LECTURE 7 - MAY 15, 2024

VaR OF A PORTFOLIO: VARIANCE-COVARIANCE METHOD

WE CONTINUE WITH THE COMPUTATION OF THE VaR OF THE PORTFOLIO INTRODUCED IN THE PREVIOUS LECTURE. COPY AND PASTE THE PYTHON CODE YOU FIND ON UniStudium.

ASSUMPTION: THE DAILY RETURN RATES ARE JOINTLY NORMAL DISTRIBUTED WITH PARAMETERS μ_i, σ_i, AND COVARIANCES $Cov_{ij} = \rho_{ij}\sigma_i\sigma_j$ $(i \neq j)$. THIS MEANS THAT THE RANDOM VECTOR

$$\left(\frac{\Delta S^1}{S_0^1}, \dots, \frac{\Delta S^m}{S_0^m}\right) \sim N(\underline{\mu}, \Sigma)$$

THIS IMPLIES THAT $\Delta S^i / S_0^i \sim N(\mu_i, \sigma_i)$ AND THE RETURN RATE OF THE PORTFOLIO IS

$$\frac{\Delta P}{P_0} = \frac{P_1 - P_0}{P_0} = \frac{P_1}{P_0} - 1 \sim N(\mu_{port}, \sigma^2_{port})$$

WHERE

$$\mu_{port} = \underline{\mu}^T \cdot \underline{w} \quad \text{AND} \quad \sigma_{port} = \sqrt{\underline{w}^T \cdot \Sigma \cdot \underline{w}}$$

```
[]: # Delete rows (if any) with NaN
    hist_data = hist_data.dropna()
```

```
[]: # Compute daily return rates
    return_rates = hist_data.pct_change()
    return_rates = return_rates.dropna()
    return_rates
```

```
[]: # Generate the variance-covariance matrix
   cov_matrix = return_rates.cov()
   cov_matrix
```

```
[]: # Compute the mean of return rates
   mean_return_rates = return_rates.mean()
   mean_return_rates
```

THE MEAN RETURN RATE OF THE PORTFOLIO IS

$$\mu_{port} = \underline{\mu}^T \cdot \underline{w} = \mu_1 w_1 + \mu_2 w_2 + \cdots + \mu_m w_m$$

```
[]: # Compute the mean of portfolio r.r.
   mean_port = mean_return_rates.dot(weights)
```

THE STANDARD DEVIATION OF THE PORTFOLIO RETURN RATE IS

$$\sigma_{port} = \sqrt{\underline{w}^T \cdot \Sigma \cdot \underline{w}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m} w_i w_j Cov_{ij}}$$

Σ → VAR-COV MATRIX

```
[]: # Compute the standard deviation of portfolio r.r.
   std_port = np.sqrt(weights.T.dot(cov_matrix)
                      .dot(weights))
```

IN THE LAST LECTURE WE HAVE SEEN RELATIVE AND ABSOLUTE VaR: MOSTLY, VaR REFERS TO ABSOLUTE VaR.

THE (ABSOLUTE) VaR OF THE PORTFOLIO IS GIVEN BY

$$VaR_\alpha(\Delta P) = P_0 \cdot \left[0 - q_{1-\alpha}\left(\frac{\Delta P}{P_0}\right)\right] = -P_0 \cdot q_{1-\alpha}\left(\frac{\Delta P}{P_0}\right)$$

$$\varepsilon = \frac{(\Delta P / P_0) - \mu_{port}}{\sigma_{port}} \sim N(0,1)$$ ← STANDARD NORMAL

$$\frac{\Delta P}{P_0} = \mu_{port} + \sigma_{port} \cdot \varepsilon$$

THE QUANTILE OF $\frac{\Delta P}{P_0}$ IS DERIVED FROM THE QUANTILE OF THE STANDARD NORMAL

INVERSE OF THE STANDARD NORMAL CDF

$$q_{1-\alpha}\left(\frac{\Delta P}{P_0}\right) = \mu_{port} + \sigma_{port} \cdot N^{-1}(1-\alpha)$$

$N^{-1}(1-\alpha) = z_{1-\alpha}$

[]:
```
# Compute the z-value
alpha = 0.99
z_value = norm.ppf(1 - alpha)
z_value
```

[]:
```
# Compute the (1- alpha)th quantile
quantile = mean_port + std_port * z_value
quantile
```

[]:
```
# Aternatively, we can compute the quantile directly
norm.ppf(1- alpha, mean_port, std_port)
```

[]:
```
# Compute the (absolute) VaR of the portfolio
VaR_port = - initial_investment * quantile
print('The VaR of the portfolio is:', VaR_port)
```

ENLARGING THE TIME HORIZON

WE HAVE COMPUTED THE VaR ON A TIME HORIZON OF 1 DAY. IF WE CONSIDER A TIME HORIZON OF K DAYS THEN

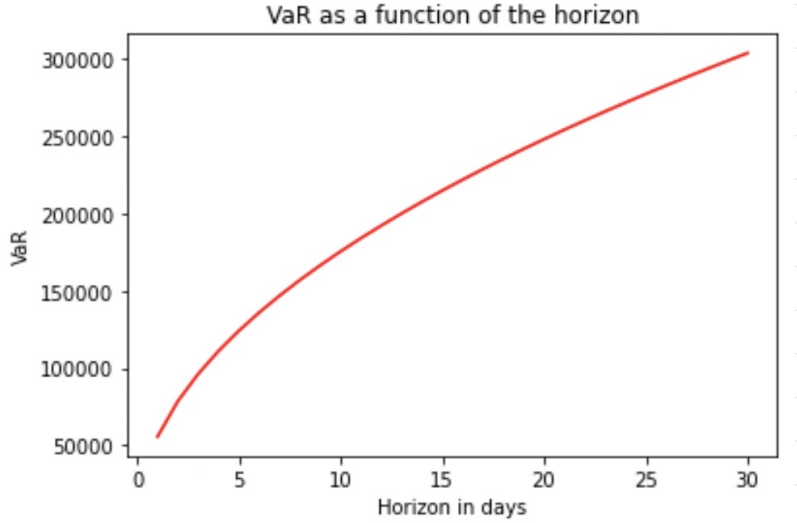
$$VaR_\alpha^{(K\ \text{DAYS})}(\Delta P) = \sqrt{K} \cdot VaR_\alpha^{(1\ \text{DAY})}(\Delta P)$$

[]:
```
# Compute the VaR enlarging the time horizon
VaR_days = []
horizons = range(1, 31)        # 1 INCLUDED, 31 EXCLUDED

for k in horizons:             # k = 1, 2, ..., 30
    VaR_k = np.sqrt(k) * VaR_port
    VaR_days.append(VaR_k)
```

[]:
```
# Plot VaR as a function of the horizon
import matplotlib.pyplot as plt

plt.xlabel('Horizon in days')
plt.ylabel('VaR')
plt.title('VaR as a function of the horizon')
plt.plot(horizons, VaR_days, color='red')
```



HOMEWORK

CONSIDER THE TICKERS:

'META' AND 'TSLA'.

1. DOWLOAD A JOINT DATASET OF CLOSING PRICES OF THE TWO TICKERS FOR THE LAST 3 YEARS.
2. TAKE AS REFERENCE THE LAST AVAILABLE DATE AN TAKE THE PRICES FOR THAT DATE S_0^1, S_0^2.
3. CREATE AN ARRAY OF OWNED SHARES ϑ_1 AND ϑ_2:

$$\vartheta_1 = [3000, 2000] = \vartheta_2$$

4. COMPUTE THE PRICE OF THE PORTFOLIO DETERMINED BY THE ABOVE SHARES, THAT IS P_0 (THE INITIAL INVESTMENT).
5. COMPUTE THE ARRAY OF WEIGHTS

$$w_i = \frac{\vartheta_i S_0^i}{P_0}.$$

6. COMPUTE THE RETURN RATES.
7. DETERMINE THE MEAN VECTOR $\underline{\mu}$ AND THE VARIANCE-COVARIANCE MATRIX OF RETURN RATES Σ
8. COMPUTE THE MEAN AND STANDARD DEVIATION OF THE PORTFOLIO.
9. COMPUTE THE (ABSOLUTE) VaR FOR A 1 DAY HORIZON.
10. PLOT THE GRAPH OF VaR FOR A TIME HORIZON RANGING FROM 1 TO 15 DAYS.

DOWNLOAD DATA FROM yahoo! FINANCE IN A SPECIFIED TIME PERIOD

TO LOAD DATA FROM 2024-05-01 (INCLUDED) TO 2024-05-08 (EXCLUDED), AND SO, TO 2024-05-07 (INCLUDED), WE USE yf.download().

```
[]: tickers = ['AMZN', 'TSLA', 'AAPL', 'GOOG']
    hist_data = yf.download(tickers,        # ALSO A SINGLE TICKER
                            start = '2024-05-01',   # INCLUDED
                            end = '2024-05-08')['Close']
                                  # EXCLUDED
```

OPTION PRICING IN CONTINUOUS TIME

THE MOST KNOWN MODEL TO DEAL WITH OPTION PRICING IN CONTINUOUS TIME IS THE **BLACK-SCHOLES MODEL** THAT CONSIDERS AN IDEAL MARKET COMPOSED BY TWO SECURITIES:

1. **A NON-DIVIDEND PAYING STOCK** (RISKY)

   S_0, CURRENT PRICE (KNOWN)

   S_T, FUTURE PRICE (RANDOM VARIABLE)

$$\ln \frac{S_T}{S_0} \sim N\left(\left(\mu - \frac{\sigma^2}{2}\right)T, \sigma^2 T\right)$$

2. **A BOND** (RISK-FREE)

   r, CONTINUOUSLY COMPOUNDED INTEREST RATE

   e^{-rT}, DISCOUNT FACTOR

IMPORTANT: TIME IS MEASURED IN YEARS

WE HAVE SEEN THAT A **EUROPEAN CALL OPTION** ON THE STOCK WITH MATURITY T AND STRIKE PRICE K IS A CONTINGENT CLAIM WITH PAYOFF AT TIME T

IT IS A CONTRACT THAT GIVES TO THE HOLDER THE RIGHT BUT NOT THE OBLIGATION TO BUY THE STOCK FOR PRICE K AT TIME T

PAYOFF AT TIME T →

$$C_T = \max\{S_T - K, 0\}$$

BLACK-SCHOLES FORMULA FOR THE CALL (DERIVED IN THE ASSET PRICING COURSE)

PRICE AT TIME 0 →

$$C_0 = S_0 \cdot N(d_1) - K \cdot e^{-rT} \cdot N(d_2)$$

$$d_1 = \frac{\ln[S_0/K] + (r + \sigma^2/2)\cdot T}{\sigma \cdot \sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

[]:
```
# Import the libraries
import numpy as np
import pandas as pd
from scipy.stats import norm
```

[]:
```
# Define a function to compute the BS formula
def call_BS(S0, K, T, r, sigma):
    d1 = (np.log(S0/K) + (r + sigma**2 /2)*T)
                        / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S0 * norm.cdf(d1) - K*np.exp(-r*T)
                        * norm.cdf(d2)
```

[]:
```
# Fix the pricing parameters
S0 = 100
T = 1
K = 101
r = 0.05
sigma = 0.3
```

[]:
```
# Compute the BS price of the call
C0_BS = call_BS(S0, T, K, r, sigma)
C0_BS
```

IT IS ≈ 13.76.

MONTE CARLO PRICING

BY THE BLACK-SCHOLES MODEL

$$\ln \frac{S_T}{S_0} \sim \mathcal{N}\left(\left(\mu - \frac{\sigma^2}{2}\right)T,\ \sigma^2 T\right) \quad (\equiv \mathbb{P})$$

WHERE THE DISTRIBUTION IS ACCORDING TO A "REAL-WORLD" PROBABILITY MEASURE $\mathbb{P}$ ENCODING MARKET AGENTS BELIEFS.

IT IS POSSIBLE TO SHOW THAT THE PRICE OF THE CALL C_0 CAN BE EQUIVALENTLY EXPRESSED AS A DISCOUNTED EXPECTATION COMPUTED WITH RESPECT TO AN "ARTIFICIAL" PROBABILITY MEASURE $\mathbb{Q}$ CALLED "RISK-NEUTRAL" PROBABILITY MEASURE:

RISK-FREE C.C. INT. RATE REPLACES μ

$$\ln \frac{S_T}{S_0} \sim \mathcal{N}\left(\left(r - \frac{\sigma^2}{2}\right)T,\ \sigma^2 T\right) \quad (\equiv \mathbb{Q})$$

THERFORE WE GET THAT

$$S_T = S_0 \cdot e^{\left(r - \frac{\sigma^2}{2}\right)T + \sigma\sqrt{T}\varepsilon} \qquad \text{WITH} \quad \varepsilon \sim \mathcal{N}(0,1)$$

SO S_T HAS LOGNORMAL DISTRIBUTION WITH PARAMETERS

$$\tilde{\mu} = \ln S_0 + \left(r - \frac{\sigma^2}{2}\right)T \quad \text{AND} \quad \tilde{\sigma} = \sigma \cdot \sqrt{T}$$

DISCOUNTED EXPECTATION REPRESENTATION

PRICE OF THE CALL

$$C_0 = e^{-rT} \cdot \mathbb{E}^{\mathbb{Q}}[C_T]$$

EXPECTATION OF THE PAYOFF WITH RESPECT TO $\mathbb{Q}$

DISCOUNT FACTOR

$$= e^{-rT} \cdot \mathbb{E}^{\mathbb{Q}}[\max\{S_T - K, 0\}]$$

INTEGRATION VARIABLE

$$= e^{-rT} \cdot \int_{-\infty}^{+\infty} \max\{x - K, 0\} \cdot \ell(x)\, dx$$

$\ell(x)$: LOGNORMAL DENSITY WITH PARAMETERS $\tilde{\mu}$ AND $\tilde{\sigma}$

THE MONTECARLO SIMULATION CONSISTS OF:

1. GENERATE M STANDARD NORMAL PSEUDO-RANDOM NUMBERS

$$\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_M$$

2. COMPUTE THE CORRESPONDING VALUES OF S_T

$$S_T(\varepsilon_1), S_T(\varepsilon_2), \ldots, S_T(\varepsilon_M)$$

3. COMPUTE THE CORRESPONDING VALUES OF THE PAYOFF

$$C_T(\varepsilon_i) = \max\{S_T(\varepsilon_i) - K, 0\} \quad i = 1, \ldots, M$$

4. COMPUTE THE DISCOUNTED AVERAGE

$$C_0 = e^{-rT} \cdot \frac{1}{M} \cdot \sum_{i=1}^{M} \max\{S_T(\varepsilon_i) - K, 0\}$$

[]:
```
# Import the NumPy's pseudo-random generator
from numpy.random import default_rng
```

[]:
```
# Create the generator
rng = default_rng(100)
```
SEED FOR REPRODUCIBILITY

[]:
```
# Generate the standard normal numbers
M = 1000000
epsilons = rng.standard_normal(M)
```

[]:
```
# Generate ST
ST = S0 * np.exp((r - sigma ** 2 / 2) * T +
                 sigma * np.sqrt(T) * epsilons)
```

[]:
```
# Generate the payoff of the call
CT = np.maximum(ST - K, 0)
```

[]:
```
# Compute the Monte Carlo price
CO_MC = np.exp(-r * T) * CT.mean()
CO_MC
```

WE CAN COMPUTE THE PRICE ALSO COMPUTING THE INTEGRAL:

$\varphi(x)$ ← PAYOFF FUNCTION

$$I = \int_{-\infty}^{+\infty} \underbrace{\max\{x - K, 0\}}_{=\varphi(x)} \cdot l(x)\, dx$$

$f_{int}(x)$ ← INTEGRAND

[]:
```
# Define the payoff function
def phi(x):
    return np.maximum(x - K, 0)
```

$\tilde{\mu}$ AND $\tilde{\sigma}$ PARAMETERS

[]:
```
# Define the lognormal density
mu_t = np.log(S0) + (r - sigma**2 /2) * T
sigma_t = sigma * np.sqrt(T)

from scipy.stats import lognorm

def l(x):
    return lognorm.pdf(x, s=sigma_t, scale=np.exp(mu_t))
```

[]:
```
# Define the integrand function
def f_int(x):
    return phi(x) * l(x)

# Compute the integral
from scipy.integrate import quad

I = quad(f_int, -np.infty, np.infty)[0]
```

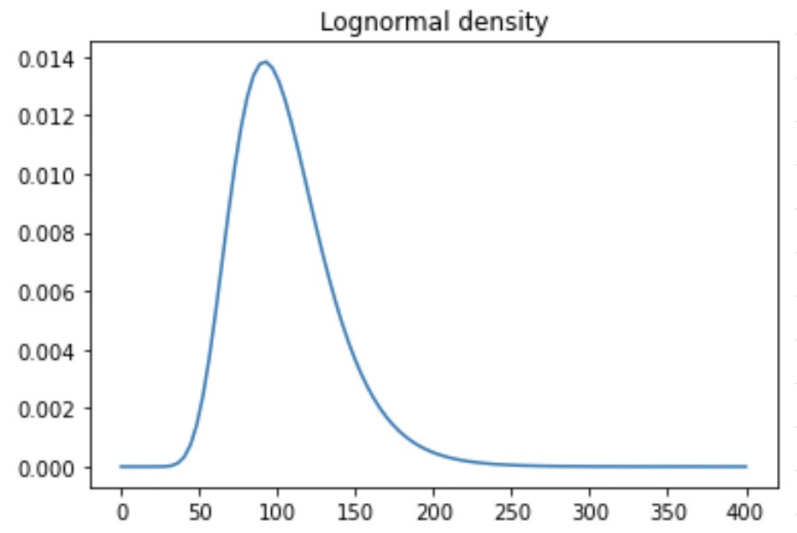
TAKES ONLY THE RESULT OF THE INTEGRAL

[]:
```
# Compute the integral price
C0_int = np.exp(-r*T) * I
C0_int
```

[]:
```
# Plot the lognormal density
import matplotlib.pyplot plt

x = np.linspace(0.01, 400, 100)
y = l(x)

plt.plot(x, y)
plt.title('Lognormal density')
```



REALISM OF THE BLACK-SCHOLES MODEL

NOW THAT WE KNOW HOW TO COMPUTE OPTION PRICES IN THE BLACK-SCHOLES MODEL WE WANT TO VERIFY IF SUCH PRICES AGREE WITH MARKET PRICES. DOWNLOAD THE FILE Calls.csv FROM UniStudium AND SAVE IT IN THE SAME DIRECTORY OF YOUR JupyterLab NOTEBOOK.
IT CONTAINS 100000 (REAL) MARKET OPTION PRICES ON DATE 2019-08-15.

[]:
```
# Load call prices from csv
calls = pd.read_csv('Calls.csv')
```

[]:
```
# Fix the risk-free rate on the market
r = 0.0195
```
← ESTIMATED THROUGH A ZERO COUPON BOND ON THE MARKET

WE CAN COMPUTE THE BLACK-SCHOLES FORMULA ON EVERY ROW USING THE METHOD apply:

```
.apply(lamba row: <function using row>, axis = 1)
```

IT ADDS A NEW COLUMN 'CO_BS' TO THE DATAFRAME

[]:
```
# Compute the BS price on every row
calls['CO_BS'] = calls.apply(lambda row:
                             call_BS(row['S0'], row['K'],
                                     row['T'], r, row['sigma']),
                             axis = 1)
```

EVALUATES THE FUNCTION call_BS ON EVERY ROW OF THE DataFrame

WE EVALUATE THE ERROR BY COMPUTING:

- MEAN ABSOLUTE ERROR

$$MAE = \frac{1}{N} \cdot \sum_{i=1}^{N} |C_i - C_i^{BS}|$$

(C_i: MARKET PRICE, C_i^{BS}: BS PRICE)

- ROOT MEAN SQUARED ERROR

$$RMSE = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (C_i - C_i^{BS})^2}$$

[]:
```
# Compute the MAE
MAE = (np.abs(calls['Opt_Price']-calls['CO_BS'])).mean()
MAE
```

[]:
```
# Compute the RMSE
RMSE = np.sqrt(((calls['Opt_Price']-calls['CO_BS'])**2).mean())
RMSE
```

HOMEWORK

IT IS A CONTRACT THAT GIVES TO THE HOLDER THE RIGHT BUT NOT THE OBLIGATION TO SELL THE STOCK FOR PRICE K AT TIME T

1 CONSIDER A **EUROPEAN PUT OPTION** ON THE STOCK IN THE BLACK-SCHOLES MODEL WITH STRIKE PRICE K, MATURITY T AND PAYOFF AT TIME T

$$P_T = \max\{K - S_T, 0\}.$$

TAKE $S_0 = 100$, $T = 1$, $K = 101$, $r = 0.05$, $\sigma = 0.3$.

(a) COMPUTE THE PRICE P_0 OF THE PUT WITH THE

BLACK-SCHOLES FORMULA FOR THE PUT

PRICE AT TIME 0 →

$$P_0 = K \cdot e^{-rT} \cdot N(-d_2) - S_0 \cdot N(-d_1)$$

SAME OF THE CALL

$$d_1 = \frac{\ln[S_0/K] + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

(b) COMPUTE THE PRICE P_0 OF THE PUT WITH THE MONTE CARLO SIMULATION TAKING $M = 1000000$.

(c) COMPUTE THE PRICE P_0 USING THE INTEGRAL EXPRESSION OF THE PRICE

$$P_0 = e^{-rT} \cdot \mathbb{E}^{\mathbb{Q}}[P_T]$$

$$= e^{-rT} \cdot \int_{-\infty}^{+\infty} \varphi(x) \cdot \underbrace{\ell(x)}_{\text{SAME LOGNORMAL DENSITY OF THE CALL}} dx$$

WITH $\varphi(x) = \max\{K - x, 0\}$.

2 TAKE $S_0 = 100$, $T = 1$, $K = 101$, $r = 0.05$, $\sigma = 0.3$.

CONSIDER A CONTINGENT CLAIM WITH PAYOFF AT TIME T

$$Y_T = \min\left\{S_T, \frac{K}{2}\right\}$$

USING THE BLACK-SCHOLES MODEL:

(a) COMPUTE THE PRICE Y_0 OF THE PUT WITH THE MONTE CARLO SIMULATION TAKING $M = 1000000$.

(b) COMPUTE THE PRICE Y_0 USING THE INTEGRAL EXPRESSION OF THE PRICE

$$Y_0 = e^{-rT} \cdot \mathbb{E}^{\mathbb{Q}}[Y_T]$$

$$= e^{-rT} \cdot \int_{-\infty}^{+\infty} \varphi(x) \cdot \underbrace{\ell(x)}_{\substack{\text{SAME LOGNORMAL} \\ \text{DENSITY OF THE CALL}}} dx$$

WITH $\varphi(x) = \min\left\{x, \frac{K}{2}\right\}$.

(c) VERIFY THAT YOU GET THE SAME RESULT IF IN THE INTEGRAL YOU TAKE 0 IN PLACE OF $-\infty$, THAT IS

$$Y_0 = e^{-rT} \cdot \int_{0}^{+\infty} \varphi(x) \cdot \underbrace{\ell(x)}_{\substack{\text{SAME LOGNORMAL} \\ \text{DENSITY OF THE CALL}}} dx$$

(THIS IS BECAUSE $\ell(x) = 0$ FOR $x \leq 0$).

MULTIVARIATE NORMAL DISTRIBUTION

A RANDOM VECTOR $\underline{X} = (X_1, \dots, X_m)$ HAS MULTIVARIATE NORMAL DISTRIBUTION WITH DENSITY IF WE HAVE:

- $\underline{\mu} = [\mu_1, \dots, \mu_m]^T \in \mathbb{R}^m$, VECTOR OF MEANS
- $\Sigma = [Cov_{i,j}] \in \mathbb{R}^{m \times m}$, VARIANCE - COVARIANCE MATRIX WHICH IS SYMMETRIC, POSITIVE SEMI-DEFINITE (NON-NEGATIVE EIGENVALUES) AND INVERTIBLE.

WE WRITE $\underline{X} \sim \mathcal{N}(\underline{\mu}, \Sigma)$ AND ITS DENSITY IS

$\forall\, \underline{x} = [x_1, \dots, x_m]^T \in \mathbb{R}^m$

$$f(\underline{x}) = \frac{1}{\sqrt{(2\pi)^m \det \Sigma}} \cdot \exp\left(-\frac{1}{2}(\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu})\right)$$

IN THE BIVARIATE CASE ($m = 2$) WE CAN PLOT THE DENSITY, WHICH IS A SURFACE IN THE 3D SPACE. IN THIS CASE, WE HAVE ($Cov_{i,i} = Var_i$):

$$\underline{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \text{ AND } \Sigma = \begin{bmatrix} Var_1 & Cov_{2,1} \\ Cov_{1,2} & Var_2 \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & \rho_{1,2}\sigma_1\sigma_2 \\ \rho_{1,2}\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

WE WRITE (X, Y) IN PLACE OF (X_1, X_2) TO SIMPLIFY NOTATION.

EXERCISE:

WE WANT TO PLOT THE GRAPH OF $f(\underline{x})$ FOR $m = 2$ AND PARAMETERS

$$\underline{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ AND } \Sigma = \begin{bmatrix} 1 & -0.6 \\ -0.6 & 1 \end{bmatrix}$$

[]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
```

[]:
```
# Bidimensional normal parameters
mu = np.array([0, 0])

Sigma = np.array([[1, -0.6],
                  [-0.6, 1]])
```

THE EIGENVALUES OF Σ ARE THE VALUES λ SATISFYING

$$\det(\Sigma - \lambda I) = 0$$

WE CAN FIND THEM WITH `np.linalg.eig()`.

MOREOVER Σ IS INVERTIBLE IFF $\det(\Sigma) \neq 0$.

TO FIND THE DETERMINANT WE USE `np.linalg.det()`

[]:
```
# Verify that the density exists
# Check eigenvalues
eig_values = np.linalg.eig(Sigma)[0]
eig_values >= 0
```

TAKE EIGENVALUES ONLY

[]:
```
# Check invertibility
np.linalg.det(Sigma) != 0
```

[]:
```
# Create a sample of points
x = np.linspace(-3, 3, 100)
y = np.linspace(-3, 3, 100)
X, Y = np.meshgrid(x, y)
pos = np.empty((X.shape[0], X.shape[1], 2))
pos[:, :, 0] = X
pos[:, :, 1] = Y
```

CREATES A GRID TO COMPUTE THE SURFACE

[]:
```
# Create the bidimensional normal random vector
rv = multivariate_normal(mean = mu, cov = Sigma)
```

[]:
```
# Make the 3D plot
fig = plt.figure(figsize = (12, 8))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, rv.pdf(pos), cmap='viridis')
```