CQF Exam One

January 2024 Cohort

In [30]:
```
import warnings
import math
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import norm


np.set_printoptions(precision=3)
pd.set_option('display.precision', 4)
warnings.filterwarnings('ignore')
```

Question 1

Let us define the notation to be as follows:

$$R = \begin{pmatrix} 1 & 0.3 & 0.3 & 0.3 \\ 0.3 & 1 & 0.6 & 0.6 \\ 0.3 & 0.6 & 1 & 0.6 \\ 0.3 & 0.6 & 0.6 & 1 \end{pmatrix}, \quad 1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad \mu = \begin{pmatrix} 0.02 \\ 0.07 \\ 0.15 \\ 0.20 \end{pmatrix}, \quad \sigma = \begin{pmatrix} 0.05 \\ 0.12 \\ 0.17 \\ 0.25 \end{pmatrix}, \quad S = \begin{pmatrix}$$

where $\sum_{i=1}^{4} w_i = 1$.

Question 1.1

Consider the min-variance-portfolio with target return m.

1.1.1: The Langrangian for this problem is defined by

$$L(w, \lambda, \gamma) = \frac{1}{2} w^T \Sigma w + \lambda \left(m - w^T \mu \right) + \gamma \left(1 - w^T 1 \right)$$

, where $\Sigma = SRS$. The partial derivatives are given by

$$\frac{\partial L}{\partial w} = \Sigma w - \lambda \mu - \gamma 1$$

$$\frac{\partial L}{\partial \lambda} = m - w^T \mu$$

$$\frac{\partial L}{\partial \gamma} = 1 - w^T 1$$

.

1.1.2: The optimal weight allocation vector w^* (derived on sl. 66/142 ff. of the lecture notes "Fundamentals of Optimimization and Application to Portfolio Selection") is given by

$$w^* = \Sigma^{-1}(\lambda\mu + \gamma 1)$$

,

where $\lambda = \frac{Am-B}{AC-B^2}, \gamma = \frac{C-Bm}{AC-B^2}$ for the real numbers
$A = 1^T\Sigma^{-1}1, B = \mu^T\Sigma^{-1}1 = 1^T\Sigma^{-1}\mu, C = \mu^T\Sigma^{-1}\mu.$

w^* can then also be written as

$$w^* = \frac{1}{AC-B^2}\Sigma^{-1}\left[(A\mu - B1)\,m + C1 - B\mu\right]$$

.

1.1.3: Computing w^* and $\sigma_\Pi = \sqrt{w^T\Sigma w}$ for $m = 4.5\%$:

First, let us calculate w^* using Python Numpy.

In [31]:
```
mu = np.array([[0.08, 0.10, 0.10, 0.14]]).T
ones = np.ones((4, 1))
std_dev = np.array([0.12, 0.12, 0.15, 0.20]).T
R = np.array([[1, 0.3, 0.3, 0.3],
              [0.3, 1, 0.6, 0.6],
              [0.3, 0.6, 1, 0.6],
              [0.3, 0.6, 0.6, 1]])

def get_covariance_matrix(rho):
    return np.outer(std_dev, std_dev) * rho  # this is the same like std_dev * std_


def get_inverse_covariance_matrix(covariance):
    try:
        return np.linalg.inv(covariance)
    except np.linalg.LinAlgError as exc:
        print(f"Covariance matrix not invertible: {exc}")


def get_a_b_c(covariance_inv):  # slide 68
    a = np.linalg.multi_dot([ones.T, covariance_inv, ones])
    b = np.linalg.multi_dot([mu.T, covariance_inv, ones])
    c = np.linalg.multi_dot([mu.T, covariance_inv, mu])
    return a.item(), b.item(), c.item()  # item converts the np.array elements back


def get_optimized_weights(rho, target_return):  # sl. 69
    covariance = get_covariance_matrix(rho)
    covariance_inv = get_inverse_covariance_matrix(covariance)
    a, b, c = get_a_b_c(covariance_inv)
    optimized_weights = np.dot(covariance_inv, (a * mu - b * ones) * target_return
    return optimized_weights / sum(optimized_weights)  # normalize weights so they


cov = get_covariance_matrix(R)
cov
```

Out[31]:
```
array([[0.014, 0.004, 0.005, 0.007],
       [0.004, 0.014, 0.011, 0.014],
       [0.005, 0.011, 0.022, 0.018],
       [0.007, 0.014, 0.018, 0.04 ]])
```

In [32]:
```
cov_inv = get_inverse_covariance_matrix(cov)
cov_inv
```

Out[32]:
```
array([[ 79.159, -10.794,  -8.636,  -6.477],
       [-10.794, 127.735, -36.701, -27.526],
       [ -8.636, -36.701,  81.75 , -22.021],
       [ -6.477, -27.526, -22.021,  45.984]])
```

In [33]:
```
weights = get_optimized_weights(R, 0.045)
```

In [34]:
```
print(f"The weight allocation vector is\n{weights}.")
```

```
The weight allocation vector is
[[ 0.975]
 [ 0.558]
 [ 0.355]
 [-0.888]].
```

For the portfolio risk $\sigma_{\Pi} = \sqrt{w^T \Sigma w}$, we obtain:

In [35]:
```
sigma_pf = np.sqrt(np.linalg.multi_dot([np.transpose(weights), cov, weights]).item(
print(f"The portfolio risk equals {sigma_pf}.")
```

```
The portfolio risk equals 0.1648132614178332.
```

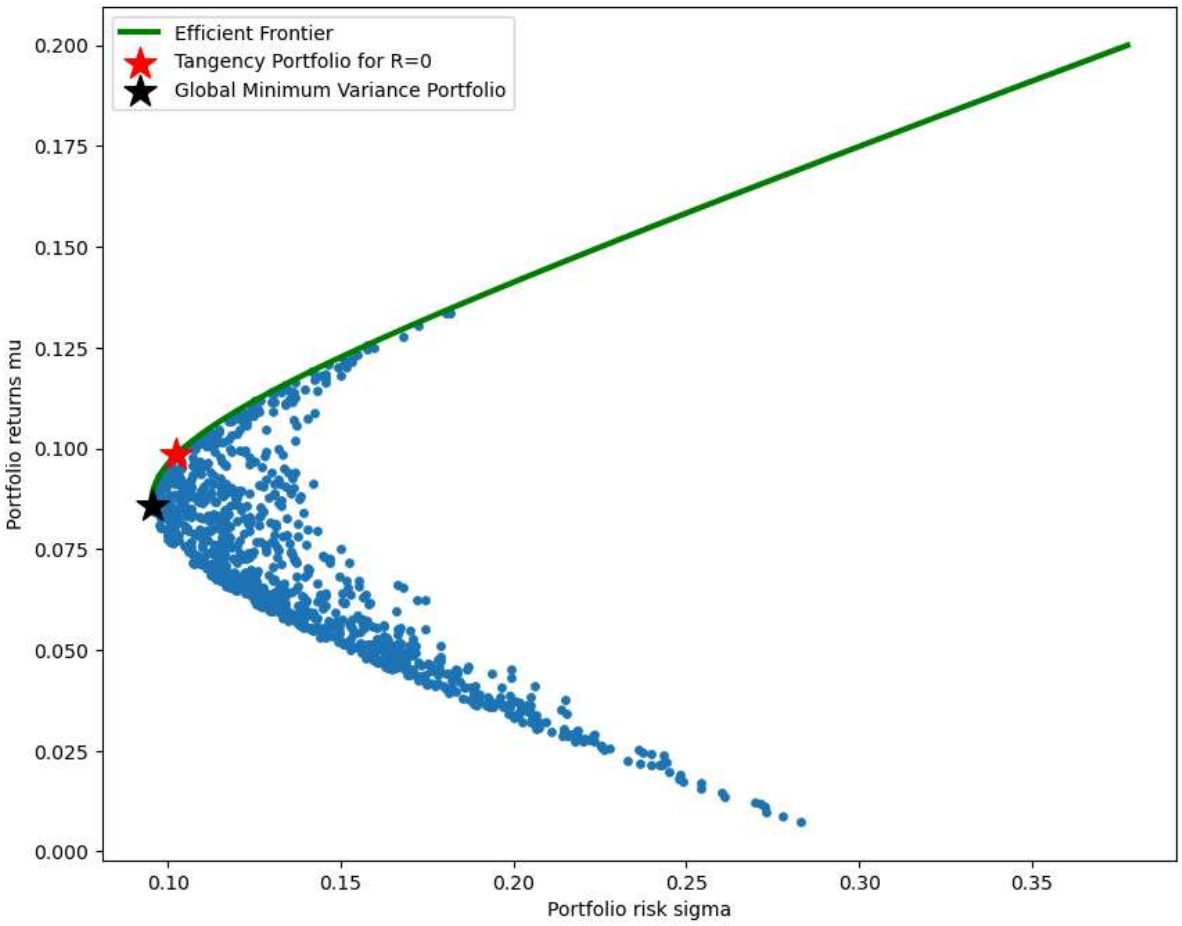
Question 1.2

Calculate + plot portfolio risk and return for 700+ random weight vectors.

In [36]:
```
def generate_random_weights(number_of_assets, sample_size):
    # generate a matrix with random weights of dimensions (number_of_assets) x (sam
    random_weights = np.random.rand(number_of_assets, sample_size)
    # ensure the weights (column vectors of the matrix) all sum up to one by normal
    random_weights[(number_of_assets-1), :] = 1 - random_weights[:(number_of_assets
    return random_weights


def get_pf_risk_and_return(covariance_matrix, weights, mu):
    pf_risk = np.sqrt(np.linalg.multi_dot([weights.T, covariance_matrix, weights]).
    pf_return = np.dot(weights.T, mu)
    return pf_risk, pf_return


def calculate_efficient_frontier(rho, covariance_matrix, target_returns, mu):
    pf_risks_ef = []
    pf_returns_ef = []
    for target_return in target_returns:
        optimized_weight = get_optimized_weights(rho, target_return)
        pf_risk_ef, pf_return_ef = get_pf_risk_and_return(covariance_matrix, optimi
        pf_risks_ef.append(pf_risk_ef)
        pf_returns_ef.append(pf_return_ef)
    return pf_risks_ef, pf_returns_ef
```

In [37]:
```
# sample size
N = 1000
number_of_assets = 4
```

In [38]:
```
# calculate necessary plot arguments
random_weights = generate_random_weights(number_of_assets, N)
pf_risks = np.zeros(N)
pf_returns = np.zeros(N)
sharpe_ratio = 0.
k_tangency = -1
global_min_risk = 1000.
k_global_min_risk = -1

# calculate portfolio risk and returns, tangency pf + GMVP
for k in range(N):
    weights = random_weights[:, k]
    pf_risks[k], pf_returns[k] = get_pf_risk_and_return(cov, weights, mu)
    ratio = pf_returns[k] / pf_risks[k]
    if ratio > sharpe_ratio:
        sharpe_ratio = ratio
        k_tangency = k
    if pf_risks[k] < global_min_risk:
        global_min_risk = pf_risks[k]
        k_global_min_risk = k

# calculate the efficient frontier
ef_number_of_points = 100
target_returns = np.linspace(pf_returns[k_global_min_risk], 0.2, ef_number_of_point
pf_risks_ef, pf_returns_ef = calculate_efficient_frontier(R, cov, target_returns, m


# plot
plt.figure(figsize=(10, 8))
plt.plot(np.reshape(pf_risks_ef, (ef_number_of_points, )), np.reshape(pf_returns_ef
color="green", label="Efficient Frontier", linewidth=3, zorder=6)
plt.scatter(x=pf_risks, y=pf_returns, marker="o", s=15, zorder=5)
plt.scatter(x=pf_risks[k_tangency], y=pf_returns[k_tangency],
            marker="*", color="red", s=300, label="Tangency Portfolio for R=0", zor
plt.scatter(x=pf_risks[k_global_min_risk], y=pf_returns[k_global_min_risk],
            marker="*", color="black", s=300, label="Global Minimum Variance Portfo
plt.xlabel("Portfolio risk sigma")
plt.ylabel("Portfolio returns mu")
plt.legend()
plt.show()
```



Discussion of above plot: Above plot displays all 1000 coordinates of (x/y) = (portfolio_risk / portfolio_return) in a scatter plot. As a result, we see a hyperbola for the opportunity set of our four risky assets. As the weights were randomly generated, we can see that some of the random portfolio allocations do not make much sense economically, because one could choose a more optimal weight allocation of assets 1, 2, 3 and 4 with less risk at the same return (along horizontal lines of the plot) or same risk with more return (along vertical lines of the plot).

I have also marked the sample portfolio with the best risk-to-return ratio with a red star and the sample portfolio coresponding to the least risky weight allocation with a black star (global minimum variance portfolio). The black one is the left-most point of all scattered coordinates and therefore has the lowest value of portfolio risk σ. The black star point can be seen as the starting point of the efficient frontier line, which is the asymptotic line on the upper half of the hyperbola where the risk profile is always optimized for any given return (or vice versa). This line is also plotted in green.

Question 2 - VaR + ES

Denote the vector of partial derivatives by

$$\frac{\partial \mathrm{VaR}(w)}{\partial w} = \left(\frac{\partial \mathrm{VaR}(w)}{\partial w_1}, \frac{\partial \mathrm{VaR}(w)}{\partial w_2}, \frac{\partial \mathrm{VaR}(w)}{\partial w_3} \right)^T, \quad \frac{\partial \mathrm{ES}(w)}{\partial w} = \left(\frac{\partial \mathrm{ES}(w)}{\partial w_1}, \frac{\partial \mathrm{ES}(w)}{\partial w_2}, \frac{\partial \mathrm{ES}(w)}{\partial w_3} \right)^T.$$

In [11]:
```
# Define all vector + matrices
mu = np.array([[0., 0., 0.]]).T
std_dev = np.array([0.3, 0.2, 0.15]).T
```

```
w = np.array([[0.5, 0.2, 0.3]]).T
rho = np.array([[1, 0.8, 0.5],
                [0.8, 1, 0.3],
                [0.5, 0.3, 1]])

# Calculating covariance
covariance = np.outer(std_dev, std_dev) * rho

# Confidence level
alpha = 0.99
factor = norm.ppf(1 - alpha)

# Calculating VaR
var = np.dot(w.T, mu) + np.dot(np.dot(factor, covariance), w) / np.sqrt(np.dot(np.d

# Calculating ES
es = np.dot(w.T, mu) - np.dot(np.dot(norm.pdf(factor), covariance), w) / (np.dot((1
```

Then, $\frac{\partial \text{VaR}(w)}{\partial w}$ and $\frac{\partial \text{ES}(w)}{\partial w}$ equal

In [12]:
```
var
```

Out[12]:
```
array([[-0.684],
       [-0.387],
       [-0.221]])
```

In [13]:
```
es
```

Out[13]:
```
array([[-0.783],
       [-0.443],
       [-0.253]])
```

Question 3 - Binomial Model

Implement the multi-step binomial model with parameters $S = 100, r = 0.05$ continuously compounded for a European call option with strike $E = 100$, maturity $T = 1$.

3.1: A suitable parametrizatio for uS, vS using continuously compunded interest can be defined as

$$uS = \exp(\sigma\sqrt{\delta t})S,$$

$$vS = \exp(-\sigma\sqrt{\delta t})S.$$

3.2: Following codes will be using (adjusted) snippets from the Python lab about Binomial Models, as well as make use of the knowledge gained from the lecture on Binomial Models

In [14]:
```
# specify the required variables
S = 100
r = 0.05
E = 100
T = 1
```

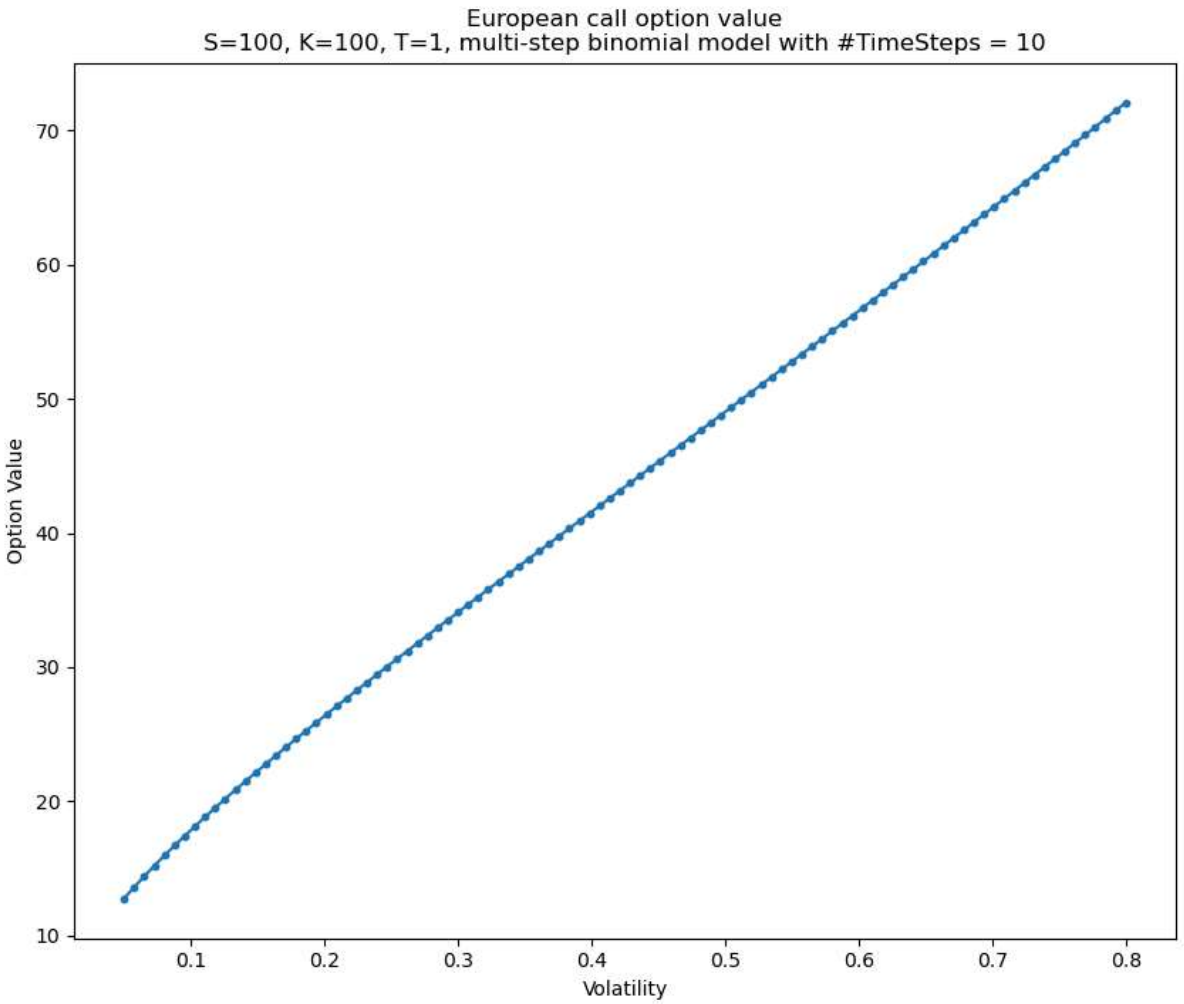
In [15]:
```
# From Python Labs with Kannan, slightly adapted
# Create a user defined function
def binomial_option(spot: float, strike: float, rate: float, sigma: float, time: fl
                    output: int = 0) -> np.ndarray:
    """
```

```
    binomial_option(spot, strike, rate, sigma, time, steps, output=0)
    Function for building binomial option tree for european call option payoff
    Params
    ------
    spot int or float - spot price
    strike int or float - strike price
    rate float - interest rate
    sigma float - volatility
    time int or float - expiration time
    steps int - number of trees
    output int - [0: price, 1: payoff, 2: option value, 3: option delta]
    Returns
    -------
    out: ndarray
    An array object of price, payoff, option value and delta specified by the outpu
    """

    # params
    dt = time / steps
    u = np.exp(sigma * np.sqrt(dt))
    v = np.exp(-sigma * np.sqrt(dt))
    p = 0.5 + rate * np.sqrt(dt) / (2 * sigma)
    df = np.exp(-r * dt)

    # initialize arrays

    px = np.zeros((steps + 1, steps + 1))  # creates quadratic matrix of dimension
    cp = np.zeros((steps + 1, steps + 1))  # for call payoff
    V = np.zeros((steps + 1, steps + 1))  # for option price
    d = np.zeros((steps + 1, steps + 1))  # for delta
    # binomial loop
    # forward loop
    for j in range(steps + 1):
        for i in range(j + 1):
            px[i, j] = spot * np.power(v, i) * np.power(u, j - i)  # create asset p
            cp[i, j] = np.maximum(px[i, j] - strike, 0)
    # reverse loop
    for j in range(steps + 1, 0, -1):
        for i in range(j):
            if j == steps + 1:  # the end of the tree (i.e. if reversed loop, techr
                V[i, j - 1] = cp[i, j - 1]
                d[i, j - 1] = 0  # delta
            else:
                V[i, j - 1] = df * (p * V[i, j] + (1 - p) * V[i + 1, j])
                d[i, j - 1] = (V[i, j] - V[i + 1, j]) / (px[i, j] - px[i + 1, j])

    results = np.around(px, 2), np.around(cp, 2), np.around(V, 2), np.around(d, 4)
    return results[output]
```

In [16]:

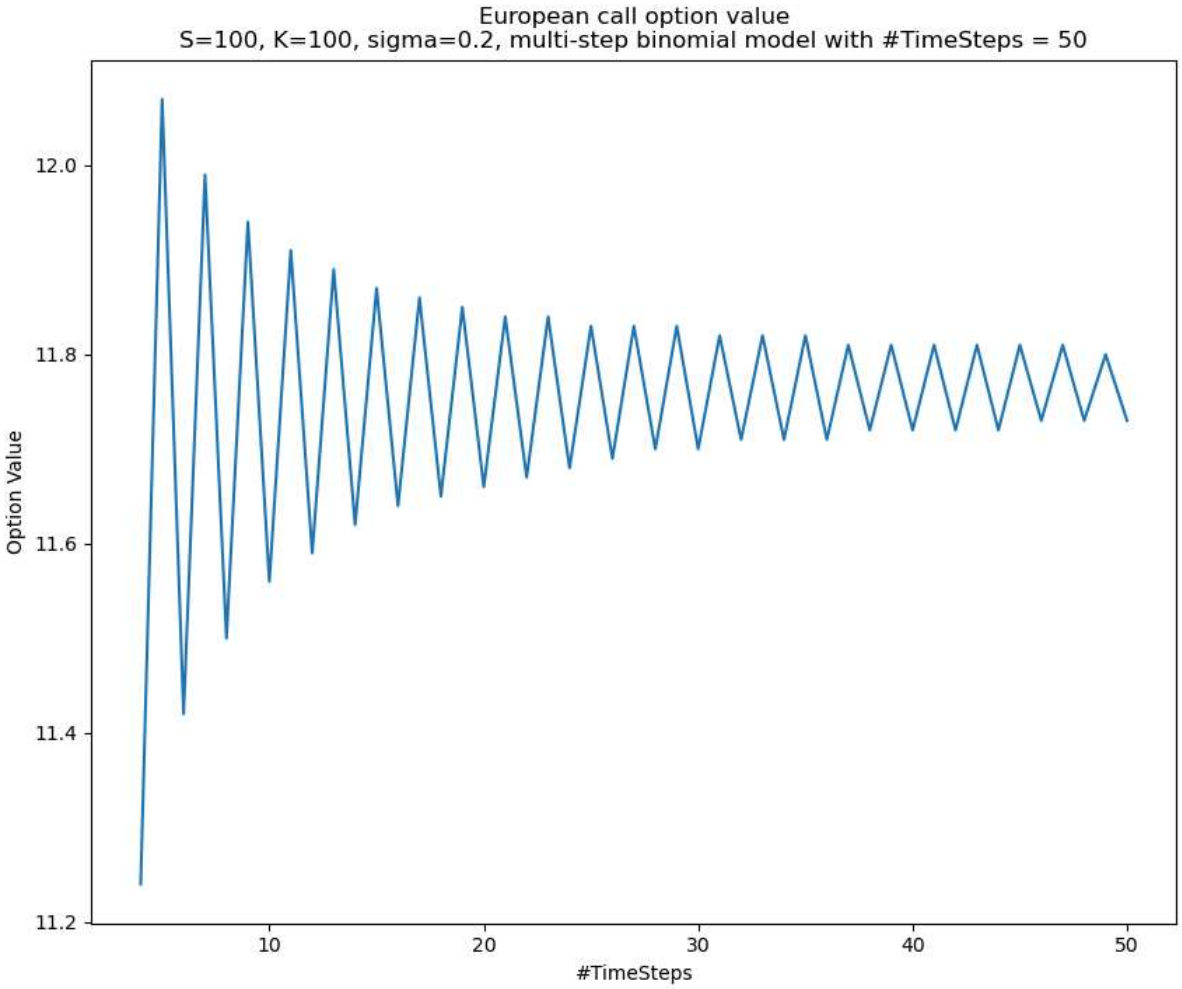
```
# Option value is given by argument 2 as described in docstring
vols = np.linspace(0.05, 0.80, 100)
opt_values = []
time_steps = 10
# Calculate vector containing all option values for a given standard deviation
for vol in vols:
    sigma = np.sqrt(vol)
    opt_values.append(binomial_option(S, E, r, sigma, T, time_steps, 2)[0, 0])

# plot
plt.figure(figsize=(10, 8))
plt.scatter(x=vols, y=opt_values, marker="o", s=10)
plt.plot(vols, opt_values)  # line to connect the scatter plot
plt.xlabel("Volatility")
plt.ylabel("Option Value")
```

```
plt.title(f"European call option value\nS={S}, K={E}, T={T}, multi-step binomial mo
plt.show()
```



3.3: As the time steps increase up to 50 time steps, we observe convergence of the option value.

In [17]:
```
sigma = 0.2
opt_values = []

for time_steps in range(4, 50+1):
    opt_values.append(binomial_option(S, E, r, sigma, T, time_steps, 2)[0, 0])

# plot
plt.figure(figsize=(10, 8))
plt.plot(range(4, 50+1), opt_values)
plt.xlabel("#TimeSteps")
plt.ylabel("Option Value")
plt.title(f"European call option value\nS={S}, K={E}, sigma={sigma}, multi-step bir
plt.show()
```



Question 4.1 - VaR + ES:

Derive a formula for $\mathrm{ES}_c(X)$ for $X \sim \mathcal{N}(\mu, \sigma^2)$.

$$
\begin{aligned}
\mathrm{ES}_c(X) &= \mathbb{E}\left[X | X \leq \mathrm{VaR}_c(X)\right] \\
&= \frac{1}{1-c}\int_c^1 \mathrm{VaR}_u(X)du \\
&= \frac{1}{1-c}\int_c^1 \mu + \sigma \cdot \underbrace{\Phi^{-1}(1-u)}_{=-\Phi(u)}du \\
&= \frac{1}{1-c}\left(\mu\cdot(1-c) - \sigma\int_c^1 \Phi^{-1}(u)du\right) \\
&= \mu - \frac{\sigma}{1-c}\int_c^1 \Phi^{-1}(u)du \\
&\overset{*}{=} \mu - \frac{\sigma}{1-c}\int_{\Phi^{-1}(c)}^{\Phi^{-1}(1)} \Phi^{-1}(\Phi(z))\phi(z)dz \\
&= \mu - \frac{\sigma}{1-c}\int_{\Phi^{-1}(c)}^{\infty} z\cdot\phi(z)dz \\
&= \mu - \frac{\sigma}{1-c}\int_{\Phi^{-1}(c)}^{\infty} z\cdot\exp\left(\frac{-z^2}{2}\right)dz \\
&= \mu - \frac{\sigma}{1-c}\left[-\exp\left(\frac{-z^2}{2}\right)\right]_{\Phi^{-1}(c)}^{\infty} \\
&= \mu - \frac{\sigma}{1-c}\left(-0 + \underbrace{\exp\left(\frac{-\Phi^{-1}(c)^2}{2}\right)}_{=\phi(\Phi^{-1}(c))}\right) = \mu - \frac{\sigma}{1-c}\phi\left(\Phi^{-1}(c)\right)
\end{aligned}
$$

where $*$ used the transformation $u = \Phi(z), du = \phi(z)dz$

Note that the last term can also be written as

$$\mu - \frac{\sigma}{1-c}\phi\left(\Phi^{-1}(c)\right) = \mu + \frac{\sigma}{1-c}\phi\left(\Phi^{-1}(1-c)\right),$$

which is closer to what is given in question 4.2 below, but still with a sign flip for the second part of the formula. Depending on conventions whether a loss is defined as negative or positive, the formulas for ES and VaR could differ slightly.

One could refer to a loss as "-1000 USD", but one could also refer to the same loss as "a loss of 1000 USD" with a positive sign. This can sometimes be confusing when reading literature, comparing lecture notes (Module 2, Lecture on VaR + ES, e.g. slide 39), exercise sheets for the same lecture, or exam formula from question 4.2 below.

Question 4.2 - ES

Compute ES for a range of percentiles [99.95; 99.75; 99.5; 99.25; 99; 98.5; 98; 97.5].

In [18]:
```
percentiles = [99.95, 99.75, 99.5, 99.25, 99, 98.5, 98, 97.5]
es = []


def get_es(alpha, mu, sigma):
    # calculates Expected Shortfall for confidence level alpha for a r.v. X~N(mu, s
```

```
    return mu - sigma * norm.pdf(norm.ppf(1 - alpha)) / (1 - alpha)


for percentile in percentiles:
    es.append(get_es(percentile / 100, 0, 1))

df = pd.DataFrame({
    'Percentile': percentiles,
    'Expected Shortfall for X~N(0, 1)': es
})

df
```

Out[18]:

	Percentile	Expected Shortfall for X~N(0, 1)
0	99.95	-3.5544
1	99.75	-3.1044
2	99.50	-2.8919
3	99.25	-2.7612
4	99.00	-2.6652
5	98.50	-2.5247
6	98.00	-2.4209
7	97.50	-2.3378

Question 5 - Backtest VaR

Analyzes 10D-VaR for confidence level $\alpha = 99\%$ S&P 500 index data.

In [19]:
```
sp500_data_set = pd.read_csv("Jan 24 Exam 1 Data.csv", sep=",", parse_dates=['Date'
sp500_data_set
```

Out[19]:

	SP500
Date	
2013-01-22	1492.5601
2013-01-23	1494.8101
2013-01-24	1494.8199
2013-01-25	1502.9600
2013-01-28	1500.1801
...	...
2017-12-29	2673.6101
2018-01-02	2695.8101
2018-01-03	2713.0601
2018-01-04	2723.9900
2018-01-05	2743.1499

1250 rows × 1 columns

In [20]:
```
def extend_data_set_with_return_data(data_set, time_period):
    # data_set: pandas dataframe with dates + prices in column in 0
    # time_period used to calculate VaR (required to calculate return information f

    # Calculate daily returns + log returns
    data_set['1D_Return'] = data_set.iloc[:, 0].pct_change()
    data_set['1D_LogReturn'] = data_set['1D_Return'].apply(lambda x: math.log(x + 1
    # Log returns are summable, i.e. the N-day-log-return is the sum of N consecuti
    data_set['ND_LogReturn'] = data_set['1D_LogReturn'].rolling(window=time_period)
    return data_set


def calculate_var(extended_data_set, confidence_level):
    # extended_data_set: extended pandas dataframe with return information
    # confidence_level: confidence interval, e.g. 0.99

    mu = extended_data_set['ND_LogReturn'].mean()
    sigma = extended_data_set['ND_LogReturn'].std()
    return norm.ppf(1 - confidence_level, mu, sigma)  # scipy.stats way to do mu -


time_period = 10
confidence_level = 0.99

sp500_df = pd.read_csv("Jan 24 Exam 1 Data.csv", sep=",", parse_dates=['Date'], inc
sp500_df = extend_data_set_with_return_data(sp500_df, time_period)

# Calculating 10-D-VaR
var_99_10d = calculate_var(sp500_df, confidence_level)
print(f"The {time_period}D-VaR at confidence level {confidence_level} for the giver
```

```
The 10D-VaR at confidence level 0.99 for the given dataset is -0.04481
```

5a: Count and percentage of VaR breaches:

In [21]:
```
var_breaches = sp500_df['ND_LogReturn'] < var_99_10d
print(f"Number of VaR breaches: {var_breaches.sum()}")
percentage = var_breaches.sum() / sp500_df['ND_LogReturn'].count()
# do not divide by var_breaches.count() as this has 10 elements more than one coula
print(f"Percentage of VaR breaches: {round(percentage * 100, 3)}%")
```

```
Number of VaR breaches: 26
Percentage of VaR breaches: 2.097%
```

5b: Count and percentage of consecutive breaches:

In [22]:
```
# add auxiliary column to dataframe to indexify the var breaches
sp500_df['Counter'] = range(sp500_df['SP500'].count())
# create new dataframe that contains only the var breaches now
sp500_var_breaches_df = sp500_df[var_breaches]
sp500_var_breaches_df
```

Out[22]:

Date	SP500	1D_Return	1D_LogReturn	ND_LogReturn	Counter
2014-02-03	1741.8900	-0.0228	-0.0231	-0.0541	260
2014-02-04	1755.2000	0.0076	0.0076	-0.0492	261
2014-02-05	1751.6400	-0.0020	-0.0020	-0.0519	262
2014-10-13	1874.7400	-0.0165	-0.0166	-0.0535	435
2014-10-14	1877.7000	0.0016	0.0016	-0.0491	436
2014-12-16	1972.7400	-0.0085	-0.0085	-0.0465	480
2015-08-21	1970.8900	-0.0319	-0.0324	-0.0527	651
2015-08-24	1893.2100	-0.0394	-0.0402	-0.1057	652
2015-08-25	1867.6100	-0.0135	-0.0136	-0.1097	653
2015-08-26	1940.5100	0.0390	0.0383	-0.0723	654
2015-08-27	1987.6600	0.0243	0.0240	-0.0470	655
2015-08-28	1988.8700	0.0006	0.0006	-0.0503	656
2015-08-31	1972.1801	-0.0084	-0.0084	-0.0640	657
2015-09-01	1913.8500	-0.0296	-0.0300	-0.0914	658
2015-09-02	1948.8600	0.0183	0.0181	-0.0649	659
2015-09-29	1884.0900	0.0012	0.0012	-0.0487	677
2016-01-07	1943.0900	-0.0237	-0.0240	-0.0482	746
2016-01-08	1922.0300	-0.0108	-0.0109	-0.0714	747
2016-01-11	1923.6700	0.0009	0.0009	-0.0690	748
2016-01-12	1938.6801	0.0078	0.0078	-0.0590	749
2016-01-13	1890.2800	-0.0250	-0.0253	-0.0949	750
2016-01-14	1921.8400	0.0167	0.0166	-0.0711	751
2016-01-15	1880.3300	-0.0216	-0.0218	-0.0834	752
2016-01-19	1881.3300	0.0005	0.0005	-0.0675	753
2016-01-20	1859.3300	-0.0117	-0.0118	-0.0813	754
2016-01-21	1868.9900	0.0052	0.0052	-0.0629	755

In [23]:

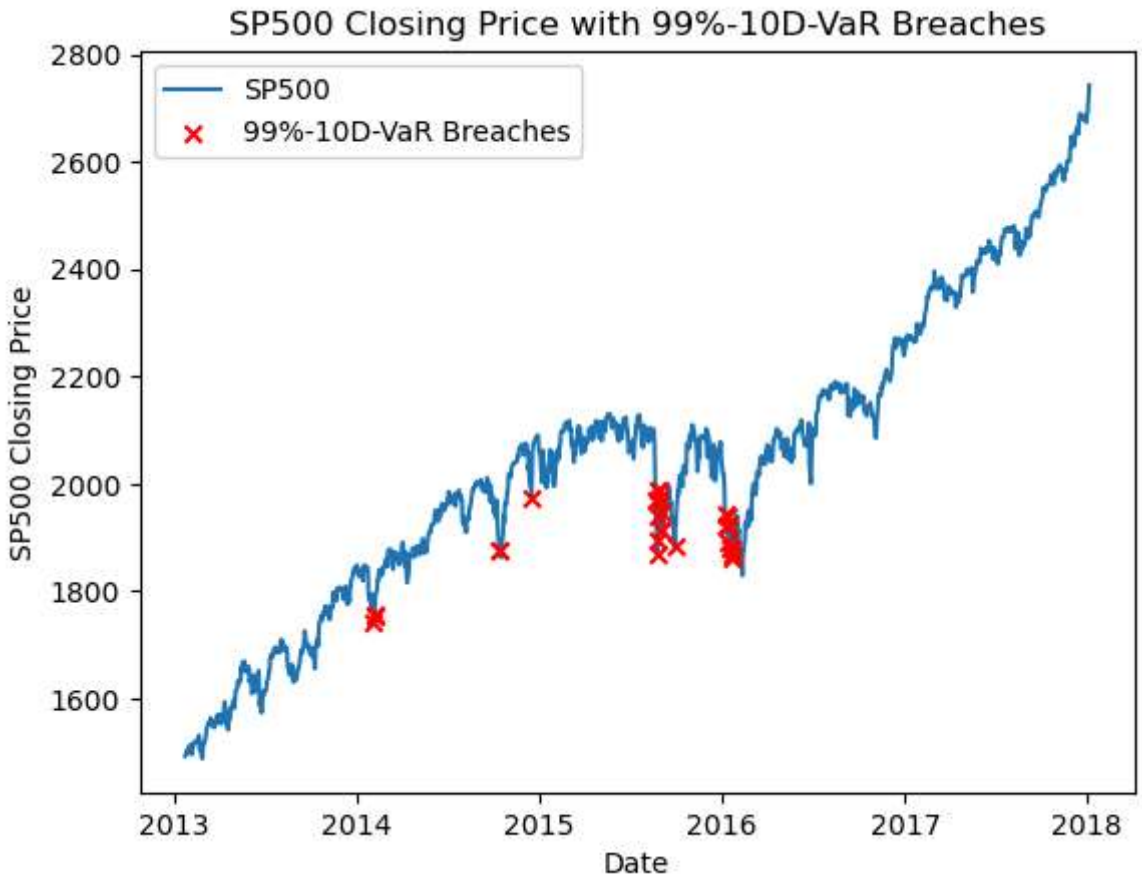
```
def count_consecutive(llist):
    count = 0
    for i in range(len(llist) - 1):
        if llist[i] + 1 == llist[i + 1]:
            count += 1
    return count


number_of_consecutive_var_breaches = count_consecutive(sp500_var_breaches_df['Count
percentage = number_of_consecutive_var_breaches / sp500_var_breaches_df['Counter'].

print(f"Number of consecutive VaR breaches: {number_of_consecutive_var_breaches}")
print(f"Percentage of VaR breaches, that are also consecutive VaR breaches: {round(
```

```
Number of consecutive VaR breaches: 20
Percentage of VaR breaches, that are also consecutive VaR breaches: 76.9231%
```

It turns out that VaR breaches often occur consecutively - 3/4 of all VaR breaches in this sample were actually consecutive breaches - which can be interpreted as a consequence of volatility clustering of asset prices.

5c: Plot data with the breaches visually identified:

In [24]:
```
fig, ax = plt.subplots()

# Plot the SP500 prices
ax.plot(sp500_df.index, sp500_df['SP500'], label='SP500', zorder=5)

# Highlight VaR breaches
ax.scatter(sp500_df.index[var_breaches], sp500_df['SP500'][var_breaches == True],
           color='red', marker='x', label='99%-10D-VaR Breaches', zorder=10)

ax.set_xlabel('Date')
ax.set_ylabel('SP500 Closing Price')
ax.set_title('SP500 Closing Price with 99%-10D-VaR Breaches')
ax.legend()
plt.show()
```



Rolling standard deviation σ from 21 daily returns:

In [25]:
```
sp500_df['21D_Sigma'] = sp500_df['1D_Return'].rolling(window=21).std()
sp500_df['21D_Mean'] = sp500_df['1D_Return'].rolling(window=21).mean()
sp500_df
```

Out[25]:

Date	SP500	1D_Return	1D_LogReturn	ND_LogReturn	Counter	21D_Sigma	21D_Mean
2013-01-22	1492.5601	NaN	NaN	NaN	0	NaN	NaN
2013-01-23	1494.8101	1.5075e-03	1.5063e-03	NaN	1	NaN	NaN
2013-01-24	1494.8199	6.6142e-06	6.6142e-06	NaN	2	NaN	NaN
2013-01-25	1502.9600	5.4455e-03	5.4307e-03	NaN	3	NaN	NaN
2013-01-28	1500.1801	-1.8496e-03	-1.8513e-03	NaN	4	NaN	NaN
...	...	...	...	...	...	...	...
2017-12-29	2673.6101	-5.1832e-03	-5.1966e-03	0.0081	1245	0.0038	0.0009
2018-01-02	2695.8101	8.3034e-03	8.2691e-03	0.0074	1246	0.0039	0.0009
2018-01-03	2713.0601	6.3988e-03	6.3784e-03	0.0085	1247	0.0040	0.0013
2018-01-04	2723.9900	4.0286e-03	4.0205e-03	0.0157	1248	0.0040	0.0015
2018-01-05	2743.1499	7.0338e-03	7.0091e-03	0.0236	1249	0.0040	0.0020

1250 rows × 7 columns

In [26]:
```
var_timescale21 = sp500_df['21D_Mean'] / 21 * time_period - sp500_df['21D_Sigma'] *
var_timescale21
```

Out[26]:
```
Date
2013-01-22       NaN
2013-01-23       NaN
2013-01-24       NaN
2013-01-25       NaN
2013-01-28       NaN
                ...
2017-12-29   -0.0085
2018-01-02   -0.0086
2018-01-03   -0.0086
2018-01-04   -0.0085
2018-01-05   -0.0083
Length: 1250, dtype: float64
```

In [27]:
```
var_breaches_timescale21 = sp500_df['1D_Return'] < var_timescale21
print(f"Number of VaR breaches using time-scaling + 21 daily returns: {var_breaches
```

```
Number of VaR breaches using time-scaling + 21 daily returns: 23
```