НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Звіти до комп’ютерних практикумів з кредитного модуля “Вступ до Data Science”

Виконав

Студент групи ІТ-02

Макаров І.С.

Перевірив:

Професор кафедри ОТ ФІОТ
Писарчук О.О.

Київ – 2022

Комп‘ютерний практикум No 2.

ДОСЛІДЖЕННЯ АЛГОРИТМІВ ЗГЛАДЖУВАННЯ
ЗА НАКОПИЧЕНОЮ ВИБІРКОЮ

Мета:

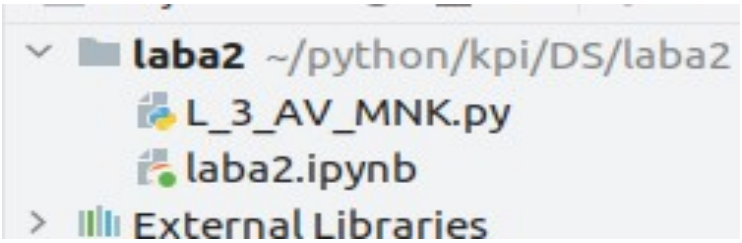
Виявити дослідити та узагальнити особливості застосування методів первинної обробки експериментальних вибірок – виявлення аномальних вимірів та алгоритмів накопиченого згладжування з використанням спеціалізованих пакетів мови програмування Python.

Варіант:

Я 11й у списку, тому аномалії я буду відкидати. А щодо методу їх виявлення, напишу нижче.

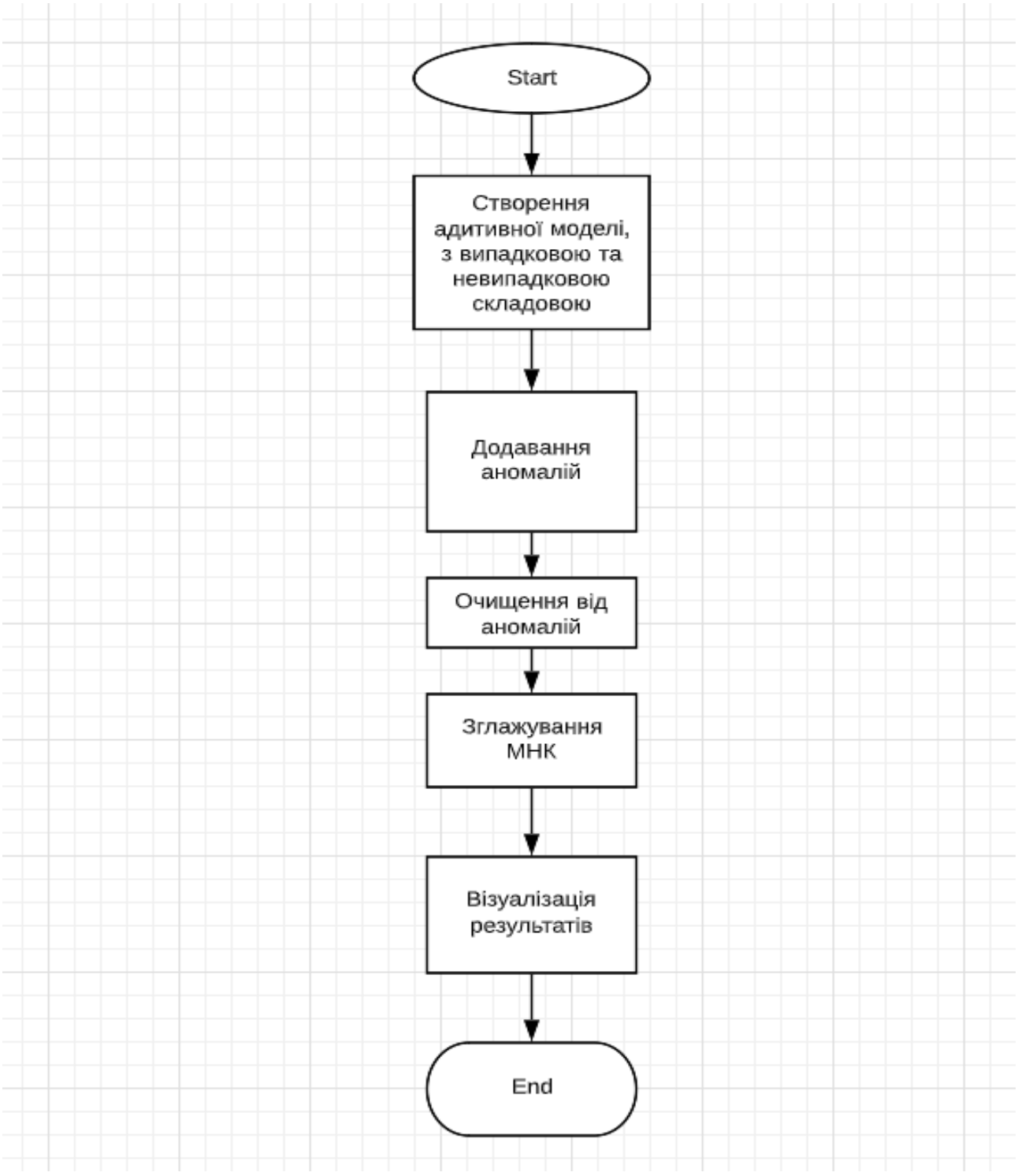
ВИКОНАННЯ

Стурктура проекту це один файл laba2.ipynb.



Не думаю, що є сенс описувати мат основу МНК, щодо мого методу, який я я буду реалізовувати для пошуку аномалій, його детальний опис буде нижче.

Діаграма:


Start
Створення адитивної моделі, з випадковою та невипадковою складовою
Додавання аномалій
Очищення від аномалій
Зглажування МНК
Візуалізація результатів
End

Перш за все вкажемо потрібні нам імпорти, та напишемо функцію, що буде виводити числові характеристики переданої вибірки.

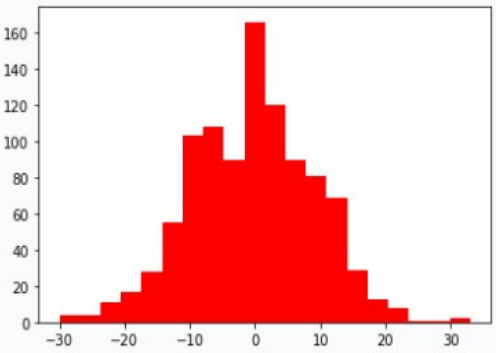
In 1
```
import numpy as np
import matplotlib.pyplot as plt
```

In 2
```
def calculate_metrics(sample: np.ndarray) -> None:
    print(f'Mathematical expectation: {np.median(sample)}')
    print(f'Dispersion: {np.var(sample).round(3)}')
    print(f'Sigma: {np.std(sample).round(3)}')
    print('=' * 50)
```

Описувати генерацію вибірки, її випадкової та невипадковою складової та побудову адитивної моделі я не буду, проходили це вже все, залишу тут лише код.

Building normal distribution

285
```
mean, sigma = 0, 10
distribution_size = 1000
normal_distribution = np.random.normal(mean, sigma, size=distribution_size).astype(int)

plt.hist(normal_distribution, bins=20, color='r')
plt.show()
calculate_metrics(normal_distribution)
```

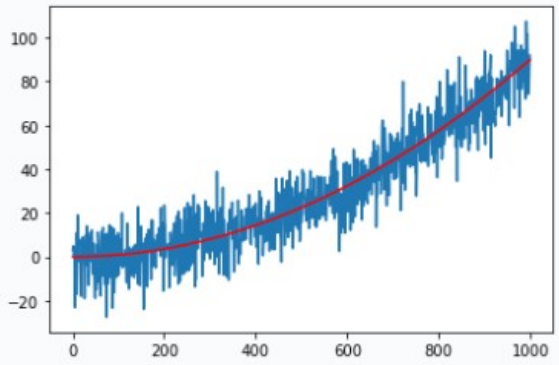



```
Mathematical expectation: 0.0
Dispersion: 87.032
Sigma: 9.329
==================================================
```

Correcting distribution

In 286

```
coef = 0.00009

correction_array = np.arange(normal_distribution.shape[0])
correction_array = (correction_array ** 2) * coef

corrected_array = normal_distribution + correction_array

plt.plot(corrected_array)
plt.plot(correction_array, color='red')
plt.show()

calculate_metrics(corrected_array)
```



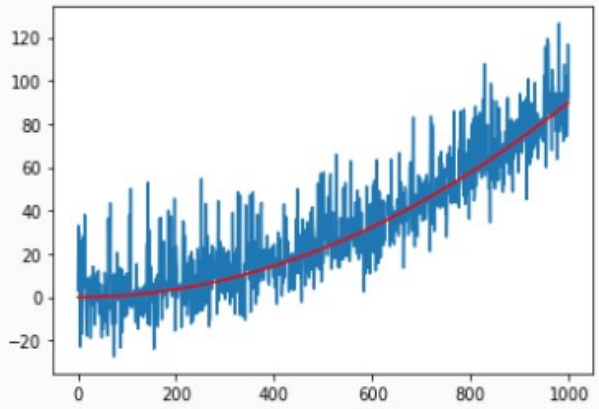

```
Mathematical expectation: 22.99333
Dispersion: 776.815
Sigma: 27.871
================================================
```

Такс, тепер я буду генерувати аномалії, аби створити 10% шанс отримання аномалії, я буду використовувати np.random.rand() ця функція повертає раптове число в межах (0, 1), відповідно, якщо помножити його на 100 отримаємо число в межах (0, 100), оскільки rand генерує рівномірний розподіл, то приблизно 10% його значень будуть <10, чим ми і скористаємось.

Adding anomalies

```
[287]
array_with_anomalies = np.zeros(corrected_array.shape[0])
for idx in range(corrected_array.shape[0]):
    # adding 10% chance of anomaly
    if np.random.rand() * 100 < 10:
        array_with_anomalies[idx] = corrected_array[idx] + 3*sigma
    else:
        array_with_anomalies[idx] = corrected_array[idx]
```

```
[288]
plt.plot(array_with_anomalies)
plt.plot(correction_array, color='red')
plt.show()

calculate_metrics(array_with_anomalies)
```




```
Mathematical expectation: 26.90833
Dispersion: 850.372
Sigma: 29.161
==================================================
```

ОПИС МЕТОДУ

Тепер щодо методу виявлення аномалій, я вирішив обратий свій метод. Коли я тільки почав працювати в IT, моєю першою роботою була позиція Data Engineer, хоча з Data Science в мене була схожість лише слово Data в назві професії, але виявлення аномалій, це була не рідка задача в рамках моїх забов’язань. В нас на проекті ми використовували дуже простий і зрозумілий метод, “Метод ядра”, щось типу того як в CNN нейронках є ядро, що бігає по картинці та “звертає” її, так само в нас було одномірне ядро, вектор, таке собі вікно, що йшло по всій вибірці і перевіряло, аби елементи в цьому вікні не були більше за середнє значення елементів ядра + якесь число (я буду юзати в лабі +3 * sigma(ядра)).

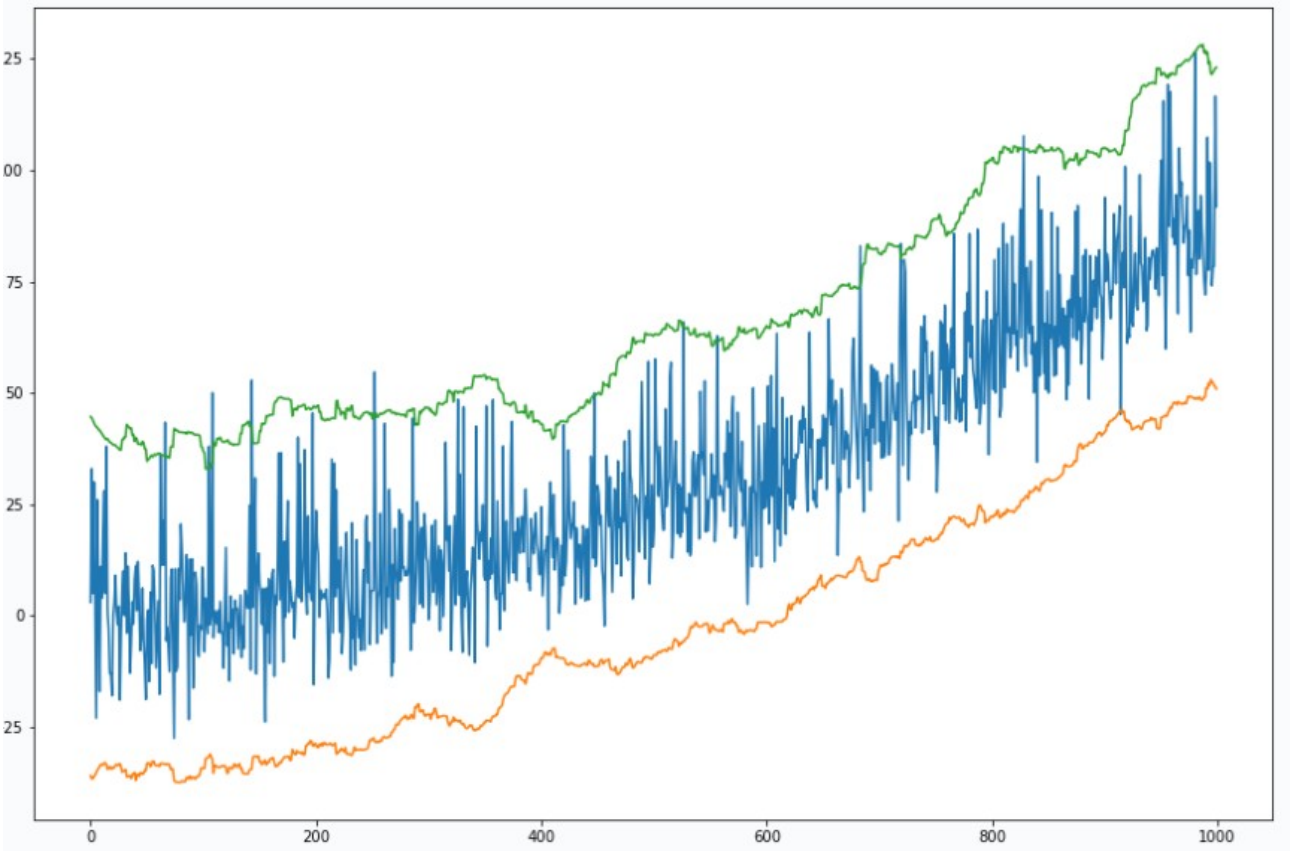
Створюємо простий метод, що буде приймати на вхід ядро та повертати максимально та мінімально допустимі значення, що можуть бути в цьому ядрі.

Разом також обираємо розмір ядра, в моєму випадку це буде 7% від розміру вибірки.

```
36
def get_thresholds(dataset: np.ndarray) -> Tuple[int, int]:
    return np.mean(dataset) + 3*np.std(dataset), np.mean(dataset) - 3*np.std(dataset)


def violate_thresholds(value: int, lt: float, ut: float) -> bool:
    if value < lt or value > ut:
        return True

    return any([math.isclose(value, threshold) for threshold in [lt, ut]])
```

Далі все доволі легко, пробігаємо нашим ядром(вікном) по масиву даних, отрмуючи трешхолди, зберігаємо їх (для подальшої візуалзації) та якщо наш елемент задовольняє трешхолдам, то зберігаємо його.

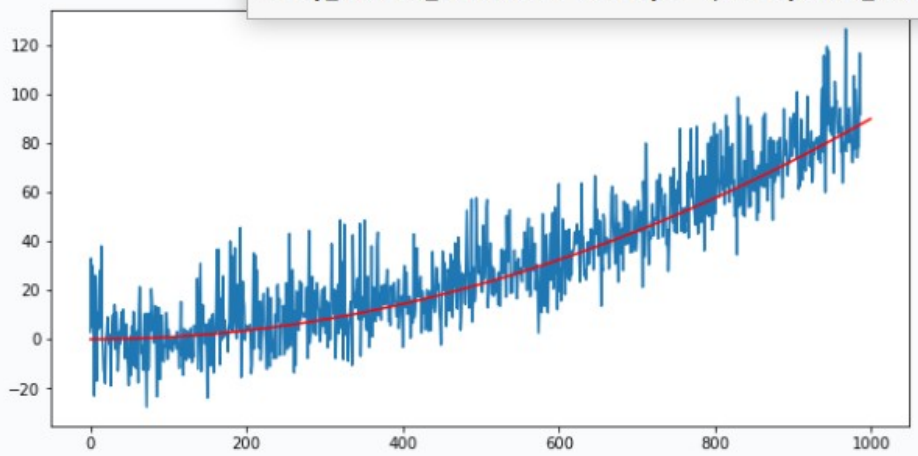
```
36
list_without_anomalies = []
upper_bounds_array = np.zeros(dataset_size)
lower_bounds_array = np.zeros(dataset_size)

for i in range(dataset_size):
    lower_bound = 0 if i - window_size//2 < 0 else i - window_size//2
    upper_bound = dataset_size if i + window_size//2 >= dataset_size else i + window_size//2
    current_window = array_with_anomalies[range(lower_bound, upper_bound)]

    upper_threshold, lower_threshold = get_thresholds(current_window)

    upper_bounds_array[i] = upper_threshold
    lower_bounds_array[i] = lower_threshold

    if lower_threshold < array_with_anomalies[i] < upper_threshold:
        list_without_anomalies.append(array_with_anomalies[i])

array_without_anomalies = np.array(list_without_anomalies)
```



А тепер приберемо аномалії, відкидаючи значення, що не попадають у проміжок +/- 3 * sigma(core)

```
2  1  plt.figure(figsize = (10,5))
   2  plt.plot(array_without_anomalies)
   3  plt.plot(correction_array, color='red')
   4  plt.show()
   5
   6  calculate_metrics(array_without_anomalies)
   7  print(f'{distribut
```

/home/kinfi4/python/kpi/DS/laba2/laba2.ipynb
array_without_anomalies: ndarray = np.array(list_without_anomalies)




```
Mathematical expectation: 26.324890000000003
Dispersion: 846.364
Sigma: 29.092
==================================================
13 anomalies were found!
```

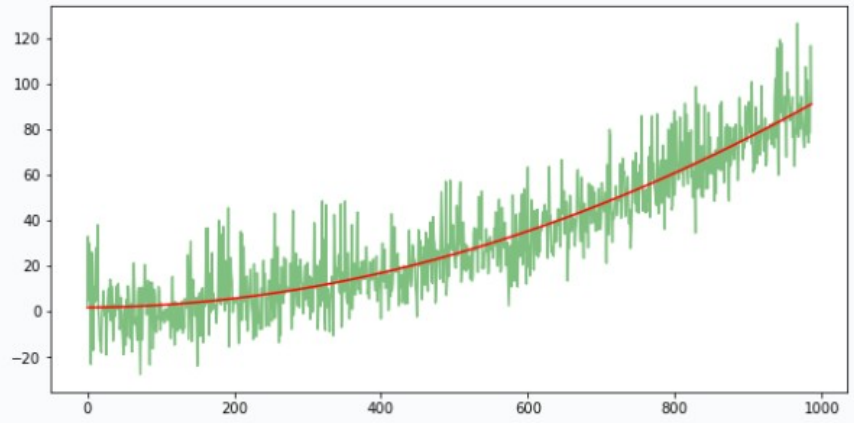
Залишилось тільки зробити Smooth графік за допомогою МНК. Я зробив маленьку функцію, що приймає на всід масив елементів, чий тренд треба знайти та повертає результат роботи МНК.

```
def lms_smoother(dataset: np.ndarray) -> np.ndarray:
    size = len(dataset)
    un_smoothed = np.zeros((size, 1))
    c = np.ones((size, 3))
    for i in range(size):
        un_smoothed[i, 0] = float(dataset[i])
        c[i, 1] = float(i)
        c[i, 2] = float(i**2)

    cT = c.T
    return c.dot(np.linalg.inv(cT.dot(c)).dot(c.T).dot(un_smoothed))
```

```
smooth_array = lms_smoother(array_without_anomalies)
```

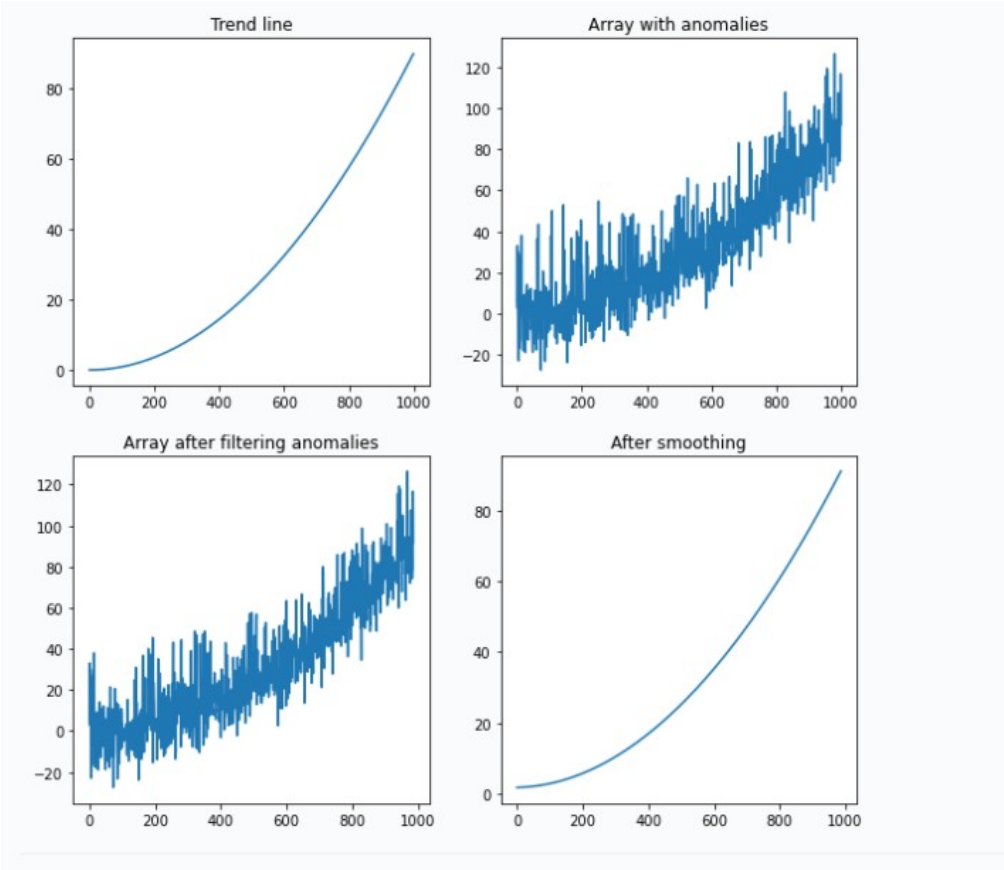
Not a rocket science, you know?

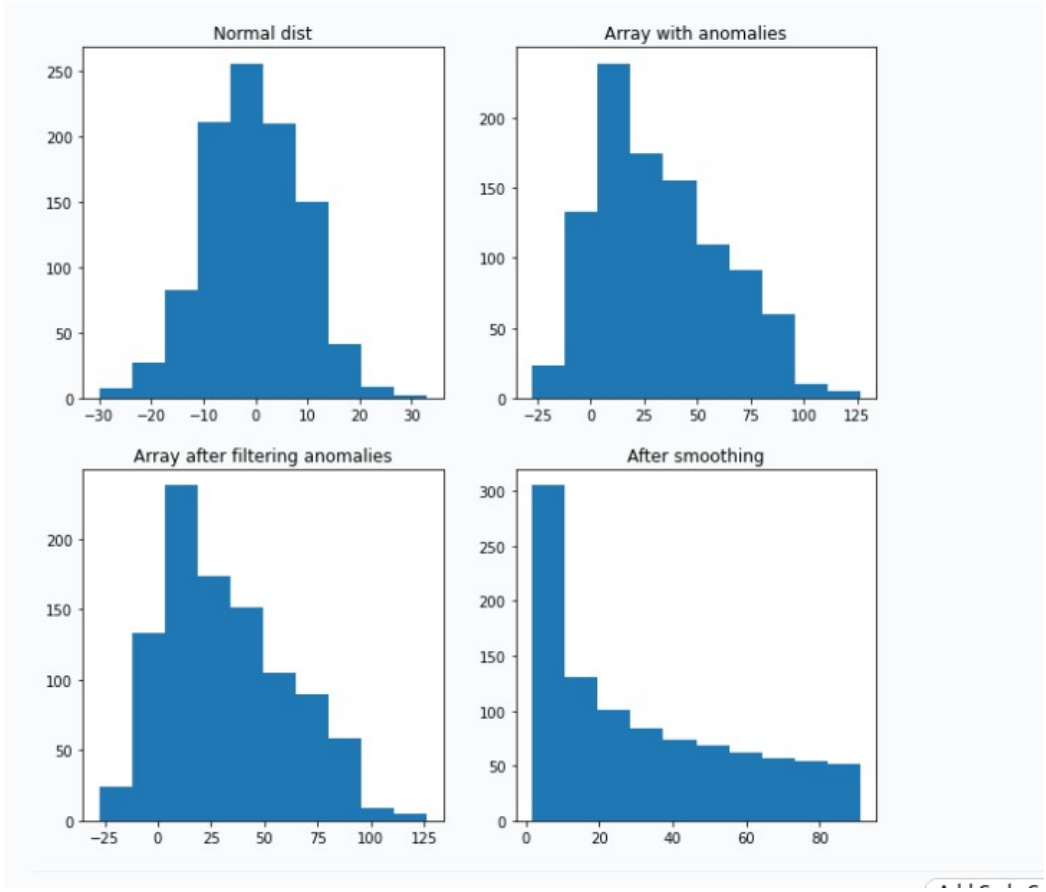
```
plt.figure(figsize = (10, 5))
plt.plot(array_without_anomalies, 'g', alpha=0.5)
plt.plot(smooth_array, 'r')
plt.show()

calculate_metrics(smooth_array)
```




```
Mathematical expectation: 24.582395191099874
Dispersion: 706.933
Sigma: 26.588
==================================================
```

Намалюємо графіки для порівняння





Результати стат характеристик

	Мат. очікування	Дисперсія	СКВ
Закон розподілу вип. похибки	0	87.03	9.33
Вхідна вибірка, без аномалій	22.99	776.82	27.87
Вхідна з аномаліями	26.91	850.37	29.16
Після очищення від аномалій	26.32	846.36	29.09
Результати МНК	24.58	706.93	26.59

ВИСНОВОК

Робота була досить велика, однак дуже цікава, ми дослідили які є способи очищення нашої вибірки від аномалій. Та як за домогою МНК подубувати smoothed версію нашої вибірки, що корисно для розуміння лінії тренду.