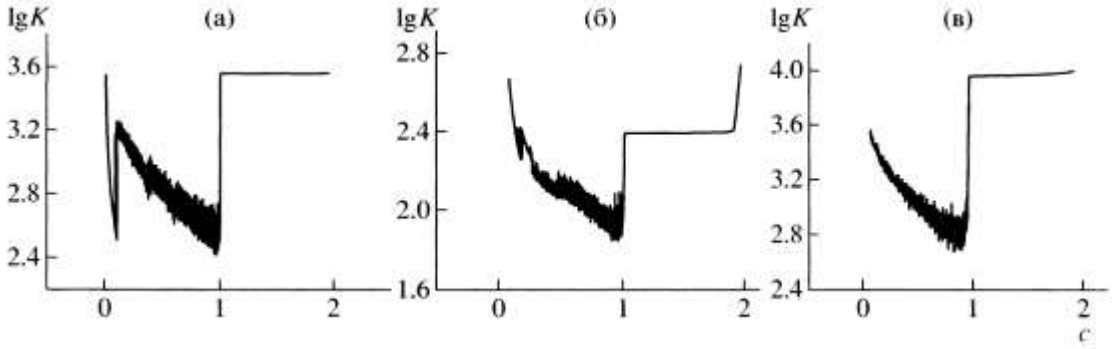


Рис. 2.4 - залежність логарифму кількості ітерацій К . $step_{k1}$

Параметри визнались з допомогою лічильників продуктивності Лічильники продуктивності – це окремі числові властивості у категоріях. Зазвичай прийнято вказувати категорію і назва лічильника продуктивності, розділяючи їх зворотним флішем, наприклад, (Процес \ Байт виняткового користування). Лічильники продуктивності можуть мати різні типи, включаючи прості числові значення (Процес \ Лічильник потоків)), швидкості проходження подій (Черга друку \ Друкованих байт/СЕК)), відсотки та середні значення (Показники роботи служб 3.0.0.0 \ Тривалість виклику)).

Примірники категорій лічильників продуктивності (входження) використовуються з метою створення різних наборів лічильників для різних екземплярів компонентів системи. Наприклад, у системі може бути кілька процесорів, тому для кожного з них є свій екземпляр у категорії Processor Information (Відомості про процесор), а також загальний екземпляр _Total). Одні категорії лічильників продуктивності можуть мати кілька екземплярів (таких більшість), інші – єдиний екземпляр (наприклад, категорія Memory (Пам'ять).

Графічна ілюстрація.

1.Постійний крок. $step_{k1}$ $step_{k2}$ = $h>0$, Для таких функцій, градієнт яких задовольняє умові Ліпшиця це можливо. Умова Ліпшиця - існує таке число L що нерівність

$$\|f'(x) - f'(y)\| < L\|x - y\|$$

має місце для всіх $f'(x)$, $f'(y)$, $x, y \in E^n$. В цьому випадку найкращим є максимальний крок $h = 1/L$[1]. Або для коефіцієнта релаксації К $h = К/L$

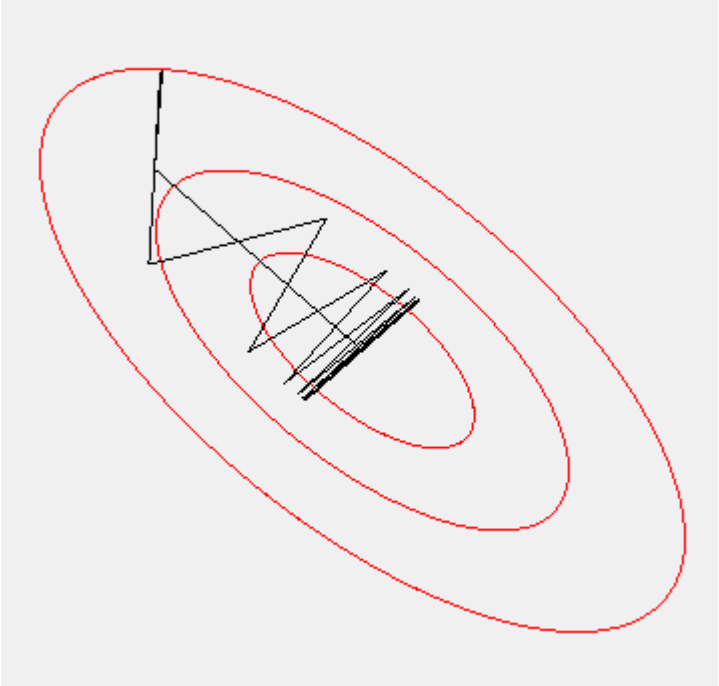


Рис. 2.5. Постійний крок з максимальним коефіцієнтом релаксації. (Повернена еліптична функція)

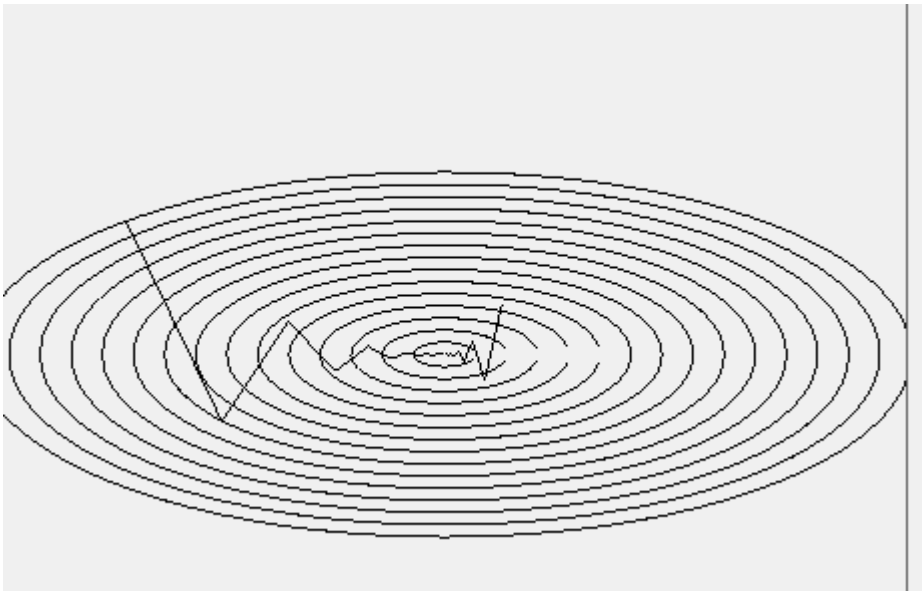


Рис. 2.5. Постійний крок з максимальним коефіцієнтом релаксації. (еліптична функція)

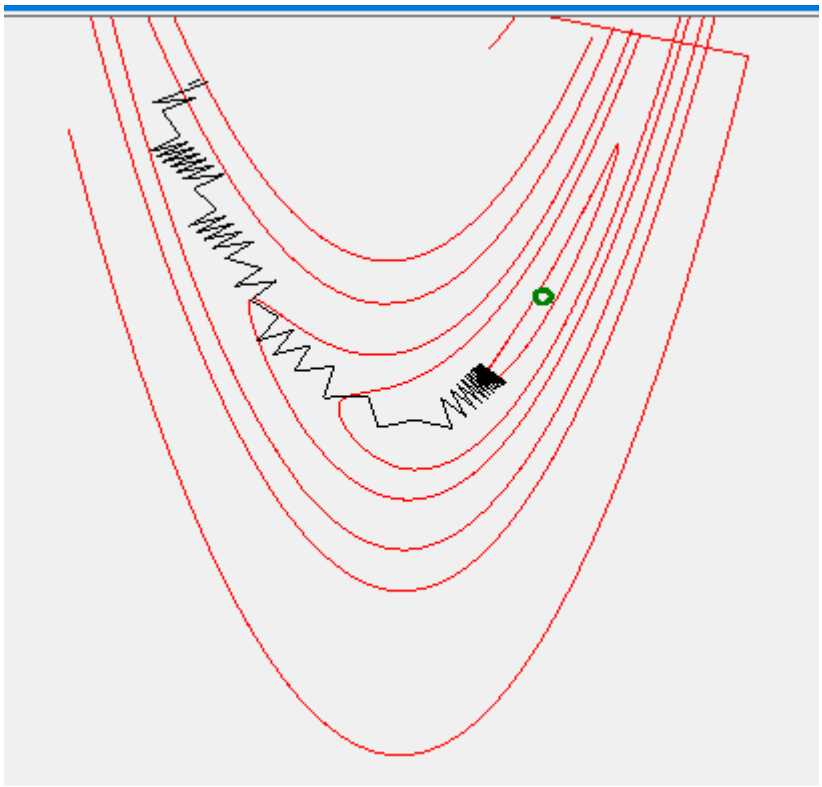


Рис. 2.6. Постійна відстань між точками з рандомізацією

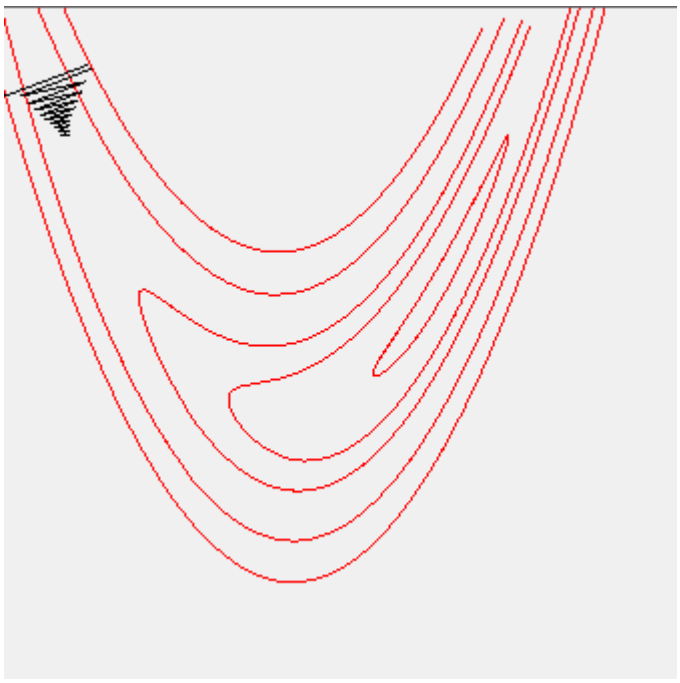


Рис. 2.7. Постійний крок

2. З дробленням кроку на кожній ітерації. Крок, на кожній ітерації, зменшується с заданим постійним коефіцієнтом якщо не досягається зменшення функції за формулою $x_{k+1} = x_k - step_{k1}\, v_{k1}$

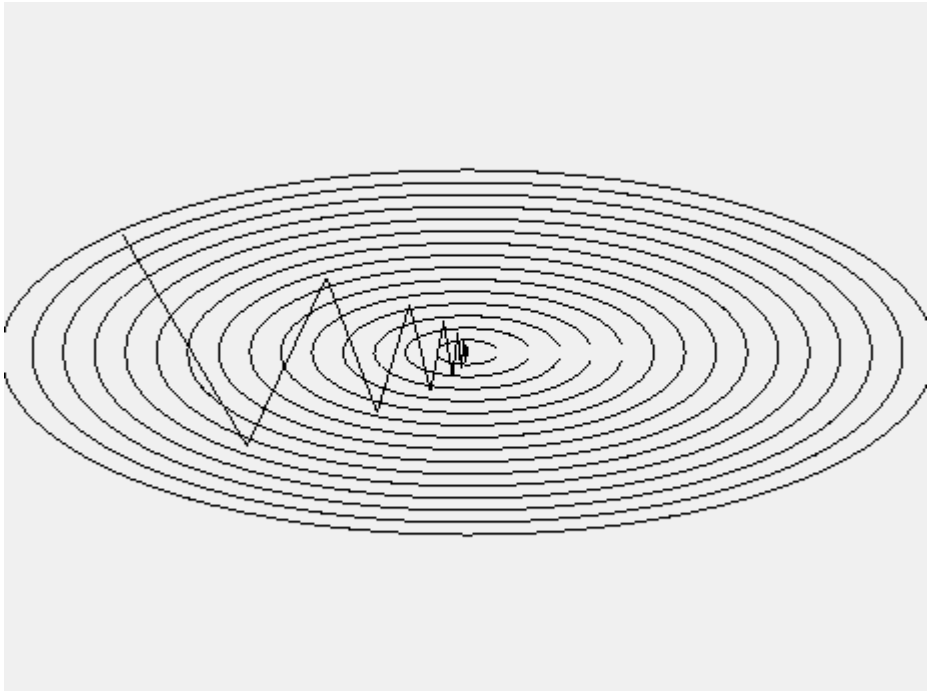


Рис. 2.9. З дробленням кроку на кожній ітерації. (еліптична функція)

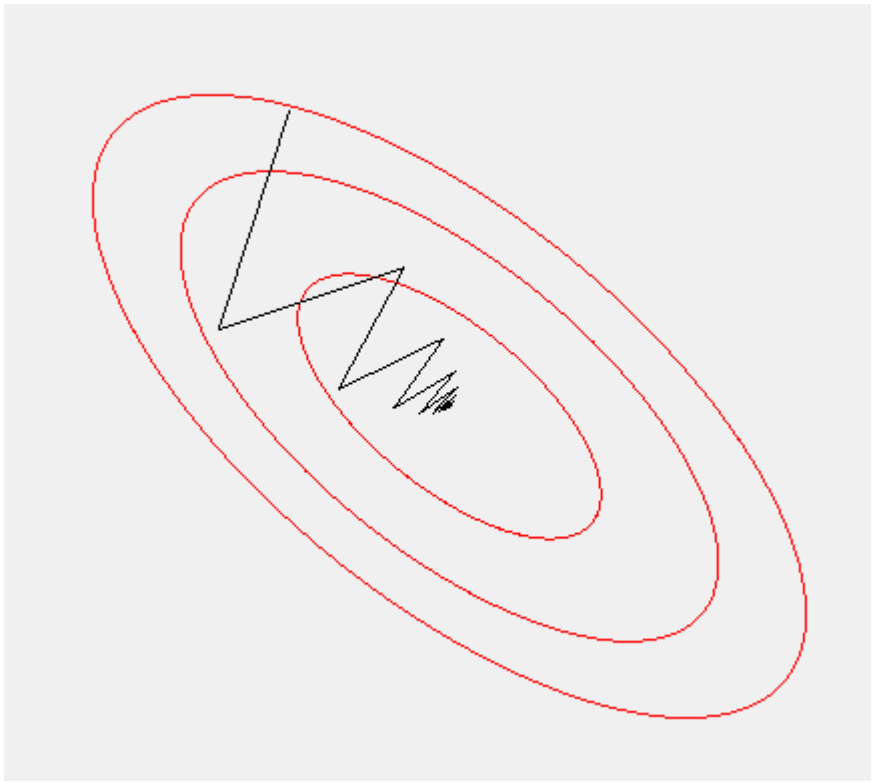


Рис. 2.9. З дробленням кроку на кожній ітерації. (повернена еліптична функція)

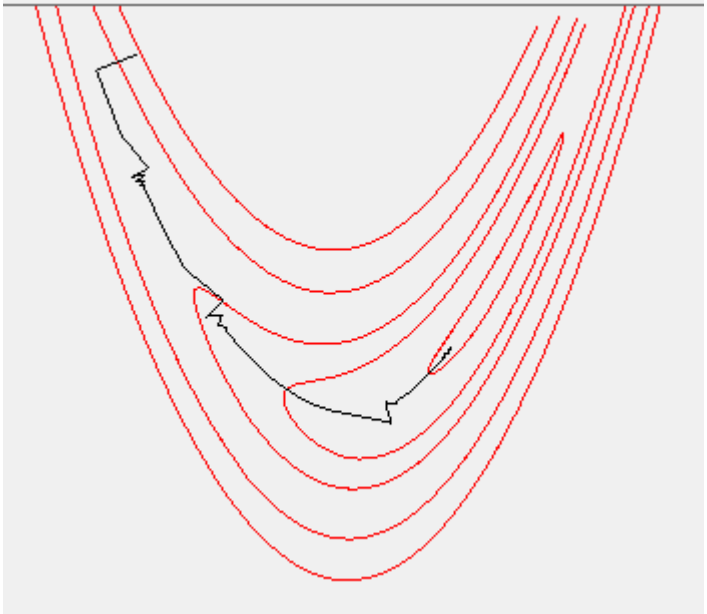
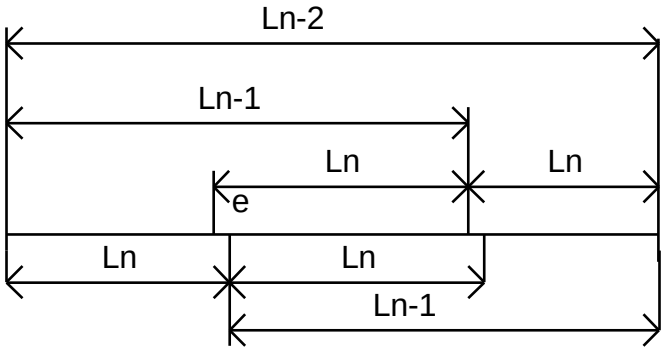


Рис. 2.10. З дробленням кроку на кожній ітерації та рестартами.

3. Найшвидший спуск (або метод повної релаксації)

h_k =arg min $x_k - step_{k1}\, v_{k1}$, мінімум визначається по $- step_{k1} > 0$.

Мінімум знаходився методом Кіфера.



Діаграма зміни інтервалів методом Кіфера.

Для того щоб приступити до обчислення відповідно до цього алгоритму, необхідно знати координати першої точки в середині початкового інтервалу невизначеності. Ця точка знаходиться на відстані L_2 від крайньої крапки сегменту.

$L_2 = F_{n-2} L_n - e F_{n-3} = (F_{n-1}/F_n) L_1 + e (-1)^n/F_n$.

Цей алгоритм дозволяє обчислити мінімум з максимальною точністю.

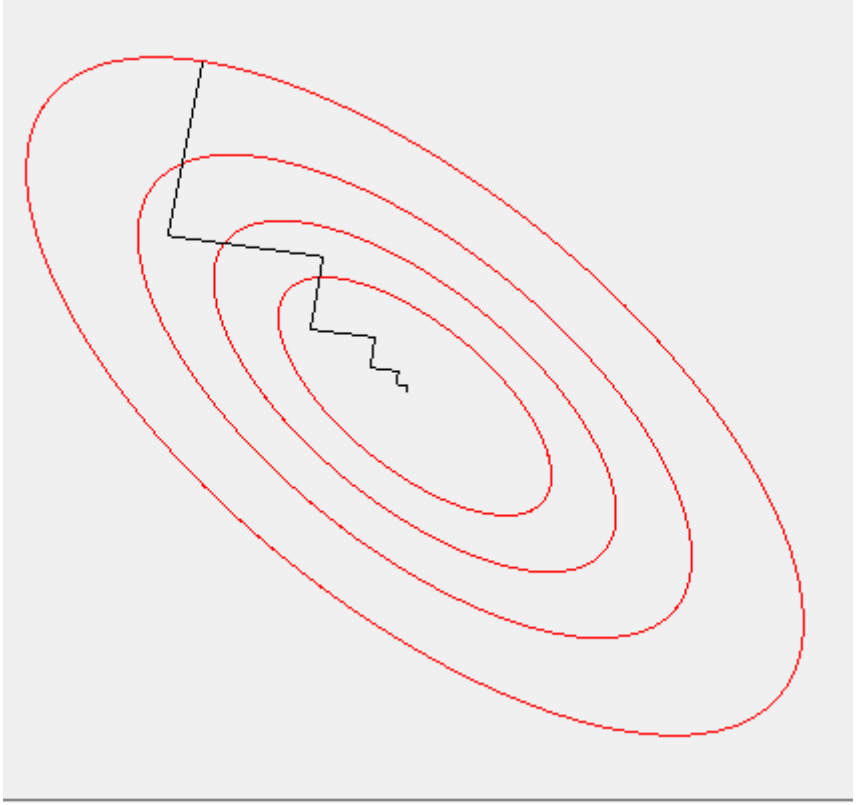


Рис. 2.11. Найшвидший спуск (повернена еліптична функція)

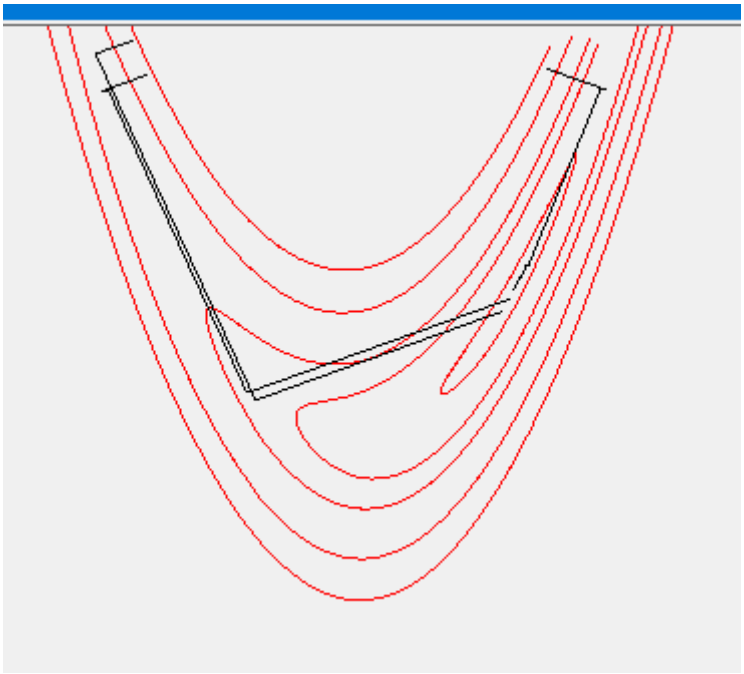


Рис. 2.11. Найшвидший спуск (функція Розенброка)

4. З регулюванням кроку. Крок, на кожній ітерації, збільшується с заданим постійним коефіцієнтом якщо досягається зменшення функції за формулою $x_{k+1} = x_k - step_{k1}\, v_{k1}$

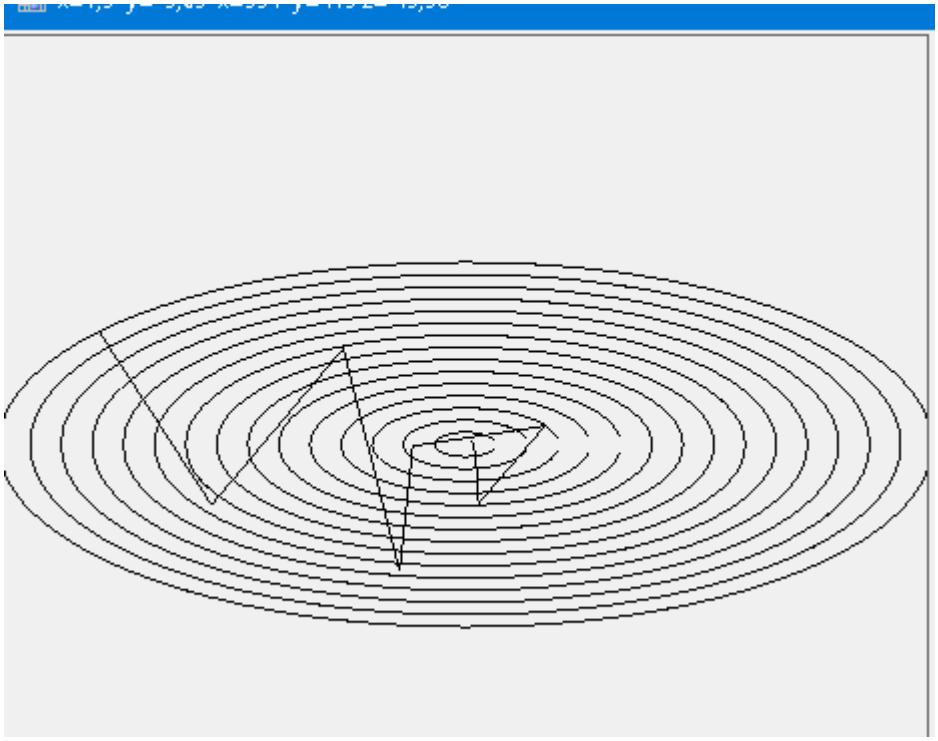


Рис. 2.12. З регулюванням кроку

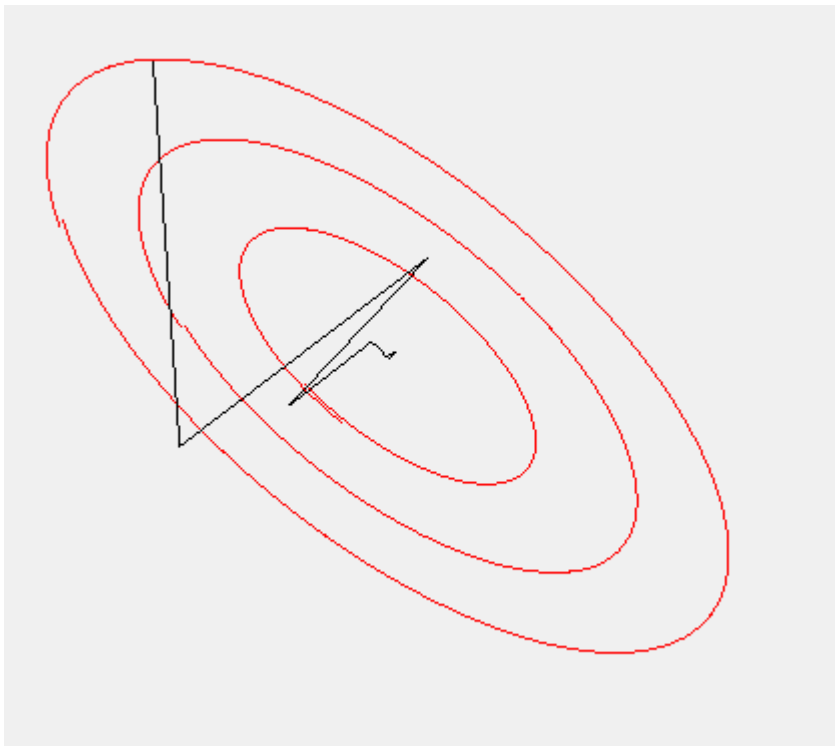


Рис. 2.13. З регулюванням кроку

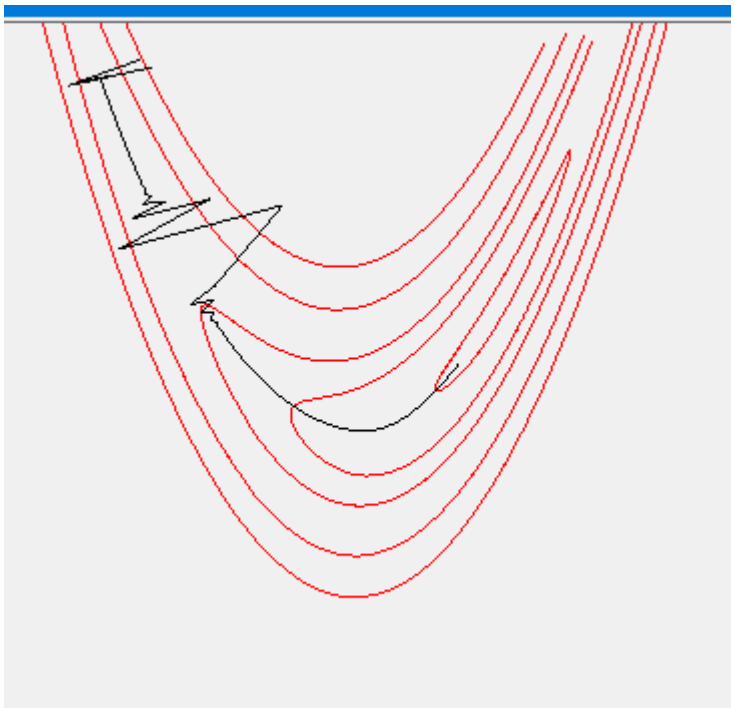


Рис 2.13. З регулюванням кроку

5. Рандомізований крок. Параметри $step_{k1}$ $step_{k2}$ обираються випадково Збіжність послідовності

$$x_1, \ldots, x_k, x_{k+1} \ldots \in E^n,$$

гарантується якщо крок знаходиться відповідно до правила Голдштейна - Армійо: h такий, що мають місце нерівності

$$x_{k+1} = x_k - hf'(x_k),$$

$$\alpha^*(f'(x_k), x_k - x_{k+1}) \leq f(x_k) - f(x_{k+1}),$$

$$\beta^*(f'(x_k), x_k - x_{k+1}) \geq f(x_k) - f(x_{k+1}),$$

де задані наперед α, β, $0<\alpha<\beta<1$, — деякі фіксовані параметри.

В останньому випадку якість алгоритму суттєво залежить від вибору фіксованих значень α, β.

6. Метод сполучених градієнтів.

Визначення сполученості формулюється наступним чином: два вектори x і y називають A - сполученими (або сполученими по відношенню до матриці A) або A-ортогональними, якщо скалярний добуток x і A * y дорівнює нулю, тобто: $x^T Ay=0$. Сполученість можна вважати узагальненням поняття ортогональності. Дійсно, коли матриця A – одинична матриця,

відповідно до рівності 4, вектори x та y – ортогональні. Яким чином обчислювати пов'язані напрямки. Один із можливих способів обчислювати пов'язані напрямки – використовувати методи лінійної алгебри, зокрема, процес ортогоналізації Грама-Шмідта. Але для цього необхідно знати матрицю A, тому для більшості завдань (наприклад, навчання багатошарових нейромереж) цей метод годиться. Існують інші, ітеративні способи обчислення сполученого напрямку, найвідоміший – формула Флетчера-Рівса. Відповідно до методу Флетчера-Рівса напрямки спуску $C_к$ на кожній ітерації обчислюються за формулами:

$$C_к = -f'(X_к) + в_{к-1} C_{к-1}, \; к > 1,$$

$$C_0 = -f'(X_0)$$

Метод Флетчера - вважається одним із ефективних методів вирішення завдань великої розмірності. Має властивості збіжності ньютонівських і квазіньтонівських алгоритмів для сильно опуклих функцій, що тричі диференціюються, але не вимагає обчислення матриці других похідних і вирішення систем рівнянь на кожній ітерації. Мінімізує квадратичну функцію за n кроків. Зручний для завдань із слабко заповненими матрицями.

Метод сполучених градієнтів або Метод Флетчера - Рівса побудований відповідно до загальної схеми методів спуску

$$x_{k+1} = x_k - step_{k1} v_{k1} -- step_{k2} v_{k2},$$

і використовує для одновимірного пошуку послідовність напрямків, кожен з яких є лінійною комбінацією антиградієнта в поточній точці та напрямку попереднього спуску. Напрямки кожної пари ітерацій є пов'язаними для модельної квадратичної функції з тими самими градієнтами. Найважливішою важливою особливістю алгоритму є необхідність мінімізації функції мети вздовж спуску на кожній ітерації.

Для обґрунтування нагадаємо загальну схему методів спуску

$$x_{k+1} = x_k - step_{k1} v_{k1}$$

де напрямок спуску утворює гострий кут з градієнтом у точці $X_к$, або v_{k1} * grad<0.

Якщо спуск найшвидший, то добуток спуску на градієнт у точці *Хк,* дорівнює 0. Ці два напрямки є ортогональними.

Відповідно до методу Флетчера-Рівса напрямки спуску *Ск* на кожній ітерації обчислюються за формулами

$$C_к = -f'(X_к) + в_{к-1} C_{к-1}, \quad к > 1,$$

$$C_0 = -f'(X_0)$$

Величини $в_{к-1}$ вибираються так щоб напрямки $C_к$, $C_{к-1}$ були $f''(Хк)$ - сполученими і, отже, мало місце співвідношення:

$$Ск\, f''(X_к)\, C_{к-1} = 0 = (-f'(X_к))^t f''(Хк)\, C_{k-1} + в_{к-1}\, C_{к-1}\, f''(Хк)\, C_{к-1.}$$

З останнього рівняння

$$в_{к-1} = (-f'(X_к))^t f''(Хк)\, C_{k-1} / C_{к-1}\, f''(X_к)\, C_{к-1.}$$

Точка $Хк+1$ визначається внаслідок мінімізації функції $f(X_к + r_к C_к)$ по r .

Обчислення за формулою можна суттєво спростити, якщо для матриці других похідних скористатися деякою її апроксимацією.

Розглянемо розкладання градієнта в ряд Тейлора на околиці точки *Хк* за ступенями компонент вектора $r_к C_к$

$$f'(Хк - r_{к-1} C_{к-1}) = f'(Хк) - f''(Хк)\,(r_{к-1} Ск\text{-}1)$$

Якщо функція f (X) є квадратичною, розкладання функції містить лише два члени, які наведені в останньому рівнянні. Для функцій загального виду остання формула є моделлю, точність якої зростає в міру наближення до точки екстремуму. З останньої формули

$$f''(Хк)\, Ск\text{-}1 = (f'(Хк) - f'(Хк - r_{к-1}\, Ск\text{-}1))/ r_{к-1} = (f'(Хк) - f'(Хк\text{-}1))/ r_{к-1}$$

Якщо підставити значення $f''(Хк)\, Ск\text{-}1$ з останньої формули рівняння, то отримаємо просту формулу для обчислення $в_{к-1}$ не містить матриць других похідних

$$в_{к-1} = (f'(Хк))^t (f'(Хк) - f'(Хк\text{-}1))/ Ск\text{-}1\, (f'(Хк) - f'(Хк\text{-}1)).$$

Коефіцієнти $r_{к-1}$ скорочуються. Також можна показати, що з ортогональності векторів $C_{к-1}$ і f' ($X_к$) (ця ортогональність є наслідком того, що Хк отримана в результаті мінімізації функції вздовж напрямку $C_{к-1}$), після розкриття дужок та відкидання творів, які дорівнюють нулю можна отримати

$в_{к-1} = - (f'(Хк))^t f'(Хк)/Ск-1 f'(Хк-1).$

Це рівняння можна спростити і зменшити пам'ять необхідну для обчислень

Якщо рівняння переписати для *Ск-1*

$Ск-1 = -f' (Хк-1) + вк-2\ Ск-2,$

помножити ліворуч і праворуч на $f' (Хк-1)$, то, після відбиття членів рівних нулю (ортогональні вектори Ск-1 і -f' (Хк-1)) отримаємо

$Ск-1\ f'(Хк-1) = -f'(Хк-1)\ f'(Хк-1).$

З цього рівняння для *Ck* подібним чином можна довести ортогональність градієнтів - $Ск\ f'(Хк-1) = -f'(Хк)\ f'(Хк-1)$. Ця рівність можлива, якщо $Ск = - f'(Хк-1)$, що не вірно, або якщо всі вектори попарно ортогональні. Отже $f'(Хк)\ f'(Хк-1) = 0$.

Остаточний варіант

$в_{к-1} = (f'(Хк))tf'(Хк)/(f'(Хк-1)\ f'(Хк-1).$

Тепер вид процедури обчислень наступний

1. На першій ітерації

$C_1 = - grad (X)$

2 На к -му кроці розв'язується задача одномірної мінімізації по r функції $f(x+ r\ C_к)$. І обчислюємо похідну у новій точці. Якщо похідна близька 0, то отримано розв'язання задачі

інакше визначаємо

$Ск+1 = - f' (Хк) + Ск ((f' (Хк+1))^t\ f' (Хк+1)/ (f' (Хк)\ f' (Хк1)).$

І повторимо обчислення.

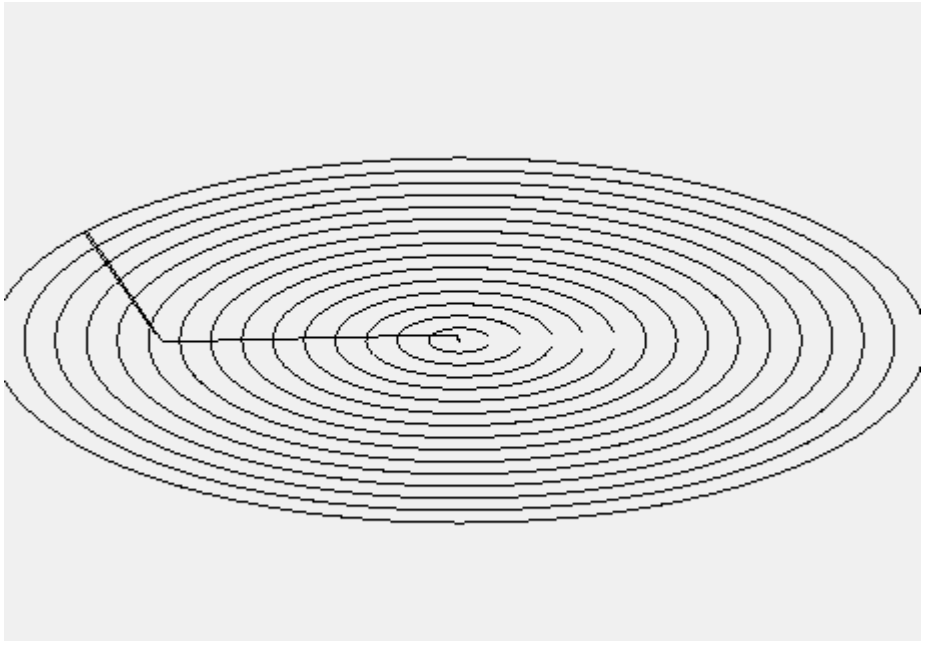


Рис 2.14 Метод сполучених градієнтів. Еліптична функція.

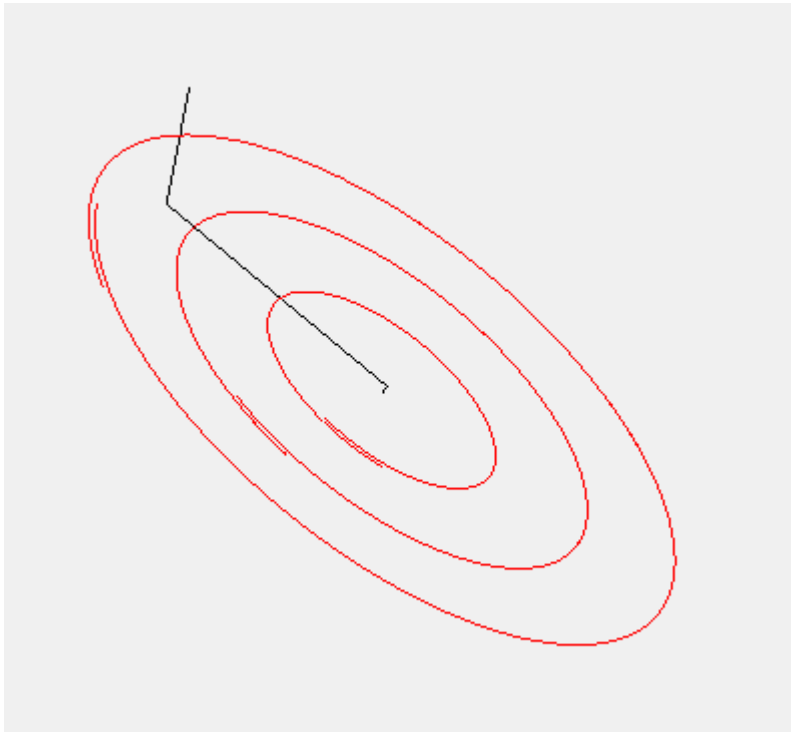


Рис 2.15 Метод сполучених градієнтів

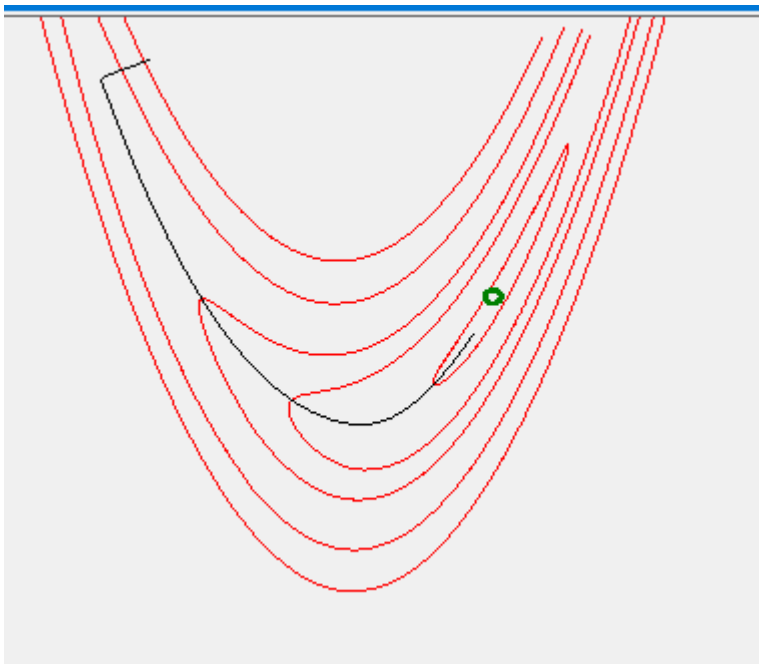


Рис 2.16 Метод сполучених градієнтів. Функція розенброка.

.

2.1. Висновки

В розділі досліджені експериментально найбільш важливі та поширені варіанти методів градієнтної мінімізації на прикладах опуклих і не опуклих функцій.. Для тестування методів використовувалось дві технології. Перша базувалась на наочному графічному зображенні траєкторії спуску для еліптичних та не опуклих функцій. В якості не опуклих функцій використовувалась функція Розенброка (англ. Rosenbrock function, Rosenbrock's valley, Rosenbrock's banana function) - не опукла функція , що використовується для оцінки продуктивності алгоритмів оптимізації , запропонована Ховардом Розенброком (англ.) в 1960 [5] . Вважається, що пошук глобального мінімуму для цієї функції є корисною перевіркою та нетривіальним завданням.

Для не графічного тестування методів використовувався наступний підхід: випадковим чином вибирався деякий n – вимірний вектор і з нього обчислювалася

$$\mathbf{x}^* = \arg\min_{x \in D} f(\mathbf{x})$$

Мірою помилки у разі може бути норма різниці векторів. Випробування проводилися M різних початкових наближень; зазвичай M = 30. Компоненти вектора початкового наближення вибиралися випадково з відрізка [-2, 2]. Потім отримані результати усереднювалися. Серії випробувань проводилися для N= 12, 17, 24, 34, ... У цій послідовності відношення наступного числа до попереднього близько до кореня з 2, що в логарифмічному масштабі відповідає рівномірному кроку.

Залежність швидкості збіжності від параметра для тестових матриць розміром N = 30 демонструється на Рис. 2.4 де представлена залежність логарифму кількості ітерацій K від коефіцієнта релаксації для алгоритму з постійним кроком.

Розділ 3. Алгоритмічне та програмне забезпечення

Обґрунтування принципів програмної реалізації

Реалізуються програмні засоби інтерфейсів застосування та обчислювальні процедури засобами Microsoft Visual Studio - це програмна найбільш ефективне середовище для розробки всіх типів додатків для ОС Windows, інших середовищ, та WEB. Застосувань як консольних, так і з графічним інтерфейсом. Система Microsoft Visual Studio є найскладнішим інтегрованим середовищем розробки (integrated development enviroment - IDE), доступним для програмістів в даний час[12]. Вона є результатом довгої історії розвитку мов програмування та інтерфейсів і увібрала у собі досягнення багатьох середовищ розробки програмного забезпечення.

Для програм як застосування оптимізації особливо важливим є швидкість роботи. До різних додатків пред'являються різні вимоги щодо продуктивності, які визначаються різними потребами. В одних випадках характеристики продуктивності явно визначаються архітектурою програми: наприклад, щоб веб - сервер міг обслуговувати мільйони користувачів, необхідно створити розподілену систему з кількох серверів, що забезпечує підтримку розподілу навантаження. В інших, результати тестування продуктивності можуть вимагати зміни самої архітектури програми.

На відміну від вимог до продуктивності, характеристики продуктивності не пов'язані з якимось конкретним типом програми або оточенням. Характеристики продуктивності – це числове вираження особливостей поведінки програми. Вони можуть вимірюватися на будь-якому апаратному забезпеченні та будь-якому оточенні, незалежно від кількості активних користувачів, запитів або сеансів. У процесі розробки обирають характеристики для вимірювання та на їх основі визначаєте вимоги до продуктивності. Програма має характеристики, специфічні для своєї області. Необхідно ідентифікувати їх тут. Найбільш важливі навантаження на CPU та час виконання. рівень деталізації вимірів часто може змінюватися Істотно,

що, втім, не впливає на важливість самих параметрів. Наприклад, час виділення пам'яті або час виконання можуть вимірюватися на рівні системи, на рівні єдиного програми або навіть на рівні окремих методів чи рядків. Час виконання одного конкретного методу може виявитися набагато важливіше, ніж загальне навантаження на CPU або час виконання процесу в цілому. Збільшення рівня деталізації вимірів часто тягне за собою додаткові накладні витрати різних інструментів профілювання.

Один із підходів до тестування невеликих програм або бібліотечних методів називається тестуванням за принципом «скляної скриньки». Цей підхід ґрунтується на дослідженні вихідного коду, аналіз його складності, зміні і додаванні коду, що виконує вимірювання. Цей підхід, особливо цінний - і часто незамінний – коли потрібна найвища точність результатів хронометражу.

Для великих програм найчастіше використовується підхід, званий

тестуванням за принципом «чорної скриньки», коли параметри продуктивності визначаються людиною і потім вимірюються з застосуванням інструментів. Застосовуючи цей підхід, розробник не повинен будувати будь-які гіпотези про вузькі місця в додатку.

Існують інструменти, що автоматично аналізують продуктивність програми та надають результати вимірювань у простому та зрозумілому вигляді. Серед цих інструментів будуть згадані лічильники продуктивності (performance counters), механізм трасування подій для Windows (Event Tracing for Windows, ETW) та комерційні профільними..

Інструменти вимірювання продуктивності можуть негативно впливати на продуктивність. Лише деякі з них здатні надати точну інформацію, не додаючи свої накладні витрати. Точність інструменту часто спотворюється накладними витратами, які вони вносять при застосуванні до додатку.

Категорії лічильників продуктивності (або об'єктів продуктивності) представляють набори окремих лічильників для певних компонентів системи

Лічильники продуктивності – це окремі числові властивості у категоріях. Зазвичай прийнято вказувати категорію і назва лічильника продуктивності, розділяючи їх зворотним флішем, наприклад, (Процес \ Байт виняткового користування). Лічильники продуктивності можуть мати різні типи, включаючи прості числові значення (Процес \ Лічильник потоків)), швидкості проходження подій (Черга друку \ Друкованих байт/СЕК)), відсотки та середні значення (Показники роботи служб 3.0.0.0 \ Тривалість виклику)).

Примірники категорій лічильників продуктивності (входження) використовуються з метою створення різних наборів лічильників для різних екземплярів компонентів системи. Наприклад, у системі може бути кілька процесорів, тому для кожного з них є свій екземпляр у категорії Processor Information (Відомості про процесор), а також загальний екземпляр _Total). Одні категорії лічильників продуктивності можуть мати кілька екземплярів (таких більшість), інші – єдиний екземпляр (наприклад, категорія Memory (Пам'ять).

Лічильники продуктивності дозволяють отримати загальне уявлення про причини низької продуктивності, а дослідження файлів журналів допоможе зрозуміти, наскільки поведінка системи відрізняється від нормального.

Нижче перераховані ситуації, коли системний адміністратор або розробник, що займається проблемами продуктивності, зможе отримати уявлення про те, де знаходиться вузьке місце у додатку, перш ніж застосувати потужніші інструменти.

• Якщо в програмі є витоки пам'яті, лічильники продуктивності допоможуть з'ясувати, які операції виділення пам'яті – низько рівневі чи керовані – є джерелом цих витоків. Для цього достатньо порівняти лічильник (Процес \ Байт виняткового користування) з лічильником .NET CLR Memory \# Bytes in All Heaps (Пам'ять CLR .NET\Байт у всіх купах). Перший підраховує обсяг усієї пам'яті, виділеної для процесу (включаючи купу збирача сміття), а другий – лише обсяг керованої пам'яті.

Інструмент Performance Monitor (Системний монітор) дозволяє також визначати установки оповіщень – виконання певних завдань при перевищенні зазначеними лічильниками встановлених порогових значень. Цю можливість можна використовувати для створення спрощеної інфраструктури моніторингу, здатної надсилати електронні листи або повідомлення системному адміністратору у разі порушення обмежень продуктивності.

Наприклад, оповіщення можна налаштувати так, що воно автоматично перезапускатиме процес при досягненні небезпечної межі використовуваного обсягу пам'яті, або коли система вичерпає весь вільний простір на диску.

Наступний фрагмент коду – це те, що необхідно, щоб експортувати з програми категорію лічильників продуктивності, що має єдиний екземпляр, та оновлювати їх періодично. Передбачається, що клас AttendanceSystem зберігає інформацію про кількість процесів (користувачів), що зареєструвалися на даний момент, і ви хочете експортувати цю інформацію у вигляді лічильника продуктивності. (Щоб скомпілювати цей фрагмент, потрібно додати простір імен System.Diagnostics.)

```
public static void CreateCategory() {
 if (PerformanceCounterCategory.Exists("Attendance")) {
  PerformanceCounterCategory.Delete("Attendance");  }
 CounterCreationDataCollection  counters = new CounterCreation-
DataCollection();
 CounterCreationData  employeesAtWork = new CounterCreationData(
 "# Employees at Work", "The number of employees currently checked in.",
 PerformanceCounterType.NumberOfItems32);
 PerformanceCounterCategory.Create(
 "Attendance", "Attendance information for Litware, Inc.",
 PerformanceCounterCategoryType.SingleInstance, counters);
}
```

```
public static void StartUpdatingCounters() {
 PerformanceCounter employeesAtWork = new PerformanceCounter(
 “Attendance”, “# Employees at Work”, readOnly: false);
 updateTimer = new Timer(_ = > {
 employeesAtWork.RawValue = AttendanceSystem.Current.EmployeeCount;
 }, null, TimeSpan.Zero, TimeSpan.FromSeconds(1));
}
```

Системний монітор можна використовувати і для збирання іншої інформації, яка не має відношення до лічильників продуктивності. Наприклад, його можна застосовувати для збору інформації про системні налаштуваннях – значень ключів з реєстру, властивостей об'єктів WMI і навіть вмісту файлів на диску. Підтримується також можливість захоплювати дані, які постачаються провайдерами механізму ETW (про який розповідається далі) для подальшого аналізу. Використовуючи XML-шаблони, адміністратори можуть створювати групи збирачів даних на інших комп'ютерах і генерувати звіти, виконавши лише кілька простих операцій з налаштування.

Лічильники продуктивності дозволяють отримати масу цікавої інформації про продуктивність, але вони не можуть використовуватися як високопродуктивна інфраструктура моніторингу. Не існує системних компонентів, здатних оновлювати лічильники продуктивності частіше за кілька разів у секунду, а сама програма Performance Monitor (Системний монітор) не дозволяє читати значення лічильників частіше ніж один раз на секунду.

Якщо для аналізу потрібно виконувати виміри будь-яких характеристик тисячі разів на секунду, лічильники продуктивності виявляться непридатними для цього. Механізм трасування подій для Windows спеціально спроектований для високошвидкісного збору даних різних типів (не тільки числових).

Механізм трасування подій для Windows (ETW) – це високопродуктивний фреймворк реєстрації подій, вбудований у Windows. За аналогією з лічильниками продуктивності, багато компонентів системи та інфраструктура підтримки додатків, включаючи ядро Windows та CLR, визначають механізми надсилання подій – інформації про внутрішній стан компонентів. На відміну від лічильників продуктивності, які завжди активні, механізм ETW можна включати і вимикати під час виконання, щоб накладні витрати на збір та відправлення інформації впливали на продуктивність, тільки коли це справді необхідно.

Одним із найбагатших джерел інформації є провайдер ядра (kernel provider), який генерує події у моменти запуску процесів та потоків, завантаження DLL, розподілу блоків пам'яті, мережевих операцій введення/виводу та при виконанні трасування стека. У табл. 3.1 наводиться перелік деяких найбільш цікавих подій, що повідомляються ETW-провайдерами ядра та CLR.

Механізм ETW можна використовувати для дослідження загальної поведінки системи, наприклад, щоб з'ясувати, який процес споживає більшу частину обчислювальної потужності CPU, проаналізувати вузькі місця в операціях введення/виводу, отримати статистику, що стосується роботи збирача сміття та використання пам'яті керованими процесами, та у багатьох інших випадках.

Події ETW несуть у собі точний час їх виникнення, можуть містити додаткову інформацію користувача, а також стан стека на момент появи. Інформація про стан стека може використовуватися для виявлення джерел різних проблем. Наприклад, провайдер CLR може посилати події на початку та в кінці кожного циклу складання сміття. Ці події у комплексі з інформацією про стан стеку викликів можна використовувати для виявлення частин програми найчастіше викликають складання сміття.

Windows Performance Toolkit (WPT) – це комплект утиліт для управління сеансами ETW, збереження подій ETW у файлах журналів та їх обробки для подальшого відображення на екрані.

Може генерувати графіки та діаграми подій ETW, зведені таблиці, що включають інформацію про стан стека, та файли CSV для автоматизованого оброблення.

У 64-розрядній версії Windows для підтримки можливості трасування стека необхідно змінити налаштування в реєстрі,що забороняють вивантаження сторінок з кодом з оперативної пам'яті у файл підкачування (для самого ядра Windows та для всіх драйверів). Це може збільшити споживання оперативної пам'яті системою на кілька мегабайт.

Для перехоплення та аналізу подій ETW використовуються інструменти XPerf.exe та XPerfView.exe . Обидва повинні запускатися з привілеями адміністратора. Утиліта XPerf.exe має кілька ключів командного рядка, за допомогою яких можна вказати, які провайдери повинні включатися, розміри використовуваних буферів, ім'я файлу для збереження інформації про події та безліч інших параметрів.

Утиліта XPerfView.exe аналізує вихідну інформацію та генерує графічні звіти на основі інформації у файлі журналу.

Разом із подіями може зберігатися також інформація про стан стеку викликів, що часто допомагає виявити додаткові межі проблем продуктивності. Однак, щоб отримати інформацію про стан стеку зовсім необов'язково включати прийом подій від якогось певного провайдера – прапорець SysProfi дозволяє отримувати цю інформацію від усіх процесорів з інтервалом 1 мі лі сек. Це спрощений спосіб зрозуміти суть подій, що протікають у системі лише на рівні методів.

Інструмент WPT може стати в нагоді в самих ситуаціях, допоможе вникнути у поведінку системи та окремих процесів. Нижче наведено перелік скріншотів та описів прикладів подібних ситуацій:

• WPT може перехоплювати всі події дискових операцій введення/виводу в системі та виводити інформацію з прив'язкою до картки фізичного диска. Це дає змогу виявити найбільш дорогі операції введення/виводу, зокрема операції, що вимагають значних переміщень головок жорсткого диска.

• WPT може надати інформацію про стан стеків викликів для всіх процесорів у системі. Він групує стеки викликів за процесами, модулями та функціями, що дозволяє візуально оцінити, де система (або конкретний додаток) проводить найбільше часу. Керовані кадри стеків не підтримуються.

• WPT може відображати зведену інформацію про стан стеків викликів (коли утиліта прийому подій запускалася з ключем -stackwalk) - це дає можливість отримати повну інформацію про стеки викликів на момент створення певних подій.

• WPT може відображати зведені графіки подій різних типів, щоб простіше було виявити кореляцію, наприклад, між операціями введення/виводу, використанням пам'яті, навантаженням на процесор та іншими характеристиками.

Інструмент PerfMonitor можна використовувати для збору подій ядра чи навіть подій власного провайдера ETW (запускаючи його) з ключами /KernelEvents і /Provider), але зазвичай він застосовується для аналізу поведінки керованих програм з використанням вбудованих провайдерів CLR. За допомогою ключа runAnalyze йому можна вказати будь-який додаток для трасування, після завершення якого PerfMonitor згенерує докладний звіт у форматі HTML і відкриє його в стандартному браузері

Коли виконується запуск PerfMonitor з метою виконати програму та згенерувати звіт, він виводить у вікні терміналу наступні рядки.

```
C:\PerfMonitor > perfmonitor runAnalyze JackCompiler.exe

Starting kernel tracing. Output file: PerfMonitorOutput.kernel.etl
Starting user model tracing. Output file: PerfMonitorOutput.etl
Starting at 4/7/2012 12:33:40 PM
Current Directory C:\PerfMonitor
Executing: JackCompiler.exe {

} Stopping at 4/7/2012 12:33:42 PM = 1.724 sec
Stopping tracing for sessions 'NT Kernel Logger' and 'PerfMonitorSession'.
Analyzing data in C:\PerfMonitor\PerfMonitorOutput.etlx
GC Time HTML Report in C:\PerfMonitor\PerfMonitorOutput.GCTime.html
JIT Time HTML Report in C:\PerfMonitor\PerfMonitorOutput.jitTime.html
Filtering to process JackCompiler (1372). Started at 1372.000 msec.
Filtering to Time region [0.000, 1391.346] msec
CPU Time HTML report in C:\PerfMonitor\PerfMonitorOutput.cpuTime.html
Filtering to process JackCompiler (1372). Started at 1372.000 msec.
Perf Analysis HTML report in C:\PerfMonitor\PerfMonitorOutput.
analyze.html
PerfMonitor processing time: 7.172 secs.
```

Наприклад для методу, що визначає, що визначає складність модифікуй градієнтного спуску, що наведений нижче буде виконано наступне

```
public static int InstrumentedMethod(int param) {
    List< int > evens = new List < int > ();
    for (int i = 0; i < param; ++i) {
        if (i % 2 == 0) {
            evens.Add(i);
        }
    }
    return evens.Count;
}
```

Для оцінки характеристик продуктивності цього методу, профіль Visual Studio змінить його так…

```
public static int mmid = (int)
    Microsoft.VisualStudio.Instrumentation.g_fldMMID_2D71B909-
C28E-4fd9-A0E7-ED05264B707A;

public static int InstrumentedMethod(int param) {
    _CAP_Enter_Function_Managed(mmid, 0x600000b, 0);
    _CAP_StartProfiling_Managed(mmid, 0x600000b, 0xa000018);
    _CAP_StopProfiling_Managed(mmid, 0x600000b, 0);
    List < int > evens = new List < int > ();
    for (int i = 0; i < param; i++) {
        if (i % 2 == 0) {
            _CAP_StartProfiling_Managed(mmid, 0x600000b, 0xa000019);
            evens.Add(i);
            _CAP_StopProfiling_Managed(mmid, 0x600000b, 0);
        }
    }
    _CAP_StartProfiling_Managed(mmid, 0x600000b, 0xa00001a);
    _CAP_StopProfiling_Managed(mmid, 0x600000b, 0);
    int count = evens.Count;
```

Інструментоване профіль виглядає більш точним,але на практиці рекомендується використовувати дискретне профілювання, якщо програма в основному вирішує обчислювальні завдання. Прийом інструментування має обмежену гнучкість через необхідності змінювати виконуваний файл програми перед запуском та неможливості підключити профільник до вже запущеного процесу. Крім того, інструментоване профілювання має чималі накладні витрати – обсяг коду, що виконується істотно збільшується і в процесі виконання частини часу витрачається на відбір інформації у точках входу та виходу з методів. (Деякі інструментовані профільники пропонують режим рядкового профілювання коду, коли кожен рядок оточується кодом профілювача, який виконує вимірювання – такий код працює ще повільніше!)

Керівництво користувача

Програмний додаток дозволяє провести дослідження ефективності наступних оптимізаційних градієнтних алгоритмів.

1.Постійний крок. $step_{k1}$ $step_{k2}$ = $const>0$, Для функції, що задовольняє умові Ліпшиця та Розенброк.

$$\|f'(x)-f'(y)\| < L\|x-y\|$$

2. З дробленням кроку на кожній ітерації.

3. Найшвидший спуск (або метод повної релаксації).

h_k =arg min $x_k - step_{k1}\, v_{k1}$, мінімум визначається по $- step_{k1} > 0$.

4. З регулюванням кроку. Крок, на кожній ітерації, с заданим постійним коефіцієнтом якщо досягається зменшення функції за формулою

$x_{k+1} = x_k - step_{k1}\, v_{k1}$

5. Рандомізований крок. Параметри $step_{k1}$ $step_{k2}$ обираються випадково

Збіжність послідовності

$$x_1, \ldots, x_k, x_{k+1} \ldots \in E^n,$$

гарантується якщо крок знаходиться відповідно до правила Голдштейна - Армійо: h такий, що мають місце нерівності

$$x_{k+1} = x_k - hf'(x_k),$$

$$\alpha * (f'(x_k), x_k - x_{k+1}) \le f(x_k) - f(x_{k+1}),$$

$$\beta * (f'(x_k), x_k - x_{k+1}) \ge f(x_k) - f(x_{k+1}),$$

де задані наперед α, β, $0<\alpha<\beta<1$, — деякі фіксовані параметри.

В останньому випадку якість алгоритму суттєво залежить від вибору фіксованих значень α, β.

6. Метод сполучених градієнтів. Цей спосіб пошуку абсолютного екстремуму поєднує у собі поняття градієнта цільової функції та пов'язаних напрямів.

Програмний додаток складається з двох форм. Перша форма містить все кодування алгоритмів без умовної оптимізації градієнтних методів,

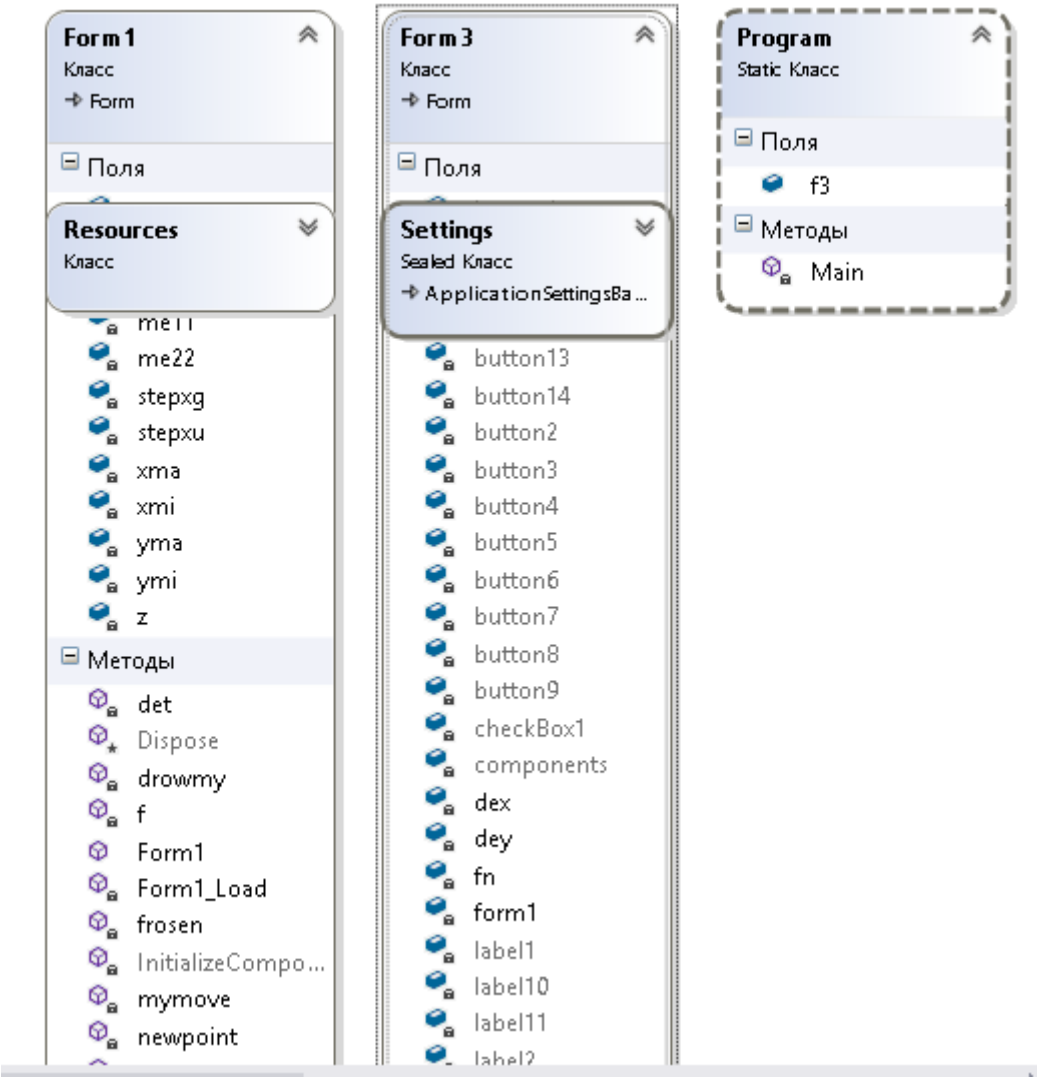


Рис 3.1, Об’єктна модель застосування.

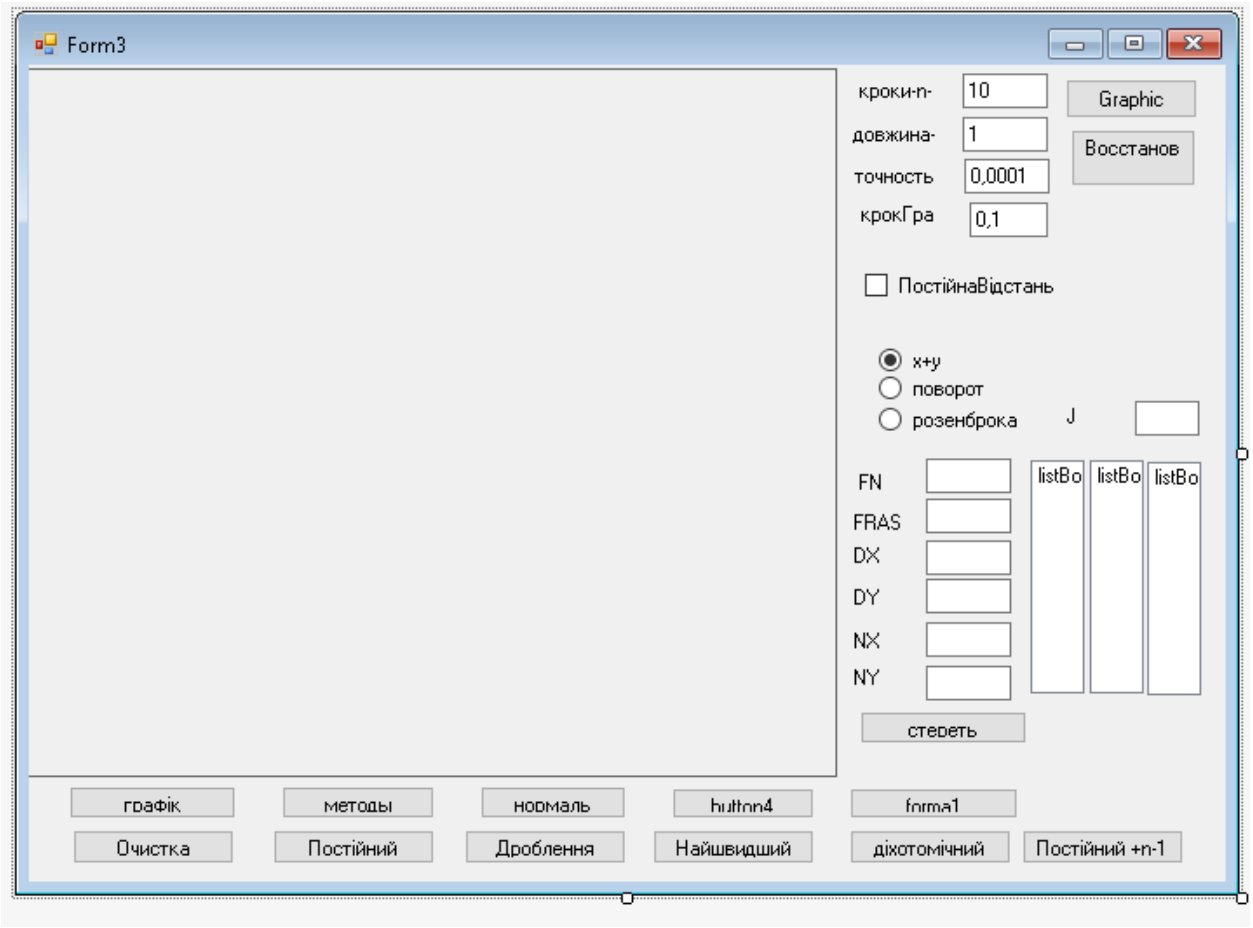


Рис 3.2, Загальний вигляд першої форми застосування для графічного відображення результатів.

3.3 Висновки

В розділі проаналізовані технології програмної реалізації задач, що пов’язані з методами градієнтного спуску

1.Постійний крок.

2. З дробленням кроку на кожній ітерації.

3. Найшвидший спуск (або метод повної релаксації)

4. З регулюванням кроку.

5. Рандомізований крок.

6. Метод сполучених градієнтів.

Проведено експериментальне дослідження продуктивності перерахованих модифікацій методу градієнтного спуску. Обґрунтований вибір програмних засобів і методи визначення складності обчислень

.

Висновки

Розроблене програмне забезпечення дозволяє отримати обґрунтовані рекомендацій вибору параметрів $v_{k1,}\ v_{k2}, \in E^n$, step_{k1} , $\text{step}_{k2} \in E^1$ алгоритмів, що для отримання розв'язку задачі мінімізації будують послідовності

$x_1 ,\ldots, x_{k,} . x_{k+1} \ldots \in E^n$,

з допомогою співвідношення

$x_{k+1} = x_k - step_{k1}\ v_{k1} \text{ -- } step_{k2}\ v_{k2},\ k = 0,\ 1\ \ldots\ x_{k+1}\ ,\ x_k \in E^n,\ step_{k1}\ ,\ step_{k2} \in E^1$

Програмний засіб дозоляє дослідити експериментально та порівняти ефективність вибору параметрів $v_{k1,}\ v_{k2}, \in E^n$, step_{k1} , $\text{step}_{k2} \in E^1$.

Чисельні методи мінімізації цього типу є важливою складовою базових інструментів наукової, інженерної та проектної діяльності.

.

СПИСОК ЛІТЕРАТУРИ

1. Дивак М.П. Ідентифікація дискретних моделей динамічних систем з інтервальними даними: монографія/ М.П. Дивак, Н.П. Порплиця, Т.М. Дивак. –Тернопіль: ВПЦ «Економічна думка ТНЕУ», 2018. – 220 с.

2. Оптимізаційні методи та моделі : підручник / В.С. Григорків, М.В. Григорків. – Чернівці : Чернівецький нац. ун-т, 2016. – 400 с.

3. Вибрані розділи багатокритеріальної оптимізації: методичні рекомендації до виконання контрольних та лабораторних робіт для студентів математичного факультету / Розробник: Н. Е. Кондрук. – Ужгород: УжНу, 2015. –56 с.

4. Дослідження операцій та методи оптимізації: методичні рекомендації до практичнх завдань для студентів усіх спеціальностей першого (бакалаврського) рівня / уклад. С. В. Прокопович, О. В. Панасенко, Л. О. Чаговець. – Харків: ХНЕУ ім. С. Кузнеця, 2019. – 64 с.

5. Синєглазов В. М. Математичні методи оптимізації: навч. посібн./ В.М. Синєглазов, О. А. Зеленков, Ш. І. Аскеров. – Нац. Авіаційний ун-т. – К.: Освіта України, 2018. – Ч. 1. – 329 с.

6. Латанська Л. О. Методичні вказівки до виконання самостійних робіт з дисципліни "Математичні методи дослідження операцій"/ Л. О. Латанська, 9 Т. А. Фаріонова ; Нац. ун-т кораблебудування ім. адмірала Макарова. – Миколаїв : НУК, 2018. – с. 29.

10. Латанська Л.О., Устенко І.В., Каіров В.О. Математичні методи дослідження операцій. Методичні вказівки до виконання лабораторних робіт (Частина 2). – Миколаїв: ФОП Швець В.М., 2018. – 36 с.

11. Snyman, J. A.; Wilke, D. N. (2018). Practical Mathematical Optimization : Basic Optimization Theory and Gradient-Based Algorithms (2nd ed.). Berlin: Springer. ISBN 978-3-319-77585-2.

12. Mathematical Programming Glossary. – Режим доступу: http://glossary.computing.society.informs.org/

13. Optimization Methods and Software. – Режим доступу:

https://www.tandfonline.com/toc/goms20/current10. Деннис Дж.Численные методы безусловной оптимизации и
решение нелинейных уравнений /Дж. Деннис., Р. Шнабель;
пер. с англ. – М:. Мир, 2016.
14. Зайченко Ю.П. Дослідження операцій /Ю.П Зайченко. К.
ВІПОЛ, 2010.
15. Карманов В.Г. Математическое программирование/В.Г.
Карманов. – М. : Наука, 1986.
16. Козицький В.А. Опуклі структури. Методи оптимізації та їхнє застосування в економічному аналізі /В.А. Козицький. –Львів.
ЛНУ імені Івана Франка, 2018.
17. Мойсеев Н.И., Методы оптимизации/Н.И. Мойсеев, Ю.П 2016
18. Полак Э. Численные методы оптимизации. Единый подход /Э.
Полак. – М. : Мир, 1974.
19. Пшеничный Б.Н. Метод линеаризации/Б.Н. Пшеничный. –М. :
Наука, 2016.
20. Пшеничный Б.Н. Численные методы в экстремальных задачах /
Б.Н Пшеничный, Ю.М. Данилин – М.: Наука, 1975.
21. Реклейтис Г. Оптимизация в технике /Г. Реклейтис., А.
Рейвиндра., К. Рэгсдел. – Т. 1,2. –М. : Мир, 1986.
22. Сухарев А.Г., Тимохов А.В., Федоров В.В. Курс методов
опмизации/ А.Г. Сухарев, А.В. Тимохов., В.В. Федоров. –М.:
Наука, 1986.
23. Химмельблау Д. Прикладное нелинейное программирование /
Д. Химмельблау. – М. : Мир, 1975.
24. Хедли Дж. Нелинейное и динамическое программирование/
Дж. Хедли. – М. : Мир, 1967..
25. Цегелик Г.Г. Лінійне програмування /Цегелик Г.Г. –Львів :
Світ, 20055.
26. Цегелик Г.Г. Чисельні методи /Цегелик Г.Г. – Львів : ЛНУ імені

Івана Франка,2014.
27. Численные методы условной оптимизации / Под ред. Ф. Гилла, У. Мюррея. – М : Мир, 1977.
28. Шор Н.З. Методы минимизации недифференциируемых функций и их приложение/ Н.З. Шор. – К.: Наук. думка, 1979.