Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Модульна контрольна робота

з дисципліни «Вступ до технології Data Science»

Виконала:
студентка групи ІС-12
Павлова Софія

Перевірив:
Баран Д. Р.

Київ – 2023

1. Постановка задачі

Варіант №22:

Білет №10

1. Оптимізація моделі.

2. Моделі випадкових законів розподілу випадкових величин.

3. З використанням засобів цифрової обробки графічних зображень Python та бібліотеки OpenCV розробити скрипт, що реалізує формування монохромного зображення з кольорового та фільтрацію зображення обраним методом. Зображення обрати самостійно.

2. Виконання

2.1. Теоретичне питання № 1

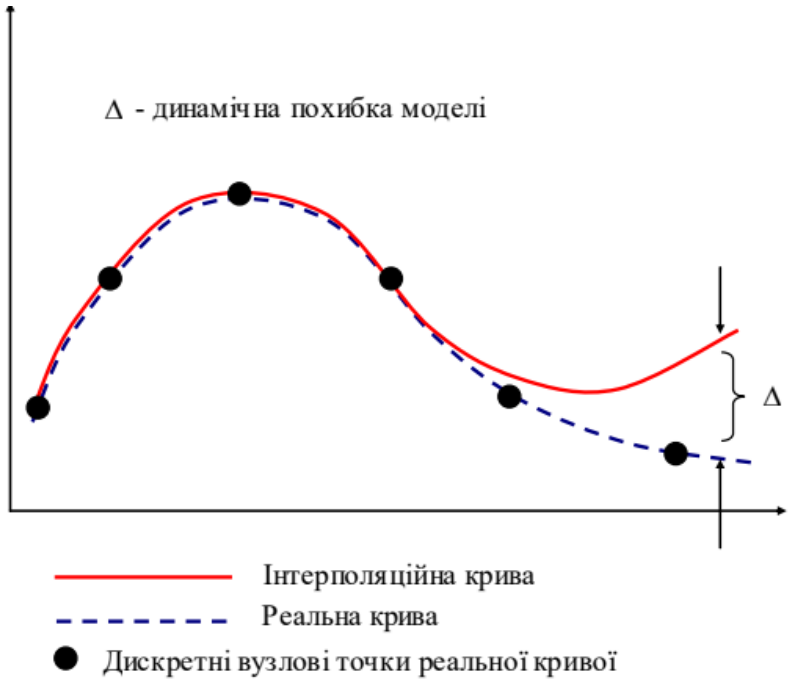
Оптимізація моделі

Для оптимізації моделі в для задач лінійної поліноміальної регресії необхідно підібрати вигляд полінома за кількісним складом елементів. Тобто кількість коефіцієнтів, порядок полінома і т. д.

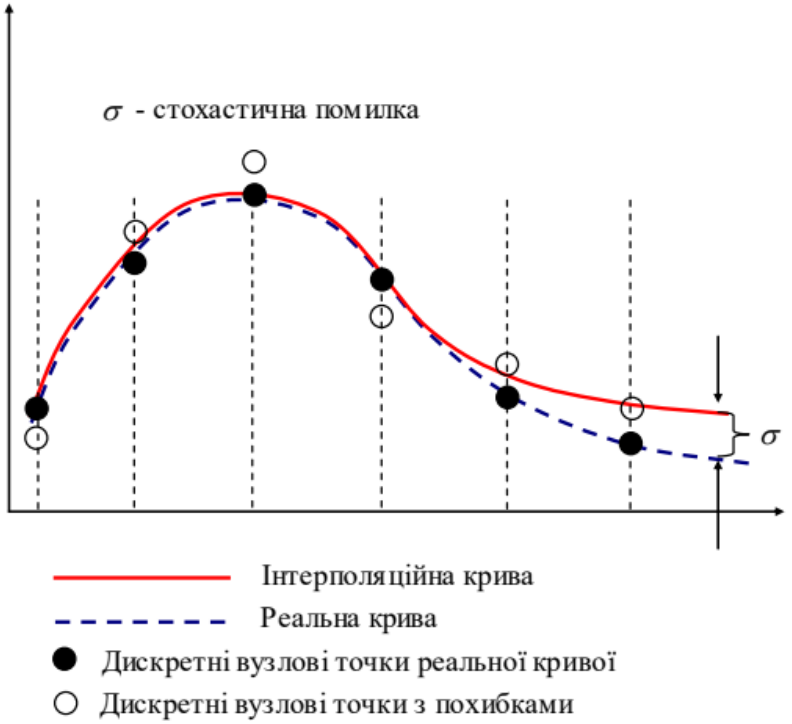
Для того, щоб оцінювати влучність підібраних параметрів прийнято загальноприйняті метрики оцінки похибки моделі.

Уведемо ці поняття.

Динамічна похибка – невідповідність реальних даних прийнятій моделі. Візуально *динамічну похибку* можна зобразити так:



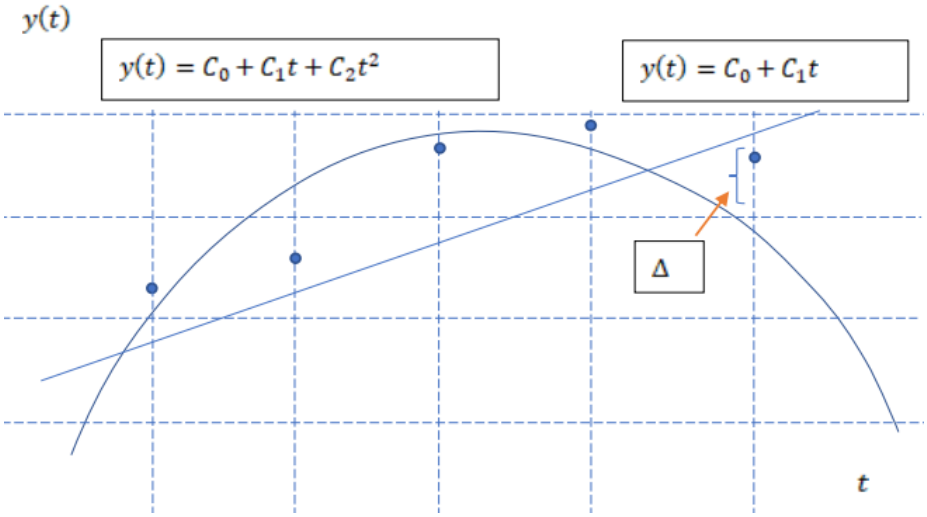
Статистична похибка – обумовлена похибками вхідних даних. Візуально *статистичну похибку* можна зобразити так:



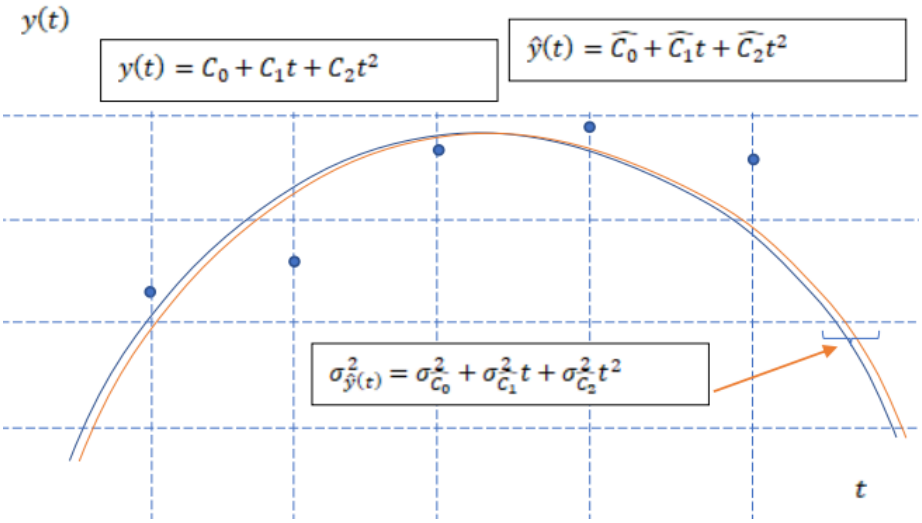
Для того, щоб розібратися глибше в природі похибок і як вони впливають на задачу оптимізації моделі, розглянемо деякий узагальнений приклад.

Припустимо, що ми маємо *поліном другої степені* – ідеальну квадратичну залежність.

Якщо ми будемо його апроксимувати простішою *лінійною моделлю*, то матимемо *меншу обчислювальну складність* за рахунок меншої кількості параметрів, але в той-же час отримаємо *велику розбіжність* результатів апроксимації. Така похибка й називається **динамічною**.



Якщо ми будемо його апроксимувати складнішою *квадратичною моделлю*, то матимемо *більшу обчислювальну складність*, так як збільшується кількість додаткових випадкових коефіцієнтів. Така похибка називається **статистичною**. Але в той-же час, така модель *краще апроксимує* дані.



Ось ми й підходимо до головної розбіжності статистичного навчання – динамічна й статистична похибки знаходяться в конфлікті. Зменшення однієї з них призводить до збільшення іншої. Таким чином **задача оптимізації моделі** зводиться до знаходження балансу між *динамічною* та *статистичною* похибками моделі.

2.2. Теоретичне питання № 2

Моделі випадкових законів розподілу випадкових величин

Для початку розберімося, що таке модель.

Будь-яку прикладну задачу галузі Data Science можна звести до дослідження процесів, систем та явищ. Для цього розробнику потрібні дані досліджуваного процесу – вхідна вибірка.

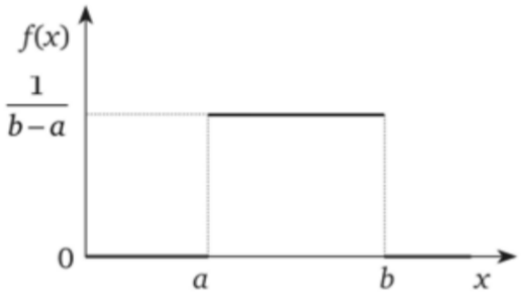
Їх можна отримати шляхом *проведення натуральних випробувань* або за допомогою *моделювання*. Останнє і є об'єктом застосування законів розподілу випадкових величин. Тому розглянемо детальніше процес моделювання даних.

Моделювання вибірки напряму пов'язане з досліджуваним процесом, який нам треба відтворити. Розробнику необхідно досконало розуміти природу досліджуваного процесу, аби підібрати відповідний йому закон розподілу даних.

Перейдемо до безпосереднього розгляду випадкових законів розподілу випадкових величин. Розрізняють **рівномірний**, **нормальний, експоненційний** та ***Хі*-квадрат** закони розподілу випадкових величин.

Рівномірний закон розподілу

Графічно його можна зобразити так:



Розподілена кількісна величина або існує в межах реалізації $[a, b]$, або її немає поза межами того інтервалу. Це добре видно з щільності розподілу ймовірностей:

$$f(x,a,b) = \begin{cases} 0, & -\infty < x < a, \\ \dfrac{1}{b-a}, & a \le x < b, \\ 0, & b \le x < \infty, \end{cases}$$

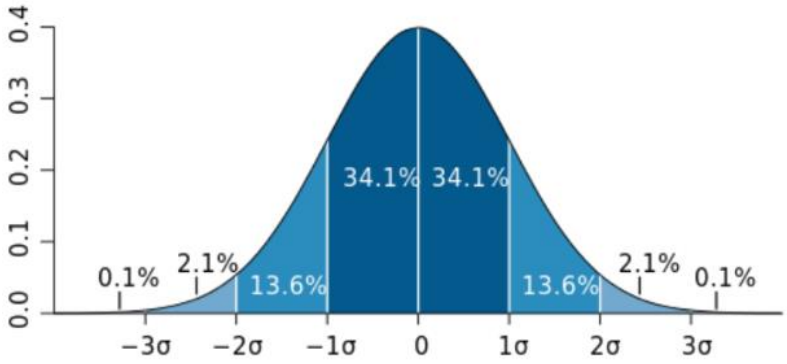
А статистичні характеристики розраховуються за наступними формулами:

$$m_x = \frac{a+b}{2},$$
$$D_x = \frac{(a-b)^2}{12}$$

Особливості застосування: Для цього закону характерне рівномірне розподілення кількісної величини на однакові за розміром ділянки $a, a_1, a_2, \dots, b$. Так, якщо в інтервал $[a, a_1]$ потрапляє 20 величин, а в наступний інтервал $[a_1, a_2]$ потрапила 21 величина, то скоріш за все ми маємо справу з *рівномірним законом розподілу випадкової величини*.

Нормальний закон розподілу

Графічно його можна зобразити так:



З графіку бачимо, що кількісна величина концентрується в центрі розподілу: на інтервали $[-1\sigma, 0], [0, 1\sigma]$ припадає 68,2% величини. У той час як на краї всього 27,2%. Математично цей процес описує щільність розподілу ймовірностей:

$$f(x, m_x, \sigma_x) = \frac{1}{\sqrt{2\pi\sigma_x}} exp\left(\frac{x - m_x}{2\sigma_x^2}\right)$$

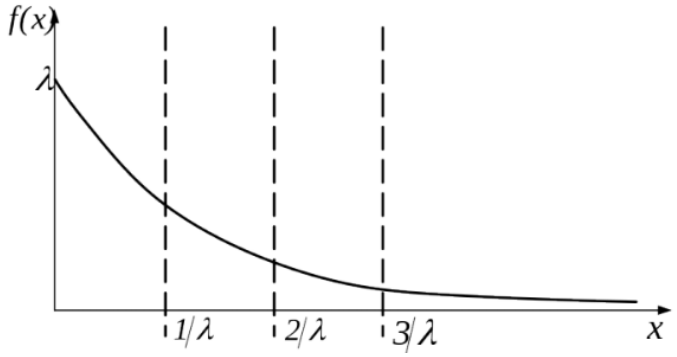
А статистичні характеристики розраховуються за формулами:

$$m_x = \frac{1}{n}\sum_{i=1}^{n} x_i,$$
$$D_x = \frac{1}{n}\sum_{i=1}^{n} (x_i - m_x)^2.$$

Особливості застосування: Нормальний закон властивий для процесів, що характеризуються декількома чинниками. Коли ми замінюємо багато чинників однією величиною (виміром), чинники накладаються один на одного і вступає в дію центральна гранична теорема – «При додаванні достатньо великого числа випадкових величин, закон розподілу їх суми нескінченно наближається до *нормального*».

Експоненційний закон розподілу

Графічно його можна зобразити так:



З графіку помітно, що розподіл кількісної величини відбувається з експоненційним спадом її впливу на рівнозначних ділянках. Для опису щільності розподілу експоненційного закону використовують формулу:

$$f(x,\alpha) = \begin{cases} 0, & -\infty < x < a \\ \alpha \cdot exp(-\alpha \cdot x), & 0 \le x < \infty \end{cases}$$

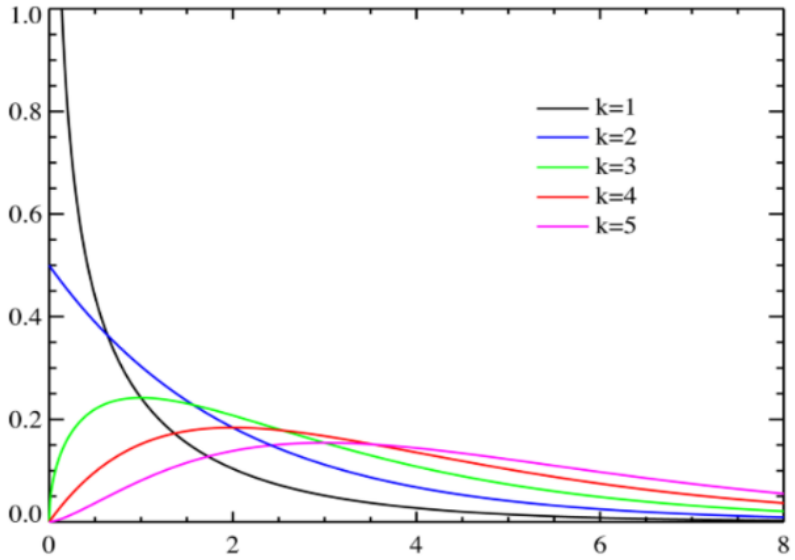
А статистичні характеристики розраховують за формулами:

$$m_x = \frac{1}{\alpha},$$
$$D_x = \frac{1}{\alpha^2}.$$

Особливості застовування: Цей закон характерний для економічних процесів. Скажімо, на досліджуваний процес діє певний зовнішній чинник. З плином часу вплив зовнішнього чинника зменшується і значення досліджуваного процесу поступово починають спадати. Це відбувається доти, доки зовнішній чинник не припинить впливати на досліджуваний процес. Тоді значення процесу стабілізуються. Такий розподіл вважається *експоненційним*.

Закон розподілу *Хі*-квадрат

Графічно його можна представити так:



Його щільність розподілу ймовірностей відповідає формулі:

$$f(x) = \begin{cases} 0, & x < 0 \\ \dfrac{1}{2^{\frac{k}{2}}\Gamma\left(\frac{k}{2}\right)} x^{\frac{k}{2}} exp\left(-\dfrac{x}{2}\right), & x \geq 0 \end{cases}$$

Особливості застосування: Цей закон характеризує події різної ймовірності. Зі збільшенням ступеня вільності цієї події k – закон перетворюється на нормальний. Таким чином для «маловільної події» (подій, на яку впливає мало чинників) отримуємо експоненційний розподіл, а для «багатовільної події» (події, на яку впливає багато чинників) – нормальний розподіл. Такий розподіл називається *Хі-квадрат*.

Підсумки: Звісно існують й інші випадкові закони розподілу випадкових величин, але на практиці вони зустрічаються значно рідше.

Таким чином, розуміння характеру досліджуваного процесу та особливостей моделей випадкових законів розподілу випадкових величин дозволяє обрати оптимальну для досліджуваного процесу модель генерації вибірки.

2.3. Практичне завдання

Постановка задачі

Сформулюємо завдання з власного досвіду.

Задача:

Тату майстра клієнти часто просять намалювати персонажа якогось всесвіту чи мультфільму. Для цього тату майстер шукає в інтернеті зображення цього героя і малює по ньому ескіз.

Від руки все перемальовувати важко і довго. Тому тату майстру необхідна програма, що приймає на вхід фото будь-якого формату, а повертає ескіз.

Для нашої задачі ідеально підійдуть такі методи обробки зображень як:

- перетворення кольорового зображення в **монохромне**;
- **фільтрація**.

Монохромне зображення

З використанням засобів цифрової обробки графічних зображень Python та бібліотеки OpenCV напишемо програмний скрипт, що перетворює кольорове зображення з інтернету в монохромне.

Напишемо програму таким чином, щоб користувач міг налаштовувати ступінь монохромності і обирати з якого режиму запуститись: монохромне зображення чи фільтрація.

Зробимо також програму гнучкою до зміни зображення, увівши його назву як параметр.

Лістинг коду:

```
from PIL import Image, ImageDraw
from matplotlib import pyplot as plt
from PIL.ImageFilter import (
   BLUR, CONTOUR, DETAIL, EDGE_ENHANCE, EDGE_ENHANCE_MORE,
   EMBOSS, FIND_EDGES, SMOOTH, SMOOTH_MORE, SHARPEN)

# Зчитування зображення
def image_read(file_name):
   # Відкриття файлу зображення
   image = Image.open(file_name)
   # Інструмент для малювання
   draw = ImageDraw.Draw(image)
   # Ширина зображення
   width = image.size[0]
   # Висота зображення
   height = image.size[1]
   # Значення пікселей для зображення
   pix = image.load()

   # Початкові характеристики
   print(f'\nПочаткове зображення: "{file_name}"')
   print(f'r = {pix[1, 1][0]},\tg = {pix[1, 1][1]},\tb = {pix[1, 1][2]}')

   # Виведення зображення
   plt.imshow(image)
   plt.show()
   image_info = {'image_file': image, 'image_draw': draw, 'image_width': width,
'image_height': height, 'image_pix': pix}

   return image_info

# Монохромне зображення
def monochrome(file_name_start):
   image_info = image_read(f'{file_name_start}.jpg')
   image = image_info['image_file']
   draw = image_info['image_draw']
   width = image_info['image_width']
   height = image_info["image_height"]
   pix = image_info['image_pix']

   print('Оберіть коефіцієнт монохромності:')
   factor = int(input('factor:'))

   for i in range(width):
      for j in range(height):
         a = pix[i, j][0]
         b = pix[i, j][1]
         c = pix[i, j][2]
         S = a + b + c
         # Рішення до якого з 2 кольорів поточне значення кольору ближче
         if (S > (((255 + factor) // 2) * 3)):
            a, b, c = 255, 255, 255
         else:
            a, b, c = 0, 0, 0
         draw.point((i, j), (a, b, c))

   # Виведення зображення
   plt.imshow(image)
   plt.show()

   # Кінцеві характеристики
```

```
    print(f'Кінцеве зображення: "{file_name_start}.jpg"')
    print(f'r = {pix[1, 1][0]},\tg = {pix[1, 1][1]},\tb = {pix[1, 1][2]}')

    # Збереження зображення
    file_name_stop = f'{file_name_start}_mono.jpg'
    image.save(file_name_stop, 'JPEG')
    del draw

    return

# Головні виклики
if __name__ == "__main__":

    file_name = 'img/friren'
    filters = [BLUR, CONTOUR, DETAIL, EDGE_ENHANCE, EDGE_ENHANCE_MORE, EMBOSS, FIND_EDGES,
SHARPEN, SMOOTH, SMOOTH_MORE]
    filter_names = ['BLUR', 'CONTOUR', 'DETAIL', 'EDGE_ENHANCE', 'EDGE_ENHANCE_MORE',
'EMBOSS', 'FIND_EDGES', 'SHARPEN','SMOOTH', 'SMOOTH_MORE']

    print('\nОберіть режим роботи програми:')
    print('1 - Монохромне зображення')
    print('2 - Фільтрація')
    mode = int(input('mode:'))
    # Якщо режим існує
    if mode in range(1, 3):
        # Монохромне зображення
        if (mode == 1):
            monochrome(file_name)
```

Результат:

Розглянемо результат на прикладі зображення героїні аніме «Фрірен, що проводжає в останню путь».

Ступінь монохромності = 10.

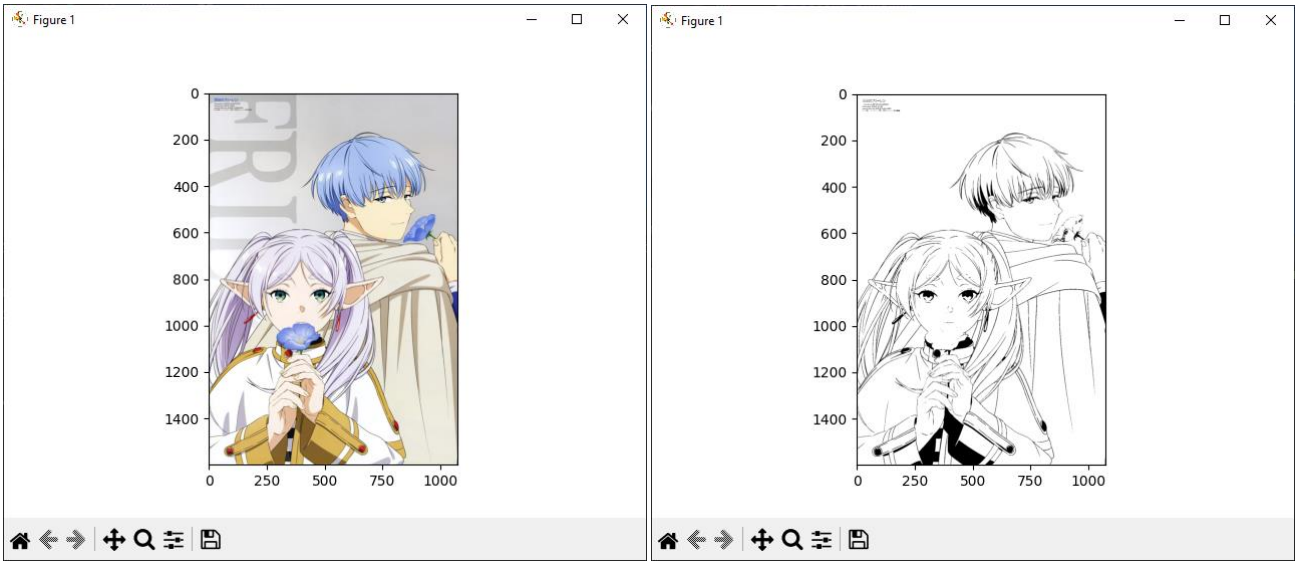


Рисунок 1 – Вхідне та монохромне зображення **factor = 10**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Оберіть коефіцієнт монохромності:
factor:10
Кінцеве зображення: "img/friren.jpg"
r = 255,    g = 255,    b = 255
```

Рисунок 2 – RGB характеристики зображення **factor = 10**

Бачимо, що через малий ступінь монохромності контур наявний не всюди, тому таке значення параметру не найкраще підходить для даного зображення.

Ступінь монохромності = 50.

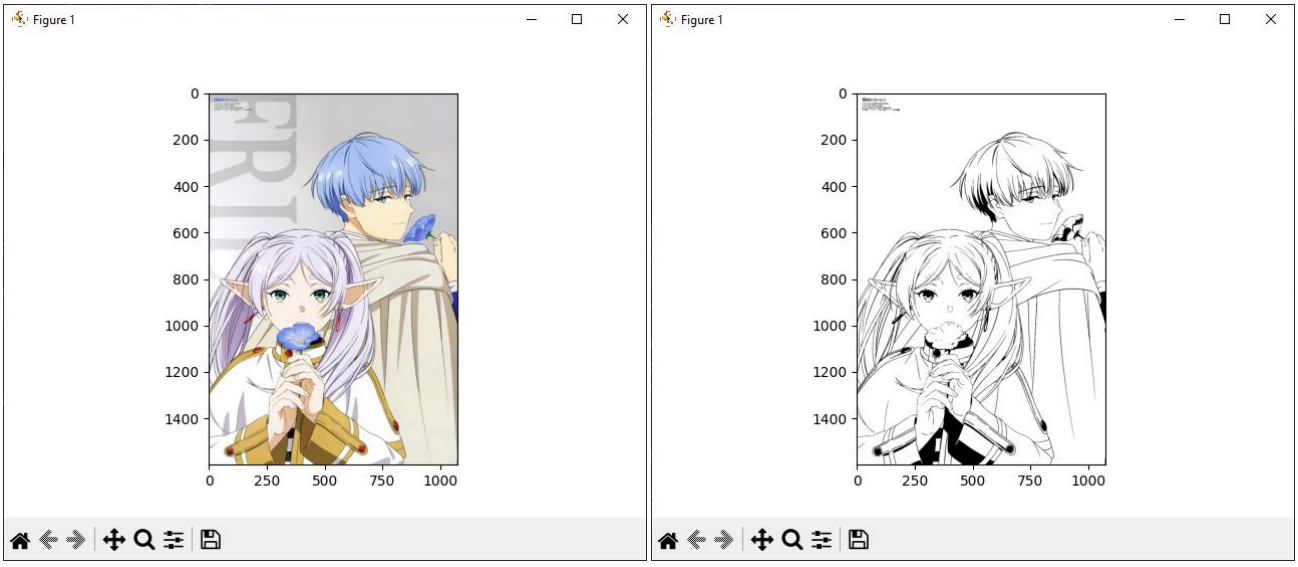


Рисунок 3 – Вхідне та монохромне зображення **factor = 50**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Оберіть коефіцієнт монохромності:
factor:50
Кінцеве зображення: "img/friren.jpg"
r = 255,    g = 255,    b = 255
```

Рисунок 4 – RGB характеристики зображення **factor = 50**

Бачимо, що ступінь монохромності чудово передає весь контур і підходить для нашого зображення.

Ступінь монохромності = 90.

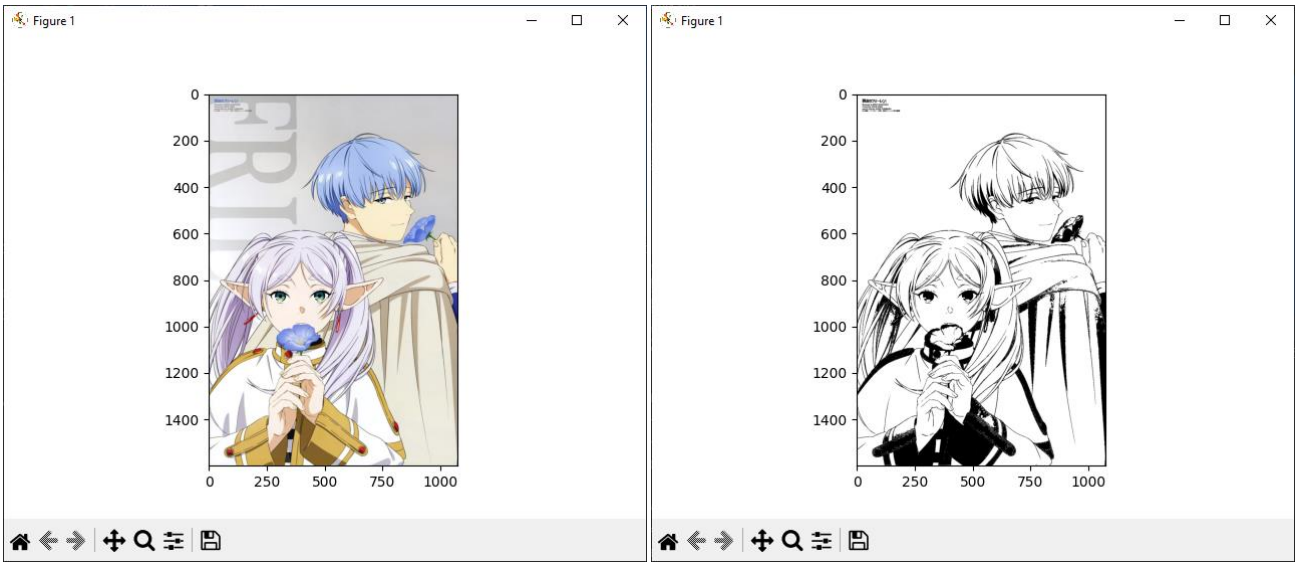


Рисунок 5 – Вхідне та монохромне зображення **factor = 90**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Оберіть коефіцієнт монохромності:
factor:90
Кінцеве зображення: "img/friren.jpg"
r = 255,    g = 255,    b = 255
```

Рисунок 6 – RGB характеристики зображення **factor = 90**

Бачимо, що через великий ступінь монохромності контуру забагато, що може збивати тату майстра. Через це таке значення параметру не найкраще підходить для даного зображення.

Фільтрація

Для реалізації фільтрації використаємо бібліотеку Pillow, за допомогою якої можна реалізувати фільтрацію в 10 різних режимах:

BLUR (Розмиття) – згладжує зображення, роблячи його менш деталізованим.

CONTOUR (Контур) – виділяє контури на зображенні.

DETAIL (Деталізація) – підкреслює деталі на зображенні.

EDGE_ENHANCE (Підвищення різкості контурів) – підсилює різкість контурів на зображенні.

EDGE_ENHANCE_MORE (Підвищення різкості контурів - більше) – подібно до EDGE_ENHANCE, але з більшим ефектом підвищення різкості.

EMBOSS (Тиснення) – створює ефект тиснення, підсилюючи різкість контурів.

FIND_EDGES (Виділення контурів) – виділяє контури об'єктів на темному фоні.

SMOOTH (Згладжування) – згладжує зображення, роблячи його менш контрастним.

SMOOTH_MORE (Згладжування - більше) – подібно до SMOOTH, але з більшим ефектом згладжування.

SHARPEN (Загострення) – загострює зображення, роблячи його більш деталізованим.

Напишемо універсальний інструмент, у якому передбачимо наявність усіх 10 режимів. Зробимо так, щоб користувач міг сам обирати який режим фільтрації використовувати.

Лістинг коду:

```
# Фільтрація
def filter(file_name_start, mode):
    image_info = image_read(f'{file_name_start}.jpg')
    image = image_info['image_file']
    draw = image_info['image_draw']

    # Фільтрація за обраним режимом
    image_filter = image.filter(filters[mode])

    # Виведення зображення
    plt.imshow(image_filter)
    plt.show()

    # Отримання значень пікселей для картинки
    pix = image_filter.load()

    # Кінцеві характеристики
    print(f'Кінцеве зображення: "{file_name_start}.jpg"')
    print(f'r = {pix[1, 1][0]},\tg = {pix[1, 1][1]},\tb = {pix[1, 1][2]}')

    # Збереження зображення
    file_name_stop = f'{file_name_start}_{filter_names[mode]}.jpg'
    image.save(file_name_stop, 'JPEG')
    del draw
```

```
    return

# Головні виклики
if __name__ == "__main__":

    [...]
    # Якщо режим існує
    [...]
        # Монохромне зображення
        [...]
        # Фільтрація
        if (mode == 2):
            print('\nОберіть варіант фільтрації:')
            for i in range(len(filters)):
                print(f'{i + 1} - {filter_names[i]}')
            mode = int(input('mode:'))
            # Якщо режим існує
            if mode in range(1, 11):
                filter(file_name, mode - 1)
```

Результат:

Розглянемо результат на прикладі того ж зображення.

BLUR

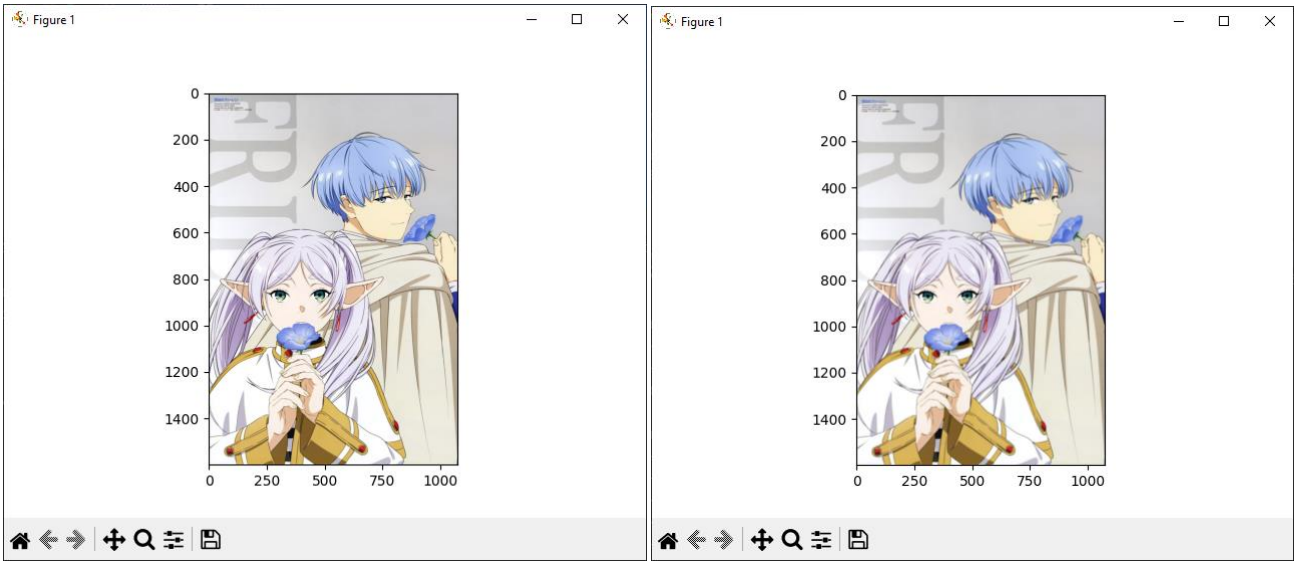


Рисунок 7 – Вхідне та фільтроване зображення **BLUR**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
```

Рисунок 8 – RGB характеристики зображення **BLUR**

Бачимо, що зображення ледь помітно заблюрилось. Характеристики кольору ж не змінились.

CONTOUR

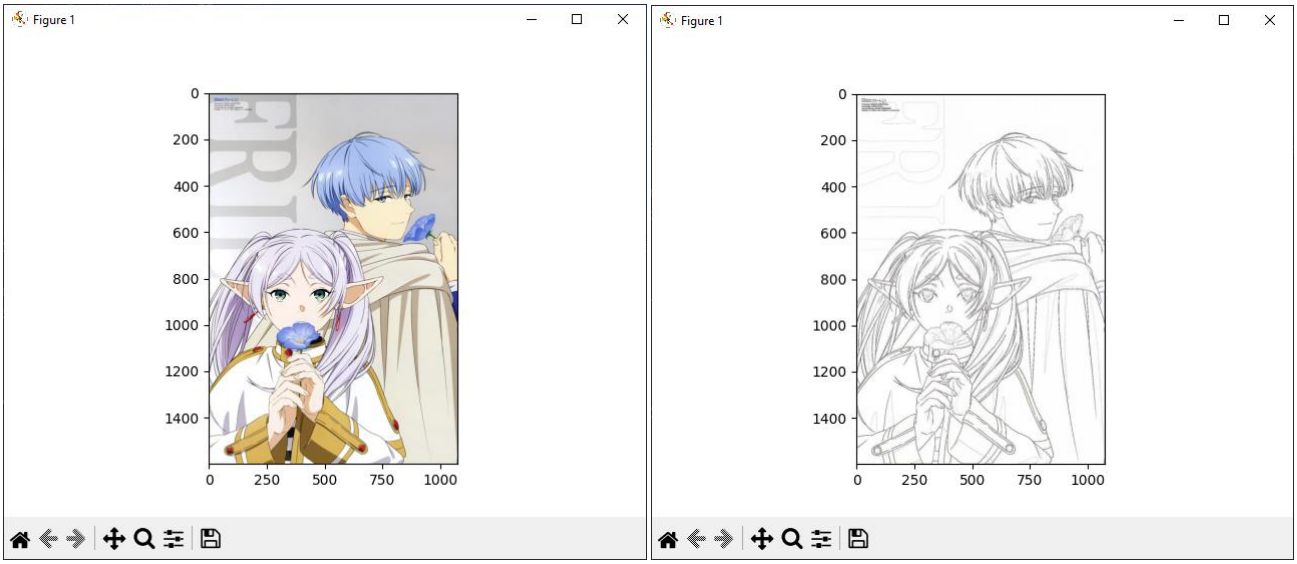


Рисунок 9 – Вхідне та фільтроване зображення **CONTOUR**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 255,    g = 255,    b = 255
```

Рисунок 10 – RGB характеристики зображення **CONTOUR**

Бачимо, що саме зображення стало білим, а контури з двох сторін виділені чорним. Такий стиль не найкраще рішення для тату майстра, але його можна розглядати як інструмент для підготовки об’ємних ескізів.

DETAIL

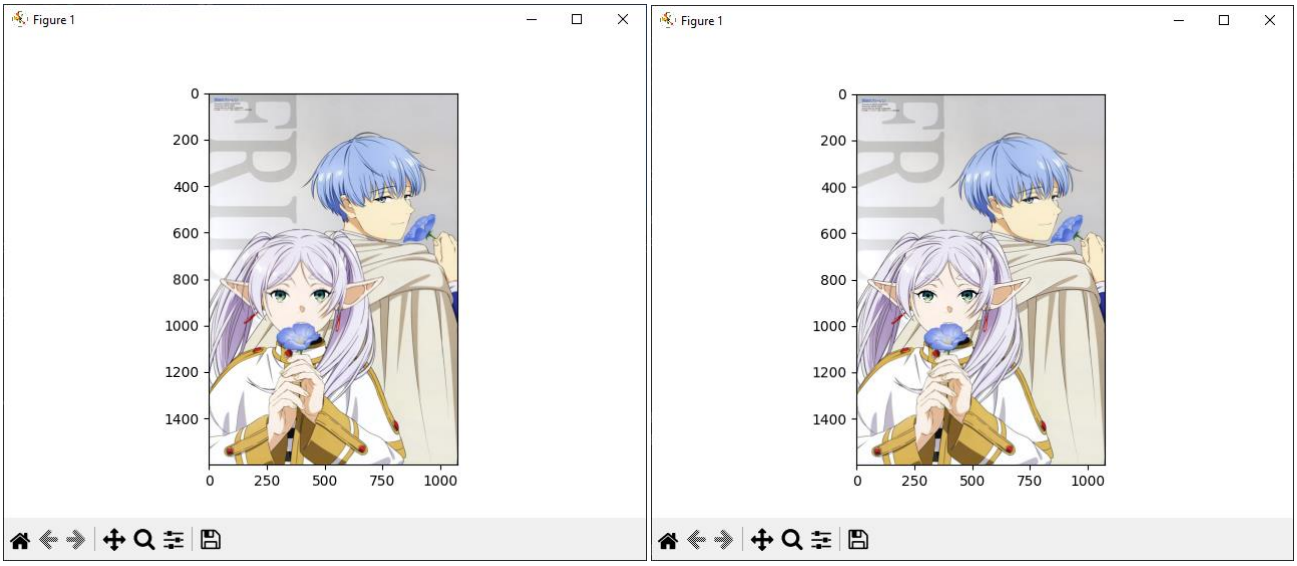


Рисунок 11 – Вхідне та фільтроване зображення **DETAIL**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
```

Рисунок 12 – RGB характеристики зображення **DETAIL**

Неозброєним оком різницю збільшення деталізації не видно, але такий інструмент може бути в нагоді тату майстру для роботи з зображеннями низької якості.

EDGE_ENHANCE

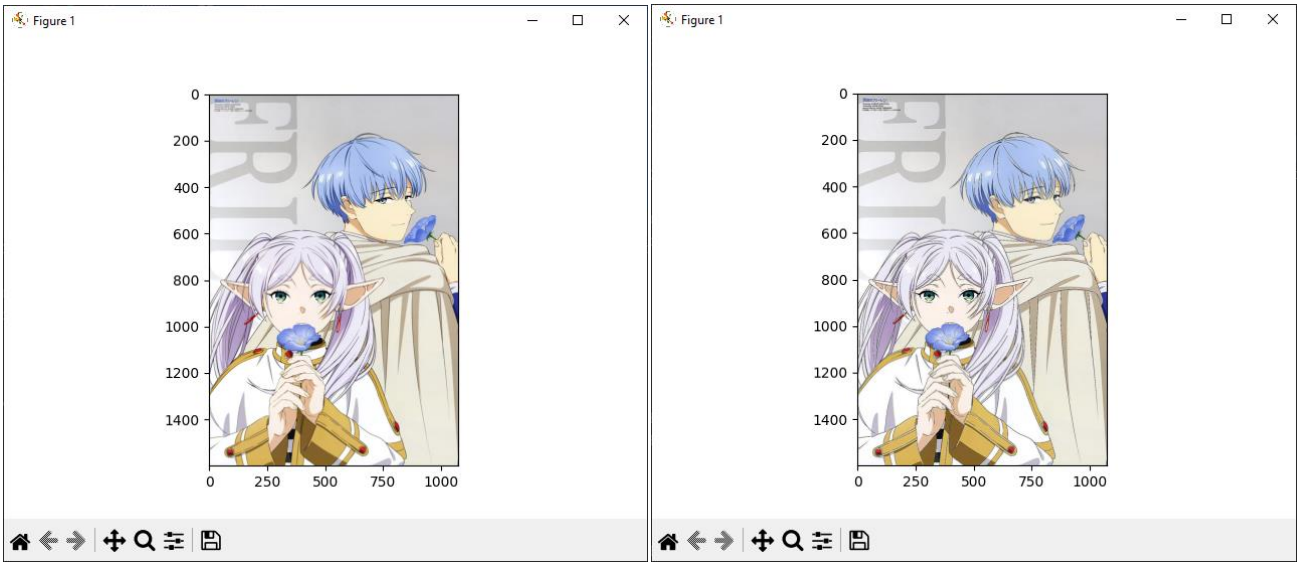


Рисунок 13 – Вхідне та фільтроване зображення **EDGE_ENHANCE**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
```

Рисунок 14 – RGB характеристики зображення **EDGE_ENHANCE**

Неозброєним оком різницю не видно, але така фільтрація підсилює контрастність контурів, що може бути корисно для попередньої обробки зображення перед або після перетворення кольорової картинки в монохромну.

EDGE_ENHANCE_MORE

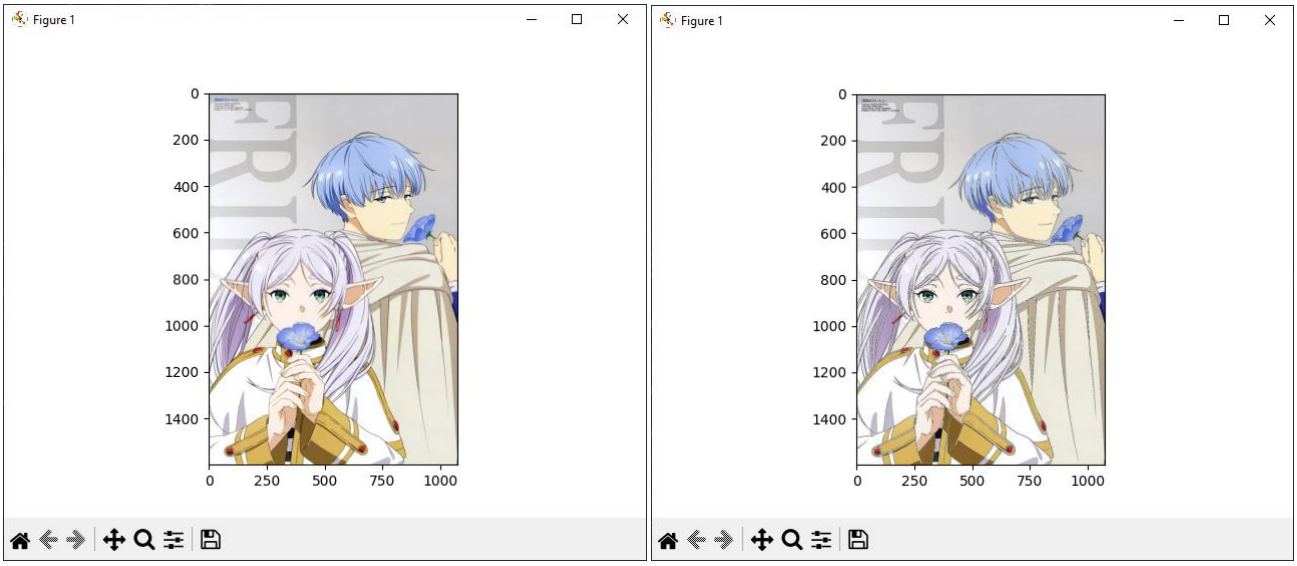


Рисунок 15 – Вхідне та фільтроване зображення **EDGE_ENHANCE_MORE**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
```

Рисунок 16 – RGB характеристики зображення **EDGE_ENHANCE_MORE**

Бачимо ледь помітне рябіння контурів. Така фільтрація може бути застосована як і попередній тип у парі з монохромним зображенням але для більш разючого результату.

EMBOSS

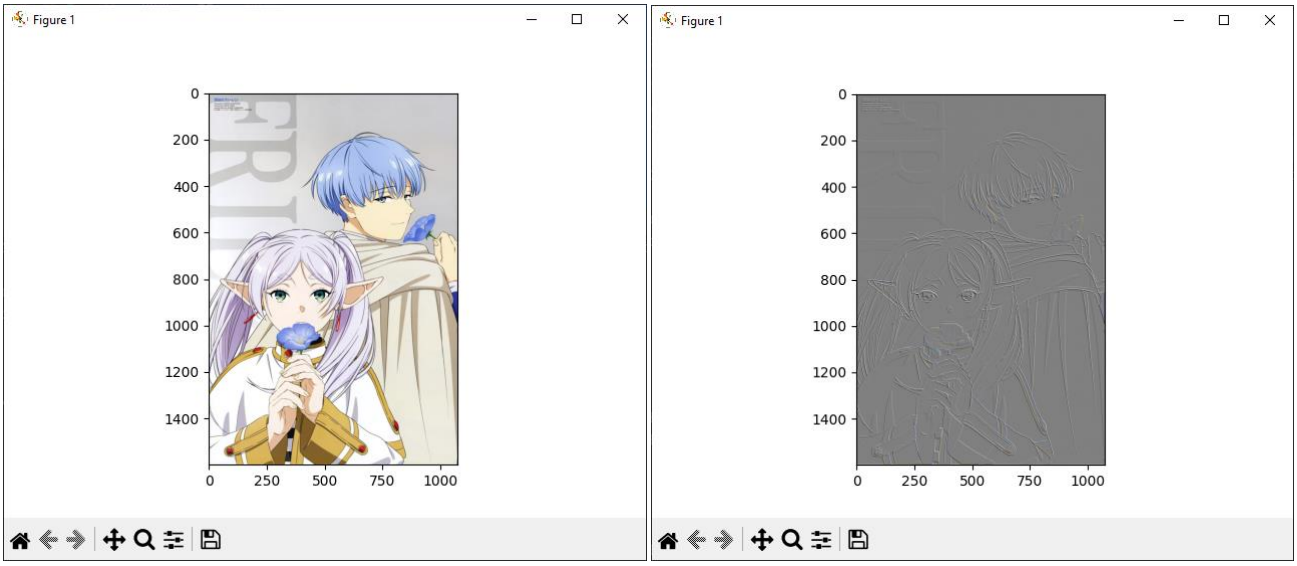


Рисунок 17 – Вхідне та фільтроване зображення **EMBOSS**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 128,    g = 128,    b = 128
```

Рисунок 18 – RGB характеристики зображення **EMBOSS**

Такий тип фільтрації навряд чи буде корисний тату майстру, навіть з урахуванням того, що він підсилює виділення контурів.

FIND_EDGES

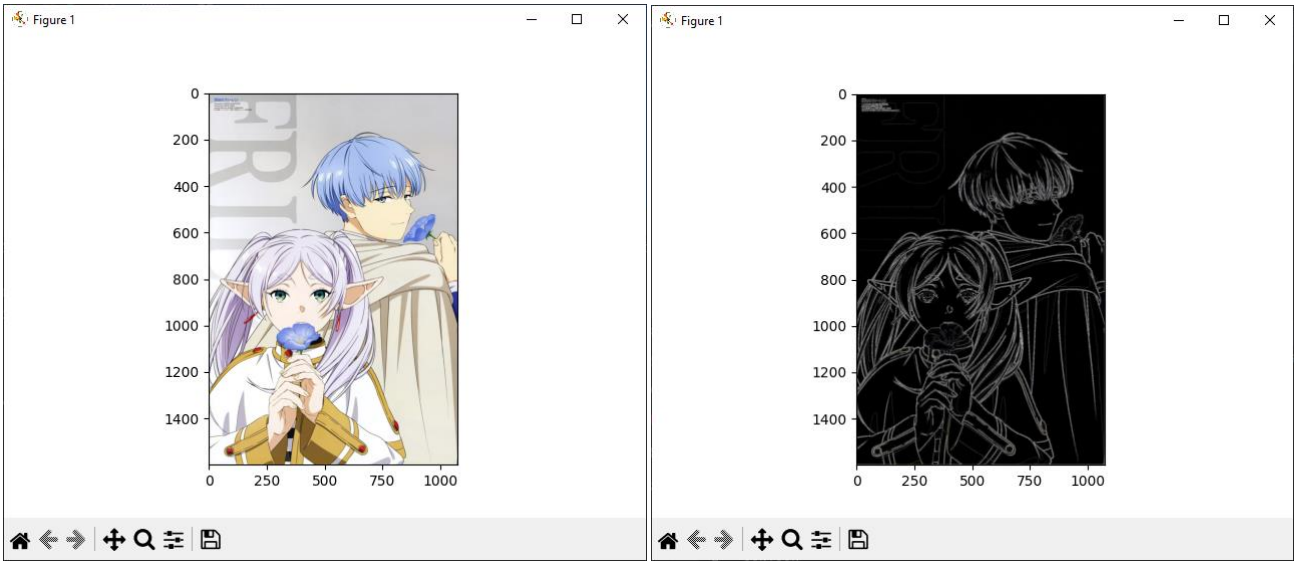


Рисунок 19 – Вхідне та фільтроване зображення **FIND_EDGES**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 0,  g = 0,  b = 0
```

Рисунок 20 – RGB характеристики зображення **FIND_EDGES**

Корисна функція для тату майстра, чудово виділяє контури, підійде для білих татуювань.

SHARPEN

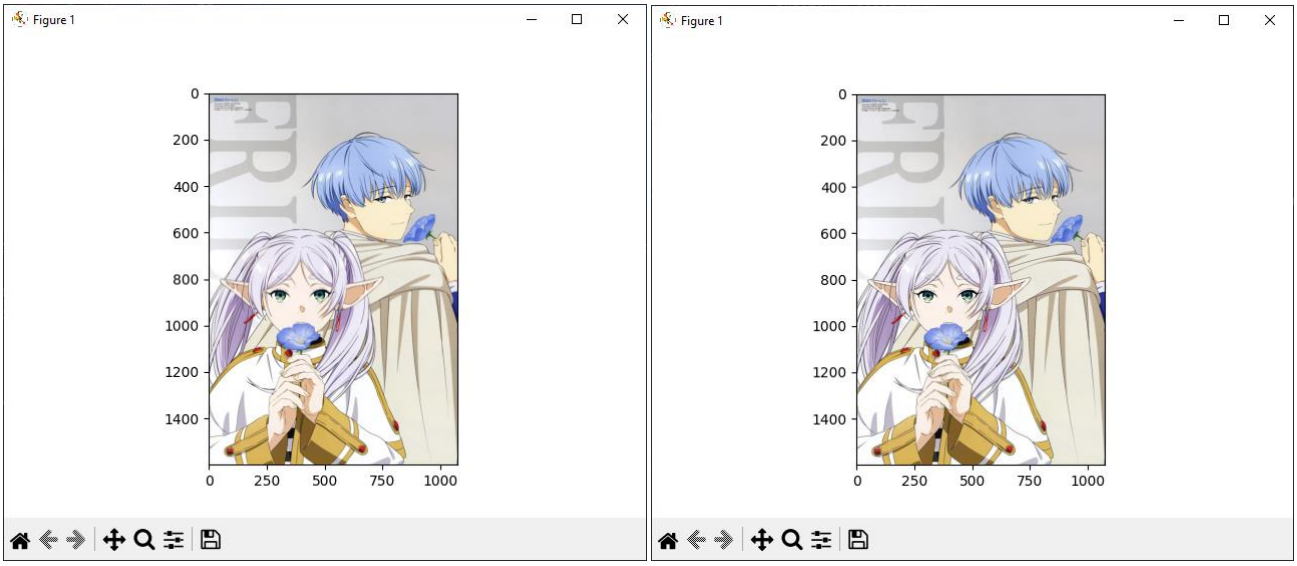


Рисунок 21 – Вхідне та фільтроване зображення **SHARPEN**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
```

Рисунок 22 – RGB характеристики зображення **SHARPEN**

Неозброєним оком різницю не видно, але така фільтрація підсилює загострить увагу на деталях. У цілому тип фільтрації подібний до DETAIL і вони взаємозамінні.

SMOOTH

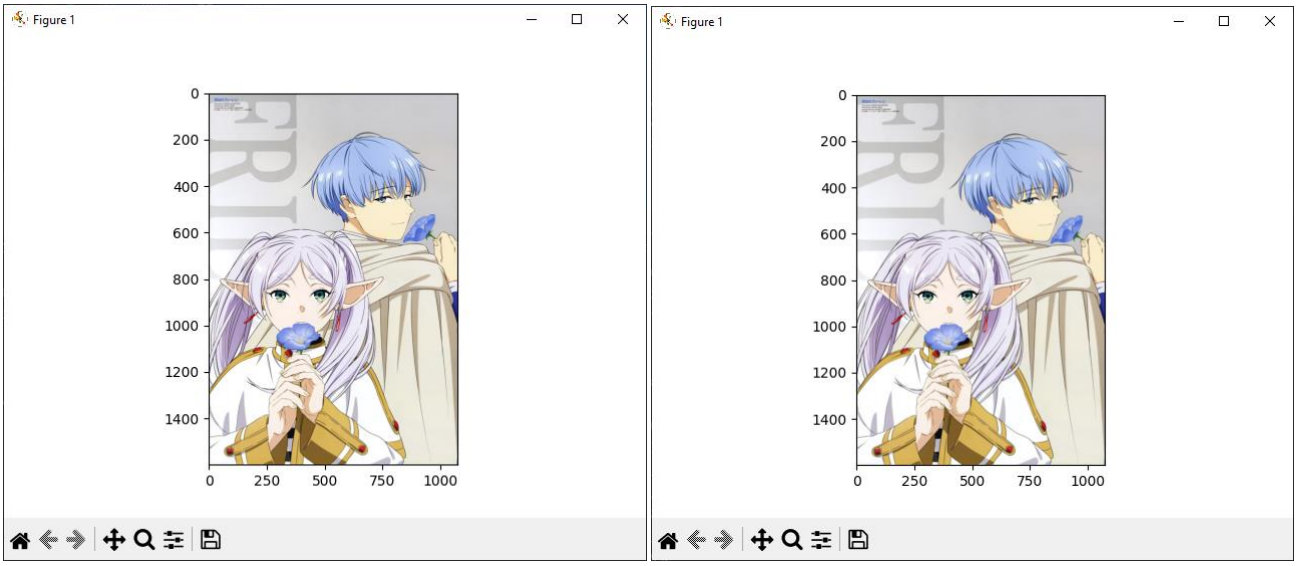


Рисунок 23 – Вхідне та фільтроване зображення **SMOOTH**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
```

Рисунок 24 – RGB характеристики зображення **SMOOTH**

Зменшення контрастності зображення не така вже й необхідна опція для обробки зображень в цілях розробки ескізів, тому тату майстру користі з неї мало.

SMOOTH MORE

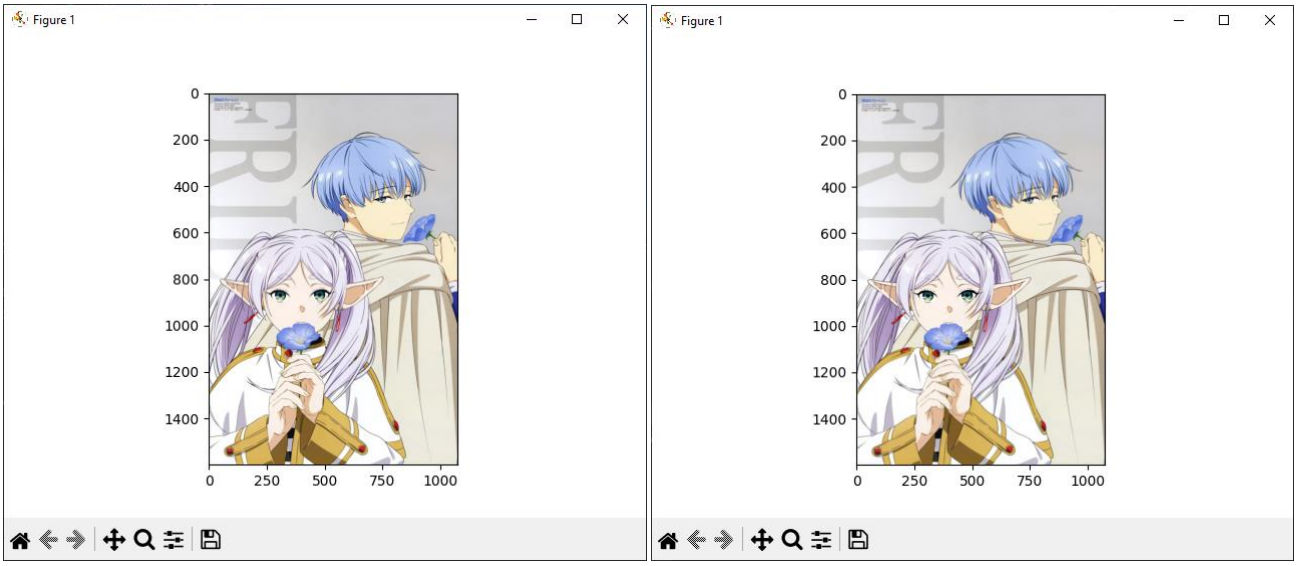


Рисунок 25 – Вхідне та фільтроване зображення **SMOOTH MORE**

```
Початкове зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
Кінцеве зображення: "img/friren.jpg"
r = 193,    g = 194,    b = 198
```

Рисунок 26 – RGB характеристики зображення **SMOOTH MORE**

Аналогічно до свого попередника, такий тип фільтрації навряд чи буде корисний тату майстру.

2.4. Аналіз отриманих результатів

Монохромне зображення

Розроблено програму, що з кольорового зображення утворює монохромне. Реалізовано ручне налаштування критерію монохромності. За результатами тестувань для задачі асистування створення ескізів татуювань отримано наступні результати.

Таблиця 1 – Порівняння влучності критерію монохромності

Монохромне зображення		
factor = 10	***factor = 50***	***factor = 90***
50%	100%	50%

Найкраще справилась програма з критерієм монохромності = 50. При менших значеннях у контурі були прогалини, а при більших контур дублював сам себе.

Фільтрація

Розроблено програму, що виконує 10 типів фільтрації зображень з метою покращення зображення перед або після операції перетворення зображення на монохромне. За результатами тестувань для задачі асистування створення ескізів татуювань отримано наступні результати.

Таблиця 2 – Порівняння влучності типу фільтрації

Фільтрація		
Тип	***Основний***	***Допоміжний***
BLUR	-	-
CONTOUR	1	0
DETAIL	0	1
EDGE_ENHANCE	0	1
EDGE_ENHANCE_MORE	0	1
EMBOSS	-	-
FIND_EDGES	1	0
SHARPEN	0	1
SMOOTH	-	-
SMOOTH_MORE	-	-

Корисними інструментами для тату майстра виявились такі типи фільтрації: *CONTOUR, DETAIL, ADGE_ENHANCE, EDGE_ENHANCE_MORE, FIND_EDGES та SHARPEN*. Більшість з них чудово підходить у якості допоміжного методи обробки зображення, але це не робить їх менш корисними. Інші 4 типи теж мають свої застосування. Єдине – вони не підходять для окресленої задачі.

Висновок:

Під час виконання модульної контрольної роботи було сформульовано прикладну задачу з власного досвіду для цифрової обробки зображень.

Розроблено 2 програмні скрипти: для формування монохромного зображення з кольорового та для фільтрації зображення обраним методом.

Реалізована програма гнучка до формату вхідного зображення, має декілька сценаріїв виконання й дозволяє користувачу налаштовувати коефіцієнт монохромності (для монохромного зображення) та обирати з 10 доступних типів фільтрації (для фільтрації).

Програма успішно апробована на тестовому зображенні.