Ветвящиеся процессы

Работа выполнена студентом группы 23КНТ2 ИМИКН ВШЭ НН ***Власовым Артёмом Дмитриевичем***

In [99]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

np.random.seed(seed=42)
```

In [100…
```
df = pd.read_excel('Великобритания.xlsx', sheet_name='Sheet1')
df.head()
```

Out[100…

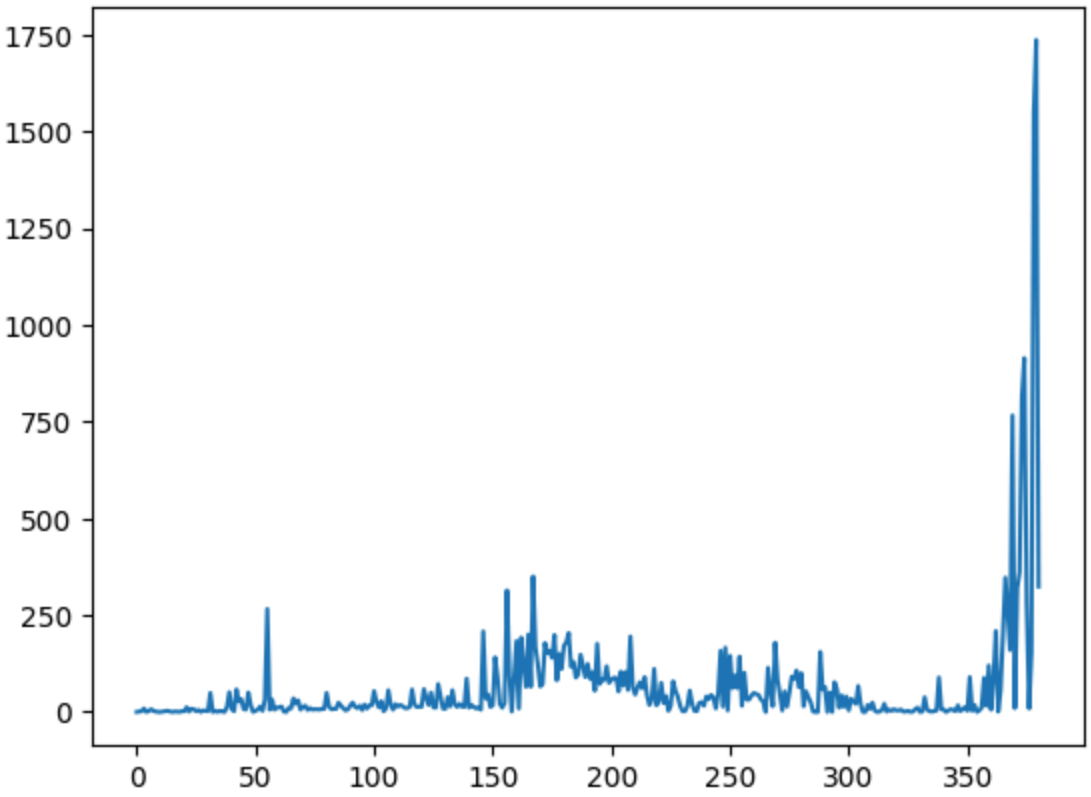
	Страна	Дата	Заражений	Выздоровлений	Смертей	Заражен за д
0	Великобритания	20.07.2020	296940	1413.0	41090	
1	Великобритания	21.07.2020	297385	1414.0	41115	
2	Великобритания	22.07.2020	297948	1416.0	41132	
3	Великобритания	23.07.2020	298727	1425.0	41141	
4	Великобритания	24.07.2020	299495	1425.0	41173	

In [101…
```
df.tail()
```

Out[101…

	Страна	Дата	Заражений	Выздоровлений	Смертей	Зараж за
604	Великобритания	16.03.2022	20059641	NaN	163833	
605	Великобритания	17.03.2022	20150847	NaN	163972	
606	Великобритания	18.03.2022	20243940	NaN	164099	
607	Великобритания	19.03.2022	20243940	NaN	164099	
608	Великобритания	20.03.2022	20243940	NaN	164099	

In [102…
```
plt.plot(df.index, df['Выздоровлений за день'])
plt.show()
```



Работаем с пропущенными значениями с использованием алгоритмов машинного обучения

Как мы видим, начиная с 381 строки в столбцах **'Выздоровлений за день'** и **'Выздоровлений'** вместо данных идут пропуски. Используем Random Forest Regressor, чтобы обучить на первых 300+ строках, а затем предсказать недостающие значения.

Перейдём к следующим шагам:

1. Построим модель случайного леса по непустым данным.
2. Применим её к строкам с пропущенными значениями.
3. Заполним пропуски предсказаниями модели.

In [103…

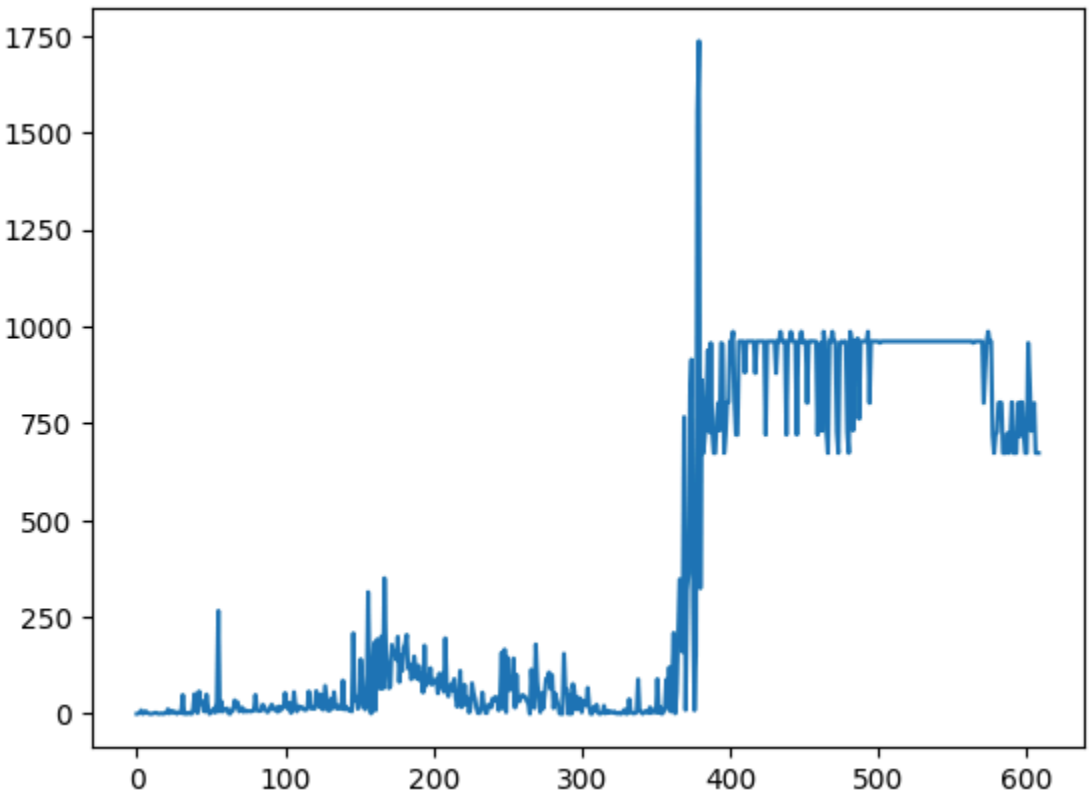
```
from sklearn.ensemble import RandomForestRegressor

def fill_daily_recovered(df):
    """
    Заполнение ежедневных выздоровлений через регрессию
```

```
    """

    df['Индекс_дня'] = df.index

    known = df[df['Выздоровлений за день'].notna()]

    X = known[['Индекс_дня', 'Тестов за день']].values
    y = known['Выздоровлений за день'].values

    # Обучаем модель
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X, y)

    all_X = df[['Индекс_дня', 'Тестов за день']].values
    predicted = model.predict(all_X)

    filled = df['Выздоровлений за день'].where(
        df['Выздоровлений за день'].notna(),
        np.round(predicted).clip(min=0)
    )
    return filled.astype(int)
```

Заполняем пропуски

In [104…

```
df['Выздоровлений за день'] = fill_daily_recovered(df)

plt.plot(df.index, df['Выздоровлений за день'])
plt.show()
```



Вычисляем значения для столбца 'Выздоровлений' начиная с 381 строки, поскольку там тоже пропуски, но зная изначальное выздоровлений и все данные о выздоровлениях в день мы можем всё вычислить

In [105…
```
start_index = 381

for i in range(start_index, len(df)):
    df.loc[i, 'Выздоровлений'] = df.loc[i-1, 'Выздоровлений'] + df.loc[i, 'Выз
```

In [106…
```
# Проверяем, нет ли пустых значений
df[df["Выздоровлений"].isnull()]
```

Out[106…

	Страна	Дата	Заражений	Выздоровлений	Смертей	Заражений за день	Выздоров з

Удаляем столбец **'Страна'**, **'Тестов'**, **'Тестов за день''**, **'Смертей за день'** и **'index'** поскольку они не несут никакой полезной информации для следующих заданий

In [107…
```
df.drop(columns=['Страна'], inplace=True)
df.drop(columns=['Тестов'], inplace=True)
df.drop(columns=['Тестов за день'], inplace=True)
df.drop(columns=['Индекс_дня'], inplace=True)
df.head()
```

Out[107…

	Дата	Заражений	Выздоровлений	Смертей	Заражений за день	Выздоровле за д
0	20.07.2020	296940	1413.0	41090	586	
1	21.07.2020	297385	1414.0	41115	445	
2	22.07.2020	297948	1416.0	41132	563	
3	23.07.2020	298727	1425.0	41141	779	
4	24.07.2020	299495	1425.0	41173	768	

1 Задание

По нормированным данным о ежедневном числе инфицированных за 30 дней найти оценку среднего числа потомков одной частицы для ветвящегося процесса, описывающего эпидемический процесс. Является ли этот процесс докритическим, критическим или надкритическим?

In [108…
```
df['I'] = df['Заражений'] - df['Выздоровлений'] - df['Смертей']
```

```
df['I_norm'] = df['I'] / (df['Население страны'] / 100000)
I = df['I_norm'].values[:30]

# Оценка среднего числа потомков (m)
I_next = I[1:]
I_current = I[:-1]
m = np.mean(I_next / I_current)
print(f"Оценка среднего числа потомков (m): {m:.4f}")
```

```
Оценка среднего числа потомков (m): 1.0032
```

In [109…
```
if m < 1:
    status = "докритический"
elif np.isclose(m, 1.0, atol=0.01):
    status = "критический"
else:
    status = "надкритический"
print(f"Процесс {status}")
```

```
Процесс критический
```

2 Задание

Найти оценку вероятности отсутствия потомков одной частицы в предположении, что число потомков одной частицы имеет геометрическое распределение.

In [110…
```
p_geom = 1 / (1 + m)
print(f"Вероятность отсутствия потомков: {p_geom:.4f}")
```

```
Вероятность отсутствия потомков: 0.4992
```

3 Задание

Найти вероятность вырождения ветвящегося процесса, описывающего эпидемический процесс, в предположении, что число потомков одной частицы имеет геометрическое распределение.

In [111…
```
from scipy.optimize import fsolve

def G(s):
    return p_geom / (1 - (1 - p_geom) * s)

def equation(s):
    return G(s) - s
```

In [112…
```
if m < 1:
    q = 1.0
```

```
else:
    q_guess = 0.9
    q = fsolve(equation, q_guess)[0]

print(f"Вероятность вырождения: {q:.4f}")
```

```
Вероятность вырождения: 0.9968
```

4 Задание

Если процесс является докритическим, найти приближенно среднее время до вырождения процесса (окончания эпидемии)

Из задания 1 мы нашли что процесс является критическим, значит в задании 4 мы ничего не высчитываем.

5 Задание

Смоделировать 5 траекторий ветвящегося процесса на интервале с 1 по 30 день наблюдений, построить соответствующие графики. Усреднить значения сгенерированных траекторий за каждый день наблюдений (с 1 по 30), построить график усредненной траектории и сравнить ее с реальными данными.

Чтобы избежать генерации массивов, состоящих из большого числа случайных величин геометрического распределения, заменим сумму этих величин одной случайной величиной, распределённой приближённо по Пуассону со средним current · m, где m — среднее число потомков. Это численно удобная апроксимация.

In [ ]:
```
def simulate_branching_poisson(m, days, n0, n_trajectories=5):
    """
    Симуляция процесса разветвляющегося Пуассона

    """
    trajectories = []
    for _ in range(n_trajectories):
        trajectory = [n0]
        for _ in range(1, days):
            current = trajectory[-1]
            offspring = np.random.poisson(current * m)
            trajectory.append(offspring)
        trajectories.append(trajectory)
    return np.array(trajectories)
```

```
# Параметры
days = 30
n0 = I[0]

trajectories = simulate_branching_poisson(m, days, n0)
mean_trajectory = trajectories.mean(axis=0)
```

In [ ]:
```
plt.figure(figsize=(12, 6))

for trajectory in trajectories:
    plt.plot(trajectory, alpha=0.5)

plt.plot(mean_trajectory, color="black", label="Усреднённая траектория", linew
plt.plot(I[:days], color="red", linestyle="--", label="Реальные данные (I)")
plt.xlabel("День")
plt.ylabel("Число инфицированных I(t)")
plt.title("Симуляция ветвящегося процесса по активным случаям")
plt.legend()
plt.grid(True)
plt.show()
```