Problem solution

Junior risk quant

Автор
Черепахин Иван Иванович
icherepaxin@bk.ru

1 Task 1

Введем из условия задачи

$$\pi_t = \mathbb{1}\{t \geqslant \tau\} - \sum_{i=1}^{t} 2^{i-1}\mathbb{1}\{t < \tau\}. \tag{1}$$

1.1 Prove that $\tau < \infty$ a.s.

Хотим показать, что τ является моментом остановки. Измеримостьτ вытекает из того, что это inf от измеримых функций. Далее рассмотрим серии эквивалентных событий $\{\tau = \infty\} \Leftrightarrow \{\pi_\infty = 1\}$, то есть мы никогда не останавливаем игру, при этом выйгрыш равен 1. На математическом языке

$$P(\tau = \infty) = P(\pi_\infty = 1) = P(\sum_{k=1}^{\infty} 2^{k-1} = 1) = 0.$$

1.2 $\mathbb{E}\pi_\tau =?$

Используем (1) и возьмем матожидание π_τ. Соответсвенно получаем $\mathbb{E}\pi_\tau = 1$.

1.3 $\mathbb{E}\pi_t =?$

Используем (1) и возьмем матожидание π_t

$$\mathbb{E}\pi_t = P(t \geqslant \tau) - \sum_{i=1}^{t} 2^{i-1} P(t < \tau) = 1 - 2^t \left(1 - P(\tau \leqslant t)\right).$$

Заметим, что распределние τ выглядит следующим образом

$$P(\tau = k) = \frac{1}{2^k}, \quad \forall k \in \mathbb{N},$$

так как из условия на каждом шаге равновероятен выйгрыш и проигрыш. Соответственно получаем $\mathbb{E}\pi_t = 0$.

1.4 Compare result

Theorem 1.1 (Дуба)**.** Пусть $X = \{X_t, t \in \mathbb{N}\}$ - мартингал и τ - момент остановки со значениями в $\mathbb{N}$ и фильтрация $\{\mathcal{F}_t\}_{t \in \mathbb{N}}$. Пусть $\exists C$ такая, что

$$|X_{min(\tau,n)}| \leqslant C \ \forall n \geqslant 0 \text{ п.н.}$$

Тогда выполнено $\mathbb{E}X_\tau = \mathbb{E}X_1$.

В нашем же случае матожидания различаются. Соответсвенно этого достаточно, что утверждать, что процесс π_t не является мартингалом.

Number 2

Now lets simulate the task. Futhemore I will estimate $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$ and $X_{(1)}$.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy.typing as npt
from typing import Callable

import statsmodels.api as sm
```

Parametrs of samples.

This helpful function is creating plot of sampels.

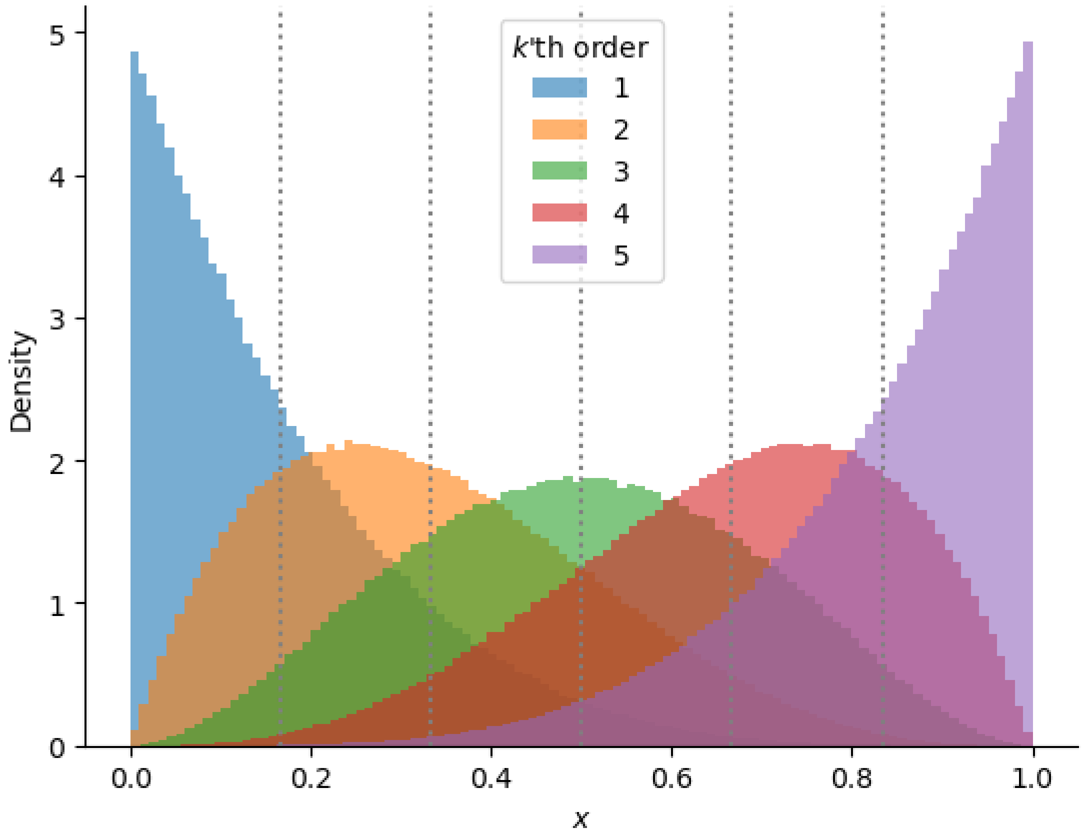
In [ ]:
```
def draw(u : np.array):
    _, ax = plt.subplots()
    for k in range(u.shape[0]):
        ax.hist(u[k, :], density=True, alpha=0.6, label=f'${k+1}$', bins=100);
    ax.legend(loc='best', title="$k$'th order")
    sns.despine();
    ax.set_xlabel('$x$')
    ax.set_ylabel('Density');
    for k in range(u.shape[0]):
        ax.axvline(u[k, :].mean(), color='gray', linestyle=':');
    plt.show()
```

2.1

In [ ]:
```
R = 1000000 # samples
N = 5 # amount of variable
a = 0
b = 1
```

In [ ]:
```
x = np.random.uniform(a,b,(N,R))
x_sort = np.sort(x, 0)
x_mean_estimate = x.mean(0)
data = {"x_mean" : list(x_mean_estimate)}
for i in range(1, N + 1):
    data["x({})".format(i)] = list(x_sort[i - 1])
data = pd.DataFrame(data)
```

In [ ]:
```
draw(x_sort)
```



In [ ]:
```
estimate = x_sort[0]
res = sm.OLS(estimate, x_sort.T).fit()
print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                      y   R-squared (uncentered):              1.0
00
Model:                            OLS   Adj. R-squared (uncentered):         1.0
00
Method:                 Least Squares   F-statistic:                     4.791e+
35
Date:                Wed, 01 Nov 2023   Prob (F-statistic):                   0.
00
Time:                        12:39:26   Log-Likelihood:                 3.5078e+
07
No. Observations:             1000000   AIC:                            -7.016e+
07
Df Residuals:                  999995   BIC:                            -7.016e+
07
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             1.0000   1.29e-18   7.74e+17      0.000       1.000       1.000
x2          2.044e-16   1.29e-18    157.956      0.000    2.02e-16    2.07e-16
x3          3.856e-16   1.29e-18    298.070      0.000    3.83e-16    3.88e-16
x4         -5.149e-17   1.29e-18    -39.765      0.000    -5.4e-17    -4.9e-17
x5         -2.739e-17   8.36e-19    -32.764      0.000    -2.9e-17   -2.57e-17
==============================================================================
Omnibus:                    40081.602   Durbin-Watson:                   0.688
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            45113.929
Skew:                          -0.520   Prob(JB):                         0.00
Kurtosis:                       2.957   Cond. No.                         15.7
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [ ]:

```
estimate = x_mean_estimate
res = sm.OLS(estimate, x_sort.T).fit()
print(res.summary())
```

```
                            OLS Regression Results
=======================================================================================
==
Dep. Variable:                      y   R-squared (uncentered):                   1.0
00
Model:                            OLS   Adj. R-squared (uncentered):              1.0
00
Method:                 Least Squares   F-statistic:                        4.631e+
36
Date:                Wed, 01 Nov 2023   Prob (F-statistic):                        0.
00
Time:                        12:40:24   Log-Likelihood:                     3.5351e+
07
No. Observations:             1000000   AIC:                               -7.070e+
07
Df Residuals:                  999995   BIC:                               -7.070e+
07
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.2000   9.83e-19   2.04e+17      0.000       0.200       0.200
x2             0.2000   9.84e-19   2.03e+17      0.000       0.200       0.200
x3             0.2000   9.84e-19   2.03e+17      0.000       0.200       0.200
x4             0.2000   9.85e-19   2.03e+17      0.000       0.200       0.200
x5             0.2000   6.36e-19   3.15e+17      0.000       0.200       0.200
==============================================================================
Omnibus:                    22567.845   Durbin-Watson:                   1.862
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            27629.430
Skew:                           0.307   Prob(JB):                         0.00
Kurtosis:                       3.534   Cond. No.                         15.7
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

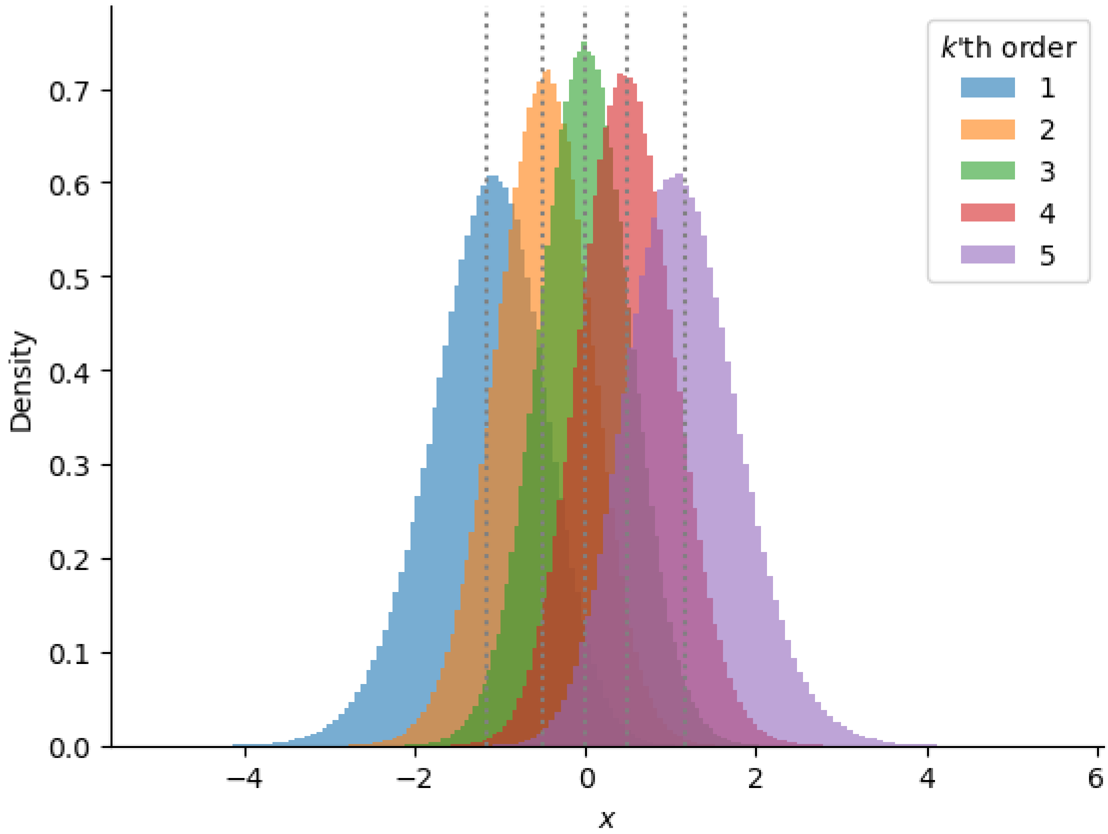
2.2

In [ ]:
```
R = 1000000 # samples
N = 5 # amount of variable

a = 0
sigma = 1
```

In [ ]:
```
x = np.random.normal(a,b,(N,R))
x_sort = np.sort(x, 0)
```

In [ ]:
```
draw(x_sort)
```



In [ ]:
```
estimate = x_sort[0]
res = sm.OLS(estimate, x_sort.T).fit()
print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                      y   R-squared (uncentered):                 1.0
00
Model:                            OLS   Adj. R-squared (uncentered):            1.0
00
Method:                 Least Squares   F-statistic:                        4.791e+
35
Date:                Wed, 01 Nov 2023   Prob (F-statistic):                      0.
00
Time:                        12:41:34   Log-Likelihood:                    3.5078e+
07
No. Observations:             1000000   AIC:                               -7.016e+
07
Df Residuals:                  999995   BIC:                               -7.016e+
07
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             1.0000   1.29e-18   7.74e+17      0.000       1.000       1.000
x2          2.044e-16   1.29e-18    157.956      0.000    2.02e-16    2.07e-16
x3          3.856e-16   1.29e-18    298.070      0.000    3.83e-16    3.88e-16
x4         -5.149e-17   1.29e-18    -39.765      0.000    -5.4e-17    -4.9e-17
x5         -2.739e-17   8.36e-19    -32.764      0.000    -2.9e-17   -2.57e-17
==============================================================================
Omnibus:                    40081.602   Durbin-Watson:                   0.688
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            45113.929
Skew:                          -0.520   Prob(JB):                         0.00
Kurtosis:                       2.957   Cond. No.                         15.7
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [ ]:

```
estimate = x_mean_estimate
res = sm.OLS(estimate, x_sort.T).fit()
print(res.summary())
```

```
                            OLS Regression Results
=======================================================================================
==
Dep. Variable:                      y   R-squared (uncentered):                   1.0
00
Model:                            OLS   Adj. R-squared (uncentered):              1.0
00
Method:                 Least Squares   F-statistic:                          4.631e+
36
Date:                Wed, 01 Nov 2023   Prob (F-statistic):                        0.
00
Time:                        12:42:28   Log-Likelihood:                       3.5351e+
07
No. Observations:             1000000   AIC:                                  -7.070e+
07
Df Residuals:                  999995   BIC:                                  -7.070e+
07
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.2000   9.83e-19   2.04e+17      0.000       0.200       0.200
x2             0.2000   9.84e-19   2.03e+17      0.000       0.200       0.200
x3             0.2000   9.84e-19   2.03e+17      0.000       0.200       0.200
x4             0.2000   9.85e-19   2.03e+17      0.000       0.200       0.200
x5             0.2000   6.36e-19   3.15e+17      0.000       0.200       0.200
==============================================================================
Omnibus:                    22567.845   Durbin-Watson:                   1.862
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            27629.430
Skew:                           0.307   Prob(JB):                         0.00
Kurtosis:                       3.534   Cond. No.                         15.7
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

2.3

In [ ]:
```
def draw_for_pois(u : np.array):
    _, ax = plt.subplots()

    for k in range(u.shape[0]):
        mass = pd.DataFrame(np.unique(u[k, :], return_counts=True))
        ax.plot(mass.iloc[1], alpha=0.6, label=f'${k+1}$')
        # ax.hist(, density=True, alpha=0.6, label=f'${k+1}$', bins=100);
    ax.legend(loc='best', title="$k$'th order")
    sns.despine();
    ax.set_xlabel('$x$')
    ax.set_ylabel('CDF');
    for k in range(u.shape[0]):
        ax.axvline(u[k, :].mean(), color='gray', linestyle=':');
    plt.show()
```

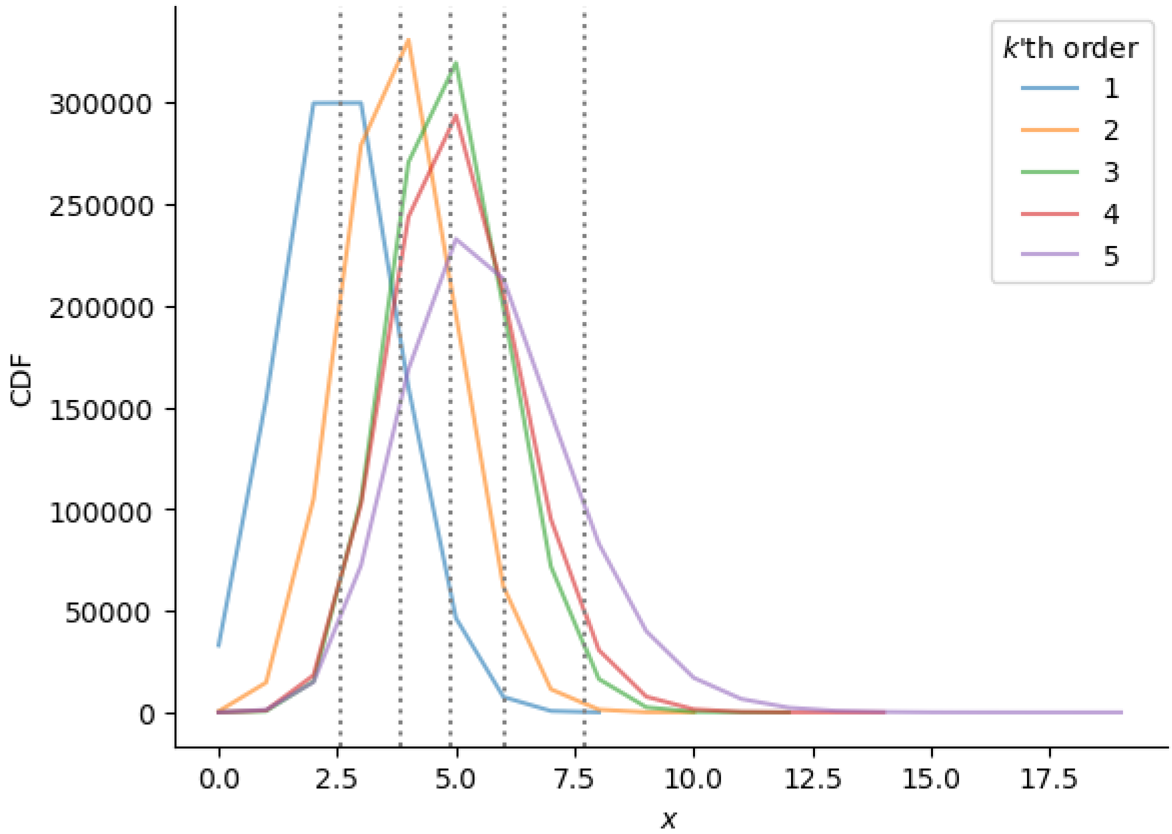
In [ ]:
```
R = 1000000 # samples
N = 5 # amount of variable

_lambda = 5
```

In [ ]:
```
x = np.random.poisson(_lambda,(N,R))
x_sort = np.sort(x, 0)
```

```
x_mean_estimate = x.mean(0)
```

In [ ]:
```
draw_for_pois(x_sort)
```



In [ ]:
```
estimate = x_sort[0]
res = sm.OLS(estimate, x_sort.T).fit()
print(res.summary())
```

```
                            OLS Regression Results                            
=======================================================================================
==
Dep. Variable:                      y   R-squared (uncentered):                   1.0
00
Model:                            OLS   Adj. R-squared (uncentered):              1.0
00
Method:                 Least Squares   F-statistic:                          4.791e+
35
Date:                Wed, 01 Nov 2023   Prob (F-statistic):                        0.
00
Time:                        12:43:05   Log-Likelihood:                      3.5078e+
07
No. Observations:             1000000   AIC:                                 -7.016e+
07
Df Residuals:                  999995   BIC:                                 -7.016e+
07
Df Model:                           5                                                  
Covariance Type:            nonrobust                                                  
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             1.0000   1.29e-18   7.74e+17      0.000       1.000       1.000
x2          2.044e-16   1.29e-18    157.956      0.000    2.02e-16    2.07e-16
x3          3.856e-16   1.29e-18    298.070      0.000    3.83e-16    3.88e-16
x4         -5.149e-17   1.29e-18    -39.765      0.000    -5.4e-17    -4.9e-17
x5         -2.739e-17   8.36e-19    -32.764      0.000    -2.9e-17   -2.57e-17
==============================================================================
Omnibus:                    40081.602   Durbin-Watson:                   0.688
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            45113.929
Skew:                          -0.520   Prob(JB):                         0.00
Kurtosis:                       2.957   Cond. No.                         15.7
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [ ]:

```
estimate = x_mean_estimate
res = sm.OLS(estimate, x_sort.T).fit()
print(res.summary())
```

```
                            OLS Regression Results
=======================================================================================
==
Dep. Variable:                      y   R-squared (uncentered):                   1.0
00
Model:                            OLS   Adj. R-squared (uncentered):              1.0
00
Method:                 Least Squares   F-statistic:                        4.631e+
36
Date:                Wed, 01 Nov 2023   Prob (F-statistic):                       0.
00
Time:                        12:43:16   Log-Likelihood:                    3.5351e+
07
No. Observations:             1000000   AIC:                               -7.070e+
07
Df Residuals:                  999995   BIC:                               -7.070e+
07
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.2000   9.83e-19   2.04e+17      0.000       0.200       0.200
x2             0.2000   9.84e-19   2.03e+17      0.000       0.200       0.200
x3             0.2000   9.84e-19   2.03e+17      0.000       0.200       0.200
x4             0.2000   9.85e-19   2.03e+17      0.000       0.200       0.200
x5             0.2000   6.36e-19   3.15e+17      0.000       0.200       0.200
==============================================================================
Omnibus:                    22567.845   Durbin-Watson:                   1.862
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            27629.430
Skew:                           0.307   Prob(JB):                         0.00
Kurtosis:                       3.534   Cond. No.                         15.7
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

In [ ]:

Number 3

Derivation

Assume given a filtered probability space $(\Omega, \mathscr{F}, (\mathscr{F}_t)_{t\in[0,T]}, \mathrm{P})$, where the filtration is generated by a Brownian motion W_t.

The market consists of two assets: a *riskless asset* with price B_t (e.g. money market account) and a *risky asset* with price S_t (e.g. FX Spot) given by a geometric Brownian motion:

$$\begin{aligned} dB_t &= rB_t dt, \qquad & B_0 &= 1, \qquad & (B_t &= e^{rt}), \\ dS_t &= S_t(\mu dt + \sigma dW_t), \qquad & S_0 &> 0, \qquad & \left(S_t \right.&\left.= S_0 e^{\sigma W_t + \left(\mu - \frac{\sigma^2}{2}\right)t}\right). \end{aligned}$$

Also denote some parametrs:

1. r_f - foreign risk free rate,
2. K - the FX strike rate,
3. T - maturity period.

And the model Garman-Kohlhagen has similar state like Black-Scholes model. The call and put price can be calculated by formula below:

$$V^{\text{call}} = Se^{-r_f T}\Phi(d_1) - e^{-rT}K\Phi(d_2), \quad V^{\text{put}} = e^{-rT}K\Phi(-d_2) - Se^{-r_f T}\Phi(-d_1),$$

where $\Phi(x)$ is the standard normal distribution function, and

$$d_1 = \frac{1}{\sigma\sqrt{T}}\left(\ln\frac{S}{K} + \left(r - r_f + \frac{\sigma^2}{2}\right)T\right), d_2 = \frac{1}{\sigma\sqrt{T}}\left(\ln\frac{S}{K} + \left(r - r_f - \frac{\sigma^2}{2}\right)T\right).$$

The greeks

Delta

$$\Delta^{call} = \frac{\partial V^{call}}{\partial S} = e^{-r_f T}\Phi(d_1)$$

Gamma

$$\Gamma = \frac{\partial^2 V}{\partial S^2} = \frac{e^{-r_f T}\Phi'(d_1)}{S\sigma\sqrt{T-t}}$$

Theta

$$\theta^{call} = \frac{\partial V^{call}}{\partial t} = r_f S e^{-r_f(T-t)}\Phi(d_1) - Se^{-r_f(T-t)}\frac{\Phi'(d_1)\sigma}{2\sqrt{T-t}} - rKe^{-r(T-t)}\Phi(d_2).$$

In [ ]:

```python
from typing import Union
from dataclasses import dataclass
import numpy as np
import numpy.typing as npt
from scipy import stats
```

In [ ]:

```python
FloatArray = npt.NDArray[np.float_]
Floats = Union[float, FloatArray]
```

Here we make support class for future application. Names of classes is meaning what it names)

In [ ]:

```python
@dataclass
class MarketState:
    spot_price: Floats
    domestic_rate: Floats
    time: Floats = 0

@dataclass
class StockOption:
    strike_price: Floats
    expiration_time: Floats  # in years
    is_call: Union[bool, npt.NDArray[np.bool_]]

    def payoff(self, spot_price: Floats) -> Floats:
        call_payoff = np.maximum(0, spot_price - self.strike_price)
        put_payoff = np.maximum(0, self.strike_price - spot_price)
        return np.where(self.is_call, call_payoff, put_payoff)

@dataclass
class CallStockOption(StockOption):
    def __init__(self, strike_price, expiration_time):
        super().__init__(strike_price, expiration_time, True)

@dataclass
class PutStockOption(StockOption):
    def __init__(self, strike_price, expiration_time):
        super().__init__(strike_price, expiration_time, False)

@dataclass
class GKParams:
    volatility: Floats
    foreign_rate: Floats
```

This block is realisation of d_1 and d_2 by formulas under. And `dt` function is helpful function for correct pricing becouse we can divide one zero at the formula.

In [ ]:

```python
def dt(option: StockOption, ms: MarketState):
    return np.maximum(option.expiration_time - ms.time, np.finfo(np.float64).eps)


def d1(option: StockOption, ms: MarketState, params: GKParams):
    return 1 / (params.volatility * np.sqrt(dt(option, ms)))\
               * (np.log(ms.spot_price / option.strike_price)
                  + (ms.domestic_rate - params.foreign_rate + params.volatility ** 2


def d2(option: StockOption, ms: MarketState, params: GKParams):
    return 1 / (params.volatility * np.sqrt(dt(option, ms)))\
               * (np.log(ms.spot_price / option.strike_price)
                  + (ms.domestic_rate - params.foreign_rate - params.volatility ** 2
    # return d1(option, ms, params) - params.volatility * np.sqrt(dt(option, ms))
```

This block is realisation of price of put/call options by formulas under.

```
In [ ]: def price(option: StockOption, ms: MarketState, params: GKParams):
            discount_factor = np.exp(-ms.domestic_rate * (dt(option, ms)))
            foreign_factor = np.exp(-params.foreign_rate * (dt(option, ms)))


            call_price = stats.norm.cdf(d1(option, ms, params),0.0,1.0) * foreign_factor * ms
                    - stats.norm.cdf(d2(option, ms, params),0.0,1.0) * option.strike_price *
            put_price = stats.norm.cdf(-d2(option, ms, params), 0.0,1.0) * option.strike_pric
                - stats.norm.cdf(-d1(option, ms, params), 0.0,1.0) * ms.spot_price * foreign_

            return np.where(option.is_call, call_price, put_price)
```

This block is realisation of greeks by formulas under.

```
In [ ]: def delta(option: StockOption, ms: MarketState, params: GKParams):
            # nd1 = stats.norm.cdf(d1(option, ms, params))*np.exp(-params.foreign_rate * opti
            # return np.where(option.is_call, nd1, nd1 - 1)
            return stats.norm.cdf(d1(option, ms, params))*np.exp(-params.foreign_rate * optio



        def gamma(option: StockOption, ms: MarketState, params: GKParams):
            return stats.norm.pdf(d1(option, ms, params))*np.exp(-params.foreign_rate * optio


        def theta(option: StockOption, ms: MarketState, params: GKParams):
            foreign_discont = np.exp(-params.foreign_rate  * (dt(option, ms)))
            a = params.foreign_rate * ms.spot_price * foreign_discont * stats.norm.cdf(d1(opt
            b = ms.spot_price * stats.norm.pdf(d1(option, ms, params)) * foreign_discont * pa
                / (2 * np.sqrt(dt(option, ms)))

            d_discount_factor = ms.domestic_rate * np.exp(-ms.domestic_rate * (dt(option, ms)

            call_theta = a - b - option.strike_price * d_discount_factor * stats.norm.cdf(d2(
            return call_theta
            # return np.where(option.is_call, call_theta, put_theta)
```

Now we can launch our model and, at the first, let's set some parametrs.

```
In [ ]: spot = 1.0581
        strikes = np.array((0.9*spot, 1.1*spot))
        matur = 1
        dom_rate=2.7
        for_rate=3
        vol=6

        calls = CallStockOption(strike_price=strikes, expiration_time=matur)

        ms = MarketState(spot_price=spot, domestic_rate=dom_rate)

        params = GKParams(volatility=vol, foreign_rate=for_rate)

        print("Price of options: ", price(calls, ms, params))
        print("Delta : ", delta(calls, ms, params))
        print("Gamma : ", gamma(calls, ms, params))
```

```
Price of options:  [0.05252301 0.05250671]
Delta :  [0.04971234 0.0497038 ]
Gamma :  [3.82877988e-05 4.22592126e-05]
```

Now let describe p.2. We need to solve system of equation:

$$
\begin{cases}
\Delta_1^{call} N_1^* - \Delta_2^{call} N_2^* - N_3^* = 0 \\
\Gamma_1 N_1^* - \Gamma_2 N_2^* = 0 \\
N_1^* + N_2^* + N_3^* = N
\end{cases}
$$

In [ ]:
```
N = 1000

first_row = np.append(delta(calls, ms, params), 1)
first_row[1] = -first_row[1]
first_row[2] = -first_row[2]

second_row = np.append(gamma(calls, ms, params), 0)
second_row[1] = -second_row[1]

A = np.array([first_row,
              second_row,
              [1,1,1]
])
b = np.array([0, 0, N])

propotion_of_portf = np.linalg.solve(A, b) / N
print("Propotions: ", propotion_of_portf)
```

```
Propotions:  [0.52336782 0.47418304 0.00244914]
```

Let's make p.3.

In [ ]:
```
budget = 100000
price_portf = np.append(price(calls, ms, params), spot)
theta_of_options = np.append(theta(calls, ms, params), 0)

amount_in_portfolio = budget / np.sum(price_portf*propotion_of_portf)
propotion_of_portf *= amount_in_portfolio
propotion_of_portf[1] = -propotion_of_portf[1]

theta_portf = np.dot(theta_of_options,propotion_of_portf)

print("Theta of portfolio: ", theta_portf)
```

```
Theta of portfolio:  14138.894547928823
```

At the end, we make fix many of paramaters in the model and this way isn't right , althought volatility is changable. In real life we can get implied vol from model and market data. Also for FX option there are another model(SABR, local vol) for more suffitient result.