Міністерство освіти і науки України

Одеський національний політехнічний університет

Інститут комп’ютерних систем

Кафедра прикладної математики та інформаційних технологій

РІШЕННЯ РОЗРІДЖЕНОЇ СЛАР МЕТОДОМ ПРОСТОЇ ІТЕРАЦІЇ І МЕТОДОМ ГАУССА

Пояснювальна записка до курсової роботи  
з дисципліни  
«Обчислювальні методи»

Виконав студент гр. АВ-171

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.О.Коваленко

Перевірила

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_доц. В.О. Гришина

ЗМІСТ

Вступ…………………………………………………………………………………….1

[1.1 Число обумовленості квадратної матриці. 2](#_Toc37450935)

[2.1 Матриці 4](#_Toc37450936)

[2.2 Види матриць і способи їх створення в Python 4](#_Toc37450937)

[2.2.2 Вектор-рядок 5](#_Toc37450938)

[2.2.3 Вектор-стовпець 5](#_Toc37450939)

[2.2.4 Квадратна матриця 6](#_Toc37450940)

[2.2.5 Діагональна матриця 7](#_Toc37450941)

[2.2.6 Одинична матриця 10](#_Toc37450942)

[2.2.7 Нульова матриця 12](#_Toc37450943)

[3.1 Лінійна алгебра 13](#_Toc37450944)

[3.2 Модуль numpy.linalg 13](#_Toc37450945)

4. Практичне дослідження............................................................................................17

Висновок.........................................................................................................................22

Список лiтератури…………………………………………………………………….23

ВСТУП

В даному проекті основоположні являєтся тема "Лінійна алгебра". Розповідається про базові поняття лінійної алгебри, які можуть бути корисні в подальшому дослідженні, матеріал супроводжується прикладами на мові *Python*.

Розглядається поняття матриці. Її важливою характеристикою є число обумовленості (condition number). Число обумовленості є мірою чутливості системи лінійних рівнянь, яка визначається матрицею. Важливість обумовлюється тим, що чим більше число обумовленості, тим сильніше цей вплив і тим більше нестійкий процес знаходження рішення лінійної системи.

Важливим залишається дослідження залежності числа обумовленості матриці від її порядку, а також її виду. У цій роботі буде розглядатися дослідження на основі теоретичного матеріалу і обчислень за допомогою мови *Python.*

**1.1 Число обумовленості квадратної матриці.**

Важливою характеристикою матриці є її число обумовленості (condition number). Число обумовленості є мірою чутливості системи лінійних рівнянь A x = b, яка визначається матрицею A, до погрішностей завдання вектора b правих частин рівнянь (див. Розділ 8). Чим більше число обумовленості, тим сильніше цей вплив і тим більше нестійкий процес знаходження рішення лінійної системи. Число обумовленості пов'язане з нормою матриці і обчислюється по-різному для кожної з норм:cond1(A) — число обусловленности в норме L1;

* cond2(A) - число обумовленості в нормі L2;
* conde (A) - число умовностей в евклідової нормі;
* condi (A) - число умовностей в норма:
* A - квадратна матриця.

Зверніть увагу, що перша з них є *добре обумовленою*, а друга - *погано обумовленою* (з точністю до множника 2). Другий рядок лістингу дає формальне визначення числа присутніх. В інших нормах визначення точно таке ж.

Перевірити чутливість рішення системи Ax = b до погрішностей можна і експериментально. Для цього досить вирішити завдання кілька разів з кількома близькими до b правими частинами b(1), b(2), …,b(n).

Можна очікувати, що величина дасть оцінку значення 

В усякому разі ця величина дає оцінку знизу так як 

**2.1 Матриці**

Матрицею називають об'єкт, що записується у вигляді прямокутної таблиці, елементами якої є числа (можуть бути як дійсні, так і комплексні). Приклад матриці наведено нижче.



У загальному вигляді матриця записується так:



Представлена вище матриця складається з i-рядків і j-стовпців. Кожен її елемент має відповідне позиційне позначення, яке визначається номером рядка і стовпця на перетині яких він розташований: aij- знаходиться на i-ому рядку і j-му стовпці.

Важливим елементом матриці є головна діагональ, її складають елементи, у яких збігаються номери рядків і стовпців.

**2.2 Види матриць і способи їх створення в Python**

Матриця в *Python* - це двовимірний масив, тому завдання матриць того чи іншого виду передбачає створення відповідного масиву. Для роботи з масивами в Python використовується тип даних список. Але з точки зору уявлення матриць і проведення обчислень з ними списки - не дуже зручний інструмент, для цих цілей добре підходить бібліотека *Numpy*, її ми і будемо використовувати в подальшій роботі.

Нагадаємо, для того, щоб використовувати бібліотеку *Numpy* її потрібно попередньо встановити, після цього можна імпортувати в свій проект. По установці *Numpy* можна докладно прочитати в розділі "*Установка* *бібліотеки* *Numpy*" зі вступу. Для того щоб імпортувати даний модуль, додайте в самий початок програми наступний рядок

import numpy as np

Якщо після імпорту не було повідомлень про помилку, то значить все пройшло вдало і можна починати роботу. *Numpy* містить велику кількість функцій для роботи з матрицями, які ми будемо активно використовувати. Обов'язково переконайтеся в тому, що бібліотека встановлена і імпортується в проект без помилок.

Розглянемо, різні варіанти матриць і способи їх завдання в *Python*.

***2.2.1 Вектор***

Вектором називається матриця, у якої є тільки один стовпець або один рядок.

#### 2.2.2 Вектор-рядок

Вектор-рядок має таку математичну запис

Такий вектор в *Python*можно задати наступним чином.

1. >>> v\_hor\_np = np.array([1, 2])
2. >>> print(v\_hor\_np )
3. [1 2]

Якщо необхідно створити **нульовий** або **одиничний** **вектор**, тобто вектор, у якого всі елементи нулі або одиниці, то можна використовувати спеціальні функції з бібліотеки *Numpy*.

Створимо нульову вектор-рядок розміру *5*.

1. >>> v\_hor\_zeros\_v1 = np.zeros((5,))
2. >>> print(v\_hor\_zeros\_v1 )
3. [0. 0. 0. 0. 0.]

У разі, якщо потрібно побудувати вектор-рядок так, щоб вона сама була елементом якогось масиву, це потрібно для можливості транспонування матриці, то це завдання можна вирішити так.

1. >>> v\_hor\_zeros\_v2 = np.zeros((1, 5))
2. >>> print(v\_hor\_zeros\_v2 )
3. [[0. 0. 0. 0. 0.]]

Побудуємо одиничну вектор-рядок в обох з представлених для нульового вектора-рядка форм.

1. >>> v\_hor\_one\_v1 = np.ones((5,))
2. >>> print(v\_hor\_one\_v1)
3. [1. 1. 1. 1. 1.]
4. >>> v\_hor\_one\_v2 = np.ones((1, 5))
5. >>> print(v\_hor\_one\_v2)
6. [[1. 1. 1. 1. 1.]]

## *2.2.3 Вектор-стовпець*

**Вектор-стовпець** має наступну математичну запис

У загальному вигляді вектор стовпець можна задати наступним чином.

1. >>> v\_vert\_np = np.array([[1], [2]])
2. >>> print(v\_vert\_np)
3. [[1]
4. [2]]

Розглянемо способи створення нульових і одиничних векторів-стовпців. Побудуємо **нульовий** **вектор-стовпець**.

1. >>> v\_vert\_zeros = np.zeros((5, 1))
2. >>> print(v\_vert\_zeros)
3. [[0.]
4. [0.]
5. [0.]
6. [0.]
7. [0.]]

**Одиничний вектор-стовпець** можна створити за допомогою функції*ones()*.

1. >>> v\_vert\_ones = np.ones((5, 1))
2. >>> print(v\_vert\_ones)
3. [[1.]
4. [1.]
5. [1.]
6. [1.]
7. [1.]]

## *2.2.4 Квадратна матриця*

Досить часто, на практиці, доводиться працювати з **квадратними** **матрицями**. Квадратної називається матриця, у якої кількість стовпців і рядків збігається. У загальному вигляді вони виглядають так.



Створимо наступну матрицю.



В *Numpy*можна створити квадратну матрицю за допомогою методу *array()*.

1. >>> m\_sqr\_arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
2. >>> print(m\_sqr\_arr)
3. [[1 2 3]
4. [4 5 6]
5. [7 8 9]]

Як ви вже напевно помітили, аргументом функції *np.array ()* є список *Python*, його можна створити окремо і передати в функцію.

1. >>> m\_sqr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
2. >>> m\_sqr\_arr = np.array(m\_sqr)
3. >>> print(m\_sqr\_arr)
4. [[1 2 3]
5. [4 5 6]
6. [7 8 9]]

Але в *Numpy* є ще одні спосіб створення матриць - це побудова об'єкта типу *matrix* за допомогою однойменного методу. Задати матрицю можна у вигляді списку.

1. >>> m\_sqr\_mx = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
2. >>> print(m\_sqr\_mx)
3. [[1 2 3]
4. [4 5 6]
5. [7 8 9]]

Також доступний стиль *Matlab*, коли між елементами ставляться прогалини, а рядки розділяються крапкою з комою, при цьому такий опис має бути передано у вигляді рядка.

1. >>> m\_sqr\_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
2. >>> print(m\_sqr\_mx)
3. [[1 2 3]
4. [4 5 6]
5. [7 8 9]]

## ***2.2.5 Діагональна матриця***

Особливим видом квадратної матриці є **діагональна** - це така матриця, у якої всі елементи, крім тих, що розташовані на головній діагоналі, дорівнюють нулю.



Діагональну матрицю можна побудувати вручну, задавши тільки значення елементів на головній діагоналі.

1. >>> m\_diag = [[1, 0, 0], [0, 5, 0], [0, 0, 9]]
2. >>> m\_diag\_np = np.matrix(m\_diag)
3. >>> print(m\_diag\_np)
4. [[1 0 0]
5. [0 5 0]
6. [0 0 9]]

Бібліотека *Numpy* надає інструменти, які можуть спростити побудову такої матриці.

Перший варіант підійде в тому випадку, якщо у вас вже є матриця, і ви хочете зробити з неї діагональну. Створимо матрицю розміру *3 3*.

1. >>> m\_sqr\_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')

Винесемо її головну діагональ.

1. >>> diag = np.diag(m\_sqr\_mx)
2. >>> print(diag)
3. [1 5 9]

Побудуємо діагональну матрицю на базі отриманої діагоналі.

1. >>> m\_diag\_np = np.diag(np.diag(m\_sqr\_mx))
2. >>> print(m\_diag\_np)
3. [[1 0 0]
4. [0 5 0]
5. [0 0 9]]

Другий варіант передбачає побудову одиничної матриці, їй буде присвячено наступний параграф.

## *2.2.6 Одинична матриця*

**Одиничною матрицею** називають таку квадратну матрицю, у якій елементи головної діагоналі рівні одиниці, а всі інші нулю.



Створимо одиничну матрицю на базі списку, який передамо в якості аргументу функції *matrix()*.

1. >>> m\_e = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
2. >>> m\_e\_np = np.matrix(m\_e)
3. >>> print(m\_e\_np)
4. [[1 0 0]
5. [0 1 0]
6. [0 0 1]]

Такий спосіб не дуже зручний, на щастя для нас, для побудови такого типу матриць в бібліотеці *Numpy* є спеціальна функція – *eye()*.

1. >>> m\_eye = np.eye(3)
2. >>> print(m\_eye)
3. [[ 1. 0. 0.]
4. [ 0. 1. 0.]
5. [ 0. 0. 1.]]

Як аргумент функції передається розмірність матриці, в нашому прикладі - це матриця *3 3*. Той же результат можна отримати за допомогою функції *identity()*.

1. >>> m\_idnt = np.identity(3)
2. >>> print(m\_idnt)
3. [[ 1. 0. 0.]
4. [ 0. 1. 0.]
5. [ 0. 0. 1.]]

## *2.2.7 Нульова матриця*

У **нульовий матриці** всі елементи дорівнюють нулю.



Приклад того, як створити таку матрицю з використанням списків, ми приводити не будемо, він робиться за аналогією з попереднім розділом. Що стосується *Numpy*, то в складі цієї бібліотеки є функція *zeros* (), яка створює потрібну нам матрицю.

1. >>> m\_zeros = np.zeros((3, 3))
2. >>> print(m\_zeros)
3. [[ 0. 0. 0.]
4. [ 0. 0. 0.]
5. [ 0. 0. 0.]]

Як параметр функції *zeros* () передається розмірність необхідної матриці у вигляді кортежу з двох елементів, перший з яких - число рядків, другий - стовпців. Якщо функції *zeros* () передати в якості аргументу число, то буде побудований нульовий вект ор-рядок, це ми робили в параграфі, присвяченому векторах.

**3.1 Лінійна алгебра**

Лінійна алгебра - це основа обчислювальної науки. Як матаналіз - основа математики в якомусь сенсі, то обчислювальна лінійна алгебра - це основа обчислювальної науки. Може здатися, що це щось просте. Що таке взагалі лінійна алгебра? Це наука про векторних просторах. Ми можемо складати вектори, є матриці або оператори, ми можемо множити матрицю на вектор, вирішувати лінійні системи. Виглядає просто.

**3.2 Модуль numpy.linalg**

*Numeric Python (NumPy)* - це кілька модулів для обчислень з багатовимірними масивами, необхідних для багатьох чисельних додатків. Масив - це набір однорідних елементів, доступних за індексами. Масиви модуля Numeric можуть бути багатовимірними, тобто мати більше однієї розмірності. Кількість розмірностей і довжина масиву по кожній осі називаються формою масиву (*shape*). Розміщення масиву в пам'яті проводиться відповідно до параметрів і може бути виконано як в мові С (за останнім індексом), як в мові Fortran (за першим індексом) або безладно.

Особливо підкреслимо відміну масиву від набору даних (списку чи кортежу). Величини, що входять в масив мають однаковий тип і їх кількість жорстко задається при ініціалізації. Елементи масиву не є об'єктами, це змінні в звичайному розумінні цього слова. Масиви дозволяють економити пам'ять і збільшувати швидкість роботи з великою кількістю однотипних даних.

Модуль *numpy.linalg* містить алгоритми лінійної алгебри, зокрема знаходження визначника матриці, рішень системи лінійних рівнянь, звернення матриці, знаходження власних чисел і власних векторів матриці, розкладання матриці на множники: Холецкого, сингулярне, метод найменших квадратів і т.д.

Нижче наведена таблиця

|  |  |
| --- | --- |
| **Команда** | **Опис** |
| dot(a, b[, out]) | скалярний добуток масивів |
| vdot(a, b) | векторний добуток векторів |
| inner(a, b) | внутрішнє твір масивів |

|  |  |
| --- | --- |
| outer(a, b) | зовнішнє твір векторів |
| tensordot(a, b[, axes]) | тензорне скалярний твір уздовж осі (розмірність більше 1) |
| einsum(subscripts, \*operands[, out, dtype, …]) | підсумовування Ейншнейн Evaluates the Einstein summation convention on the operands. |
| linalg.matrix\_power(M, n) | зведення квадратної матриці в ступінь n |
| kron(a, b) | твір Кронекера двох масивів |
| linalg.norm(a[, ord]) | норма матриці або вектора. |
| linalg.cond(a[, ord]) | число обумовленості матриці. |
| linalg.det(a) | визначник |
| linalg.slogdet(a) | знак і натуральний логарифм визначника |
| trace(a[, offset, axis1, axis2, dtype, out]) | сума елементів по діагоналі. |
| linalg.cholesky(a) | розкладання Холецкого |
| linalg.qr(a[, mode]) | розкладання QR |
| linalg.svd(a[, full\_matrices, compute\_uv]) | сингулярне розкладання |
| linalg.solve(a, b) | рішення лінійного матричного рівняння або системи скалярних рівнянь. |
| linalg.tensorsolve(a, b[, axes]) | рішення тензорного рівняння a x = b для x. |

|  |  |
| --- | --- |
| linalg.lstsq(a, b[, rcond]) | рішення матричного рівняння методом найменших квадратів |
| linalg.inv(a) | зворотна матриця (для множення) |
| linalg.pinv(a[, rcond]) | псевдо-обернена матриця (Мура-Пенроуза) |
| linalg.tensorinv(a[, ind]) | «Зворотний» до багатовимірного масиву |
| linalg.eig(a) | власні значення і праві власні вектора квадратної |
| linalg.eigh(a[, UPLO]) | власні значення і власні вектори ермітової або симетричною матриці |
| linalg.eigvals(a) | власні значення довільної матриці |

|  |  |
| --- | --- |
| linalg.eigvalsh(a[, UPLO]) | власні значення ермітової або дійсної симетричною матриці |

Табл. 1 Функціі біблиотеки numpy

Позначення:

a, b - матрицi

out - місце для результату

ord – визначає спосіб обчислення норми

axes – масив осей для підсумовування

axis – індекс осі

subscripts – індекси для підсумовування

\*operands – список масивів

dtype – тип результату

offset – становище діагоналі

mode - {‘full’, ‘r’, ‘economic’} – вибір алгоритму розкладання

full\_matrices – складати повні матриці

compute\_uv – виводити все матриці

rcond – межа для відкидання маленьких власних значень

ind – число індексів для обчислення зворотної

UPLO - {‘L’, ‘U’} вибирає частину матриці для роботи

1. **Практичне дослідження**

Для відстеження тенденції зміни значення числа обумовленості матриці були розглянуті квадратні матриця наступних типів (значення n 10/100/1000):

нульова матриця

еденічна

Матриця заповнена одиницямі

Матриця наповнена випадковими числами (розмір n 2x2 - 100x100)

Для обчислення числа обумовленості і реалізації графіків був використаний мову python, а також можливості бібліотек numpy і matplotlib для реалізації операцій роботи з матрицями

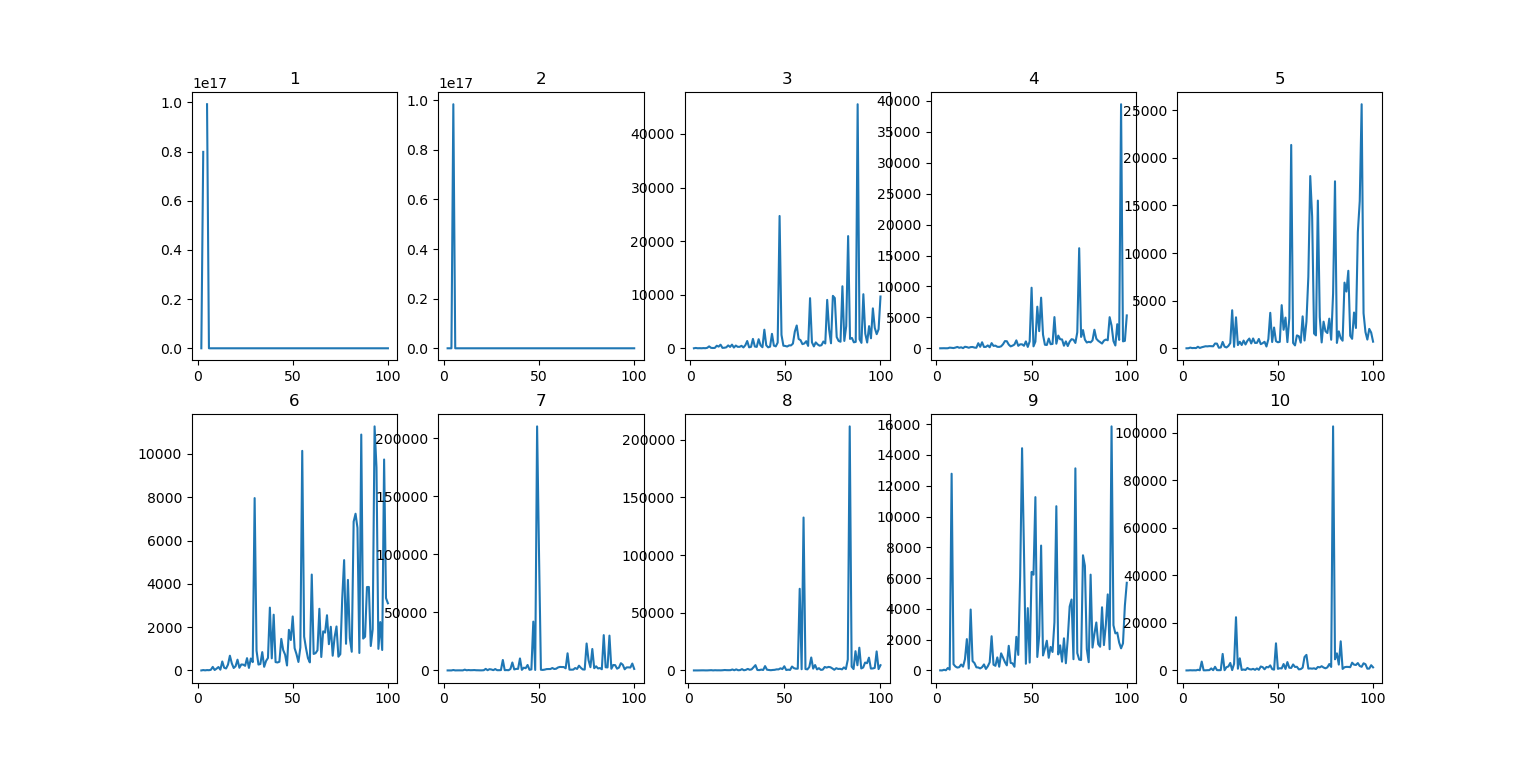


Рис 1.1 Графіки залежності числа обумовленності від n

Номерами графіків було вказано максимально можливе значення в кожному осередку матриці (1-10 відповідно).

ZERO Matrix size : 10/10 100/100 1000/1000 : [inf, inf, inf]

Ones Matrix size : 10/10 100/100 1000/1000 : [inf, inf, inf]

Matrix with ones diagonal size : 10/10 100/100 1000/1000 : [1.0, 1.0, 1.0]

Результат роботи програми для відображення відсутності зростання числа обумовленості в залежності від порядку n у випадках нульовими і еденічнамі матрицями.

**Код реализаціі (реализація на мові програмування python):**

# Для роботи з матрицями

import numpy as np

# число обумовленості

from numpy.linalg import cond

from random import randint

# графічне відображення

import pylab

def return\_compr(a:int, b:int, n:int):

# масив nxn заповнений випадковими числами в діапазоні a, b

return [[randint(a,b) for i in range(n)] for j in range(n)]

#zeros

zeros10 = np.zeros((10, 10))

zeros100 = np.zeros((100, 100))

zeros1000 = np.zeros((1000, 1000))

y\_zeros = [cond(zeros10), cond(zeros100), cond(zeros1000)]

print("ZERO Matrix size : 10/10 100/100 1000/1000 : ", y\_zeros)

#ones

ones10 = np.ones((10, 10))

ones100 = np.ones((100, 100))

ones1000 = np.ones((1000, 1000))

y\_ones = [cond(ones10), cond(ones100), cond(ones1000)]

print("Ones Matrix size : 10/10 100/100 1000/1000 : ", y\_ones)

#diags

y\_diag = [cond(np.diag(np.diag(ones10))), cond(np.diag(np.diag(ones100))), cond(np.diag(np.diag(ones1000)))]

print(" Matrix with ones diagonal size : 10/10 100/100 1000/1000 : ", y\_diag)

#random\_values

x = [i for i in range(2,101)]

y = [cond(return\_compr(0,5,i)) for i in range(2, 101)]

# pprint(y)

def code\_part(randrange):

# Отримуємо число обумовленості для нашої осі y

return [cond(return\_compr(0,randrange,i)) for i in range(2, 101)]

# побудова подграфіков

#2 рядки 5 стовпців

# 3й параметр відповідає за номер комірки

# графік будується по х - [0..100] у - [число обумовленості для згенерованих масивів з випадковими числами]

pylab.subplot (2, 5, 1)

pylab.plot (x, code\_part(2))

pylab.title ("1")

pylab.subplot (2, 5, 2)

pylab.plot (x, code\_part(3))

pylab.title ("2")

pylab.subplot (2, 5, 3)

pylab.plot (x, code\_part(4))

pylab.title ("3")

pylab.subplot (2, 5, 4)

pylab.plot (x, code\_part(5))

pylab.title ("4")

pylab.subplot (2, 5, 5)

pylab.plot (x, code\_part(6))

pylab.title ("5")

pylab.subplot (2, 5, 6)

pylab.plot (x, code\_part(7))

pylab.title ("6")

pylab.subplot (2, 5, 7)

pylab.plot (x, code\_part(8))

pylab.title ("7")

pylab.subplot (2, 5, 8)

pylab.plot (x, code\_part(9))

pylab.title ("8")

pylab.subplot (2, 5, 9)

pylab.plot (x, code\_part(10))

pylab.title ("9")

pylab.subplot (2, 5, 10)

pylab.plot (x, code\_part(11))

pylab.title ("10")

# Покажемо вікно з намальованим графіком

pylab.show()

ВИСНОВОК

У даній роботі було проведено дослідження щодо відстеження залежності числа обумовленості матриці від її порядку n. Нам також довелося познайомитися з рядом технологій для роботи з матрицями і відображення їх в графічному форматі для зручного сприйняття. Це бібліотеки мови python такі як

matplotlib (зокрема підпакету pylab) - для комплексного відображення реузльтата на графіках

numpy - для роботи з матрицями, зокрема такі операції як побудова нульовий або одиничних матриці. За підсумками перших із них було з'ясовано що для нульових і одиничних матриць число обумовленості не залежить від порядку n Що ж стосовно квадратних матриць заповнених випадковими числами був проведений аналіз для діапазонів випадкових чисел від 1-10 на графіках 1-10 відповідно. І можна помітити на більшій частині графіків що при n = +/- 50 число обумовленості доходило до максимального значення і далі спадало, звичайно присутній похибка через заповнення матриць випадковими велечіни, але для більш точного результату треба брати більші діапазон і будувати більшу кількість графіків.

СПИСОК ЛIТЕРАТУРИ

1. <https://studfile.net/preview/6372475/page:53/>
2. <https://devpractice.ru/python-linalg-lesson1-create-matrix/>
3. <https://physics.susu.ru/vorontsov/language/numpy.html>
4. <https://pyprog.pro/linear_algebra_functions/linalg_functions.html>
5. <http://scask.ru/i_book_clm.php?id=42>