Regression Method

In [1]:
```
# Necessary packages
import pandas as pd
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

# Importing the CSV file
df_reg = pd.read_csv("treasury_data_all.csv")
df_reg.head()
```

Out[1]:

	CUSIP	ISSUE_DATE	MATURITY_DATE	COUPON_RATE	BID_PRICE	ASK_PRICE	TIME_TO_MATURITY
0	912810EB	22/11/1988	15/11/2018	9.000	103.093750	103.218750	0.5
1	912828M6	16/11/2015	15/11/2018	1.250	99.609375	99.648438	0.5
2	912828JR	17/11/2008	15/11/2018	3.750	100.734375	100.757812	0.5
3	912796QJ	17/05/2018	15/11/2018	0.000	99.069000	99.073667	0.5
4	912828R4	16/05/2016	15/05/2019	0.875	98.679688	98.718750	1.0

This is an overview over the different variables which is located in this dataframe

- CUSIP: A unique security-level identifier
- ISSUE_DATE: The date the Treasury bond was issued
- MATURITY_DATEThe date the Treasury bond matures
- COUPON_RATEThe bond’s coupon rate in percent
- BID_PRICEThe price at which dealers are willing to buy
- ASK_PRICEThe price at which dealers are willing to sell
- TIME_TO_MATURITYThe time-to-maturity measured in years

In [2]:
```
df_reg["Mid_Price"] = (df_reg.ASK_PRICE + df_reg.BID_PRICE)/2
df_reg.head()
```

Out[2]:

	CUSIP	ISSUE_DATE	MATURITY_DATE	COUPON_RATE	BID_PRICE	ASK_PRICE	TIME_TO_MATURITY	Mid_Price
0	912810EB	22/11/1988	15/11/2018	9.000	103.093750	103.218750	0.5	103.156250
1	912828M6	16/11/2015	15/11/2018	1.250	99.609375	99.648438	0.5	99.628906
2	912828JR	17/11/2008	15/11/2018	3.750	100.734375	100.757812	0.5	100.746094
3	912796QJ	17/05/2018	15/11/2018	0.000	99.069000	99.073667	0.5	99.071333
4	912828R4	16/05/2016	15/05/2019	0.875	98.679688	98.718750	1.0	98.699219

In [3]:
```
## Creating the Cash Flow Matrix
unique_maturities = sorted(df_reg["TIME_TO_MATURITY"].unique())

# Initialize the cash flow matrix with zeroes
cash_flow_matrix = np.zeros((len(df_reg), len(unique_maturities)))

# Iterative process to create a vector row of each bonds casf flows at different maturities
for i, bond in df_reg.iterrows():
    for j, maturity in enumerate(unique_maturities):
        # Coupon payment for maturities before, and at the bonds maturity
        cash_flow = 100*bond["COUPON_RATE"] / 100 / 2  # Assuming semiannual
        if maturity == bond["TIME_TO_MATURITY"]:
            cash_flow += 100    # Adding principal
        cash_flow_matrix[i, j] = cash_flow

# We convert the cash flow matrix to Dataframe
C = pd.DataFrame(cash_flow_matrix, columns = unique_maturities)
C.head()
```

Out[3]:

	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5	8.0	
0	104.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.
1	100.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.6250	0.
2	101.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.8750	1.
3	100.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.
4	0.4375	100.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.

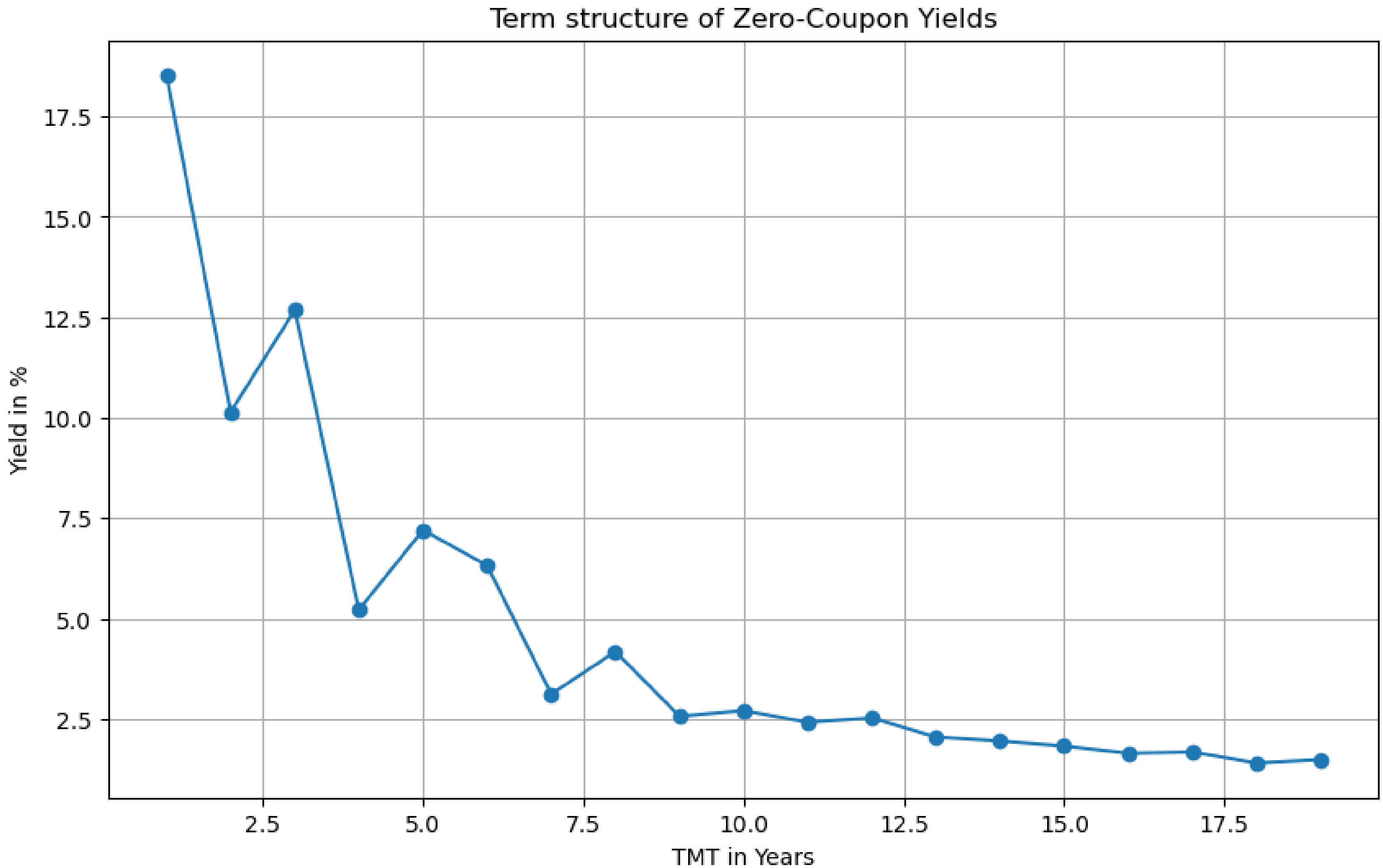
In [4]:
```
## Step 2. Prepare the Price Vector
price_vector = df_reg["Mid_Price"].values
```

In [5]:
```
# Transpose and multiply the cash flow matrix
CT_C = np.dot(C.T, C)

# Invert the resulting matrix
inv_CT_C = inv(CT_C)

# Multiply by tranposed cash flow matrix and price vector
Z_0 = np.dot(np.dot(inv_CT_C, C.T), price_vector)

# Convert the discount factors to DataFrame
df_reg_discount = pd.DataFrame(Z_0)
df_reg_discount["Discount_Factors"] = df_reg_discount[0]
df_reg_discount = df_reg_discount.drop(df_reg_discount.columns[0], axis = 1)
df_reg_discount.head()
```

Out[5]:

	Discount_Factors
0	0.733487
1	0.830802
2	0.816418
3	0.683638
4	0.811555

In [6]:
```
## Task 2 is to calculate the continously compounded zero-coupon bond yields for each maturity

# We will be using the same discount factors as calculated above
df_reg_discount["Continously_Compounded_Yield"] = -np.log(df_reg_discount["Discount_Factors"]) / df_reg_disco

# Convert the yield to percent
df_reg_discount["Continously_Compounded_Yield_in%"] = df_reg_discount.Continously_Compounded_Yield * 100


# Plot the term structure
plt.figure(figsize = (10, 6))
plt.plot(df_reg_discount.index, df_reg_discount["Continously_Compounded_Yield_in%"], marker = "o")
plt.title("Term structure of Zero-Coupon Yields")
plt.xlabel("TMT in Years")
plt.ylabel("Yield in %")
plt.grid(True)
plt.show()
```



Nelson Siegel

In [7]:
```
# Necessary package to optimize
from scipy.optimize import minimize
```

In [8]:
```
# We define the df_NS as same as the df_reg
df_NS = df_reg

# Define the NS model functions
def nelson_siegel_yield(theta_0, theta_1, theta_2, lambda_, T):
    # Calculates the yield for a given maturity T based on NS
    return theta_0 + (theta_1 + theta_2)*(1-np.exp(-T / lambda_)) / (T / lambda_) - theta_2 * np.exp(-T / lam

def discount_factor(theta_0, theta_1, theta_2, lambda_, T):
    # Calculates the discount factor for a given maturity
    r = nelson_siegel_yield(theta_0, theta_1, theta_2, lambda_, T)
    return np.exp(-r * T)
```

In [9]:
```
# Defining the objective function for Optimization
def objective_function(params, df_NS):
    theta_0, theta_1, theta_2, lambda_ = params
    squared_errors = []

    for index, row in df_NS.iterrows():
        T = row["TIME_TO_MATURITY"]
        coupon = row["COUPON_RATE"] / 100
        mid_price = row["Mid_Price"]
        face_value = 100     # We assume that the standard face value is 100

        # Calculate the price of the bond under the NS model
        # We will assume semi-annual coupons
        cash_flows = [coupon * face_value / 2 for _ in range(int(2*T))]
        cash_flows[-1] += face_value     # Add the Face value as the last payment
        model_price = sum(discount_factor(theta_0, theta_1, theta_2, lambda_, t/2) * cf for t, cf in enumerat

        # Calculate the squared error between the model price and the observed mid price
        squared_errors.append((model_price - mid_price) ** 2)

    return np.sum(squared_errors)
```

In [10]:
```
# We will initialize the optimization model by a guess for our thetas, lambda and T
initial_guess = [0.03, 0, 0, 3]

# Start the optimization
result = minimize(objective_function, initial_guess, args = (df_NS, ), method = 'L-BFGS-B')

# Print out the result
print("Optimized Parameters: θ0 = {:.4f}, θ1 = {:.4f}, θ2 = {:.4f}, λ = {:.4f}".format(*result.x))
```

```
Optimized Parameters: θ0 = 0.0300, θ1 = -0.0092, θ2 = 0.0054, λ = 3.0000
```

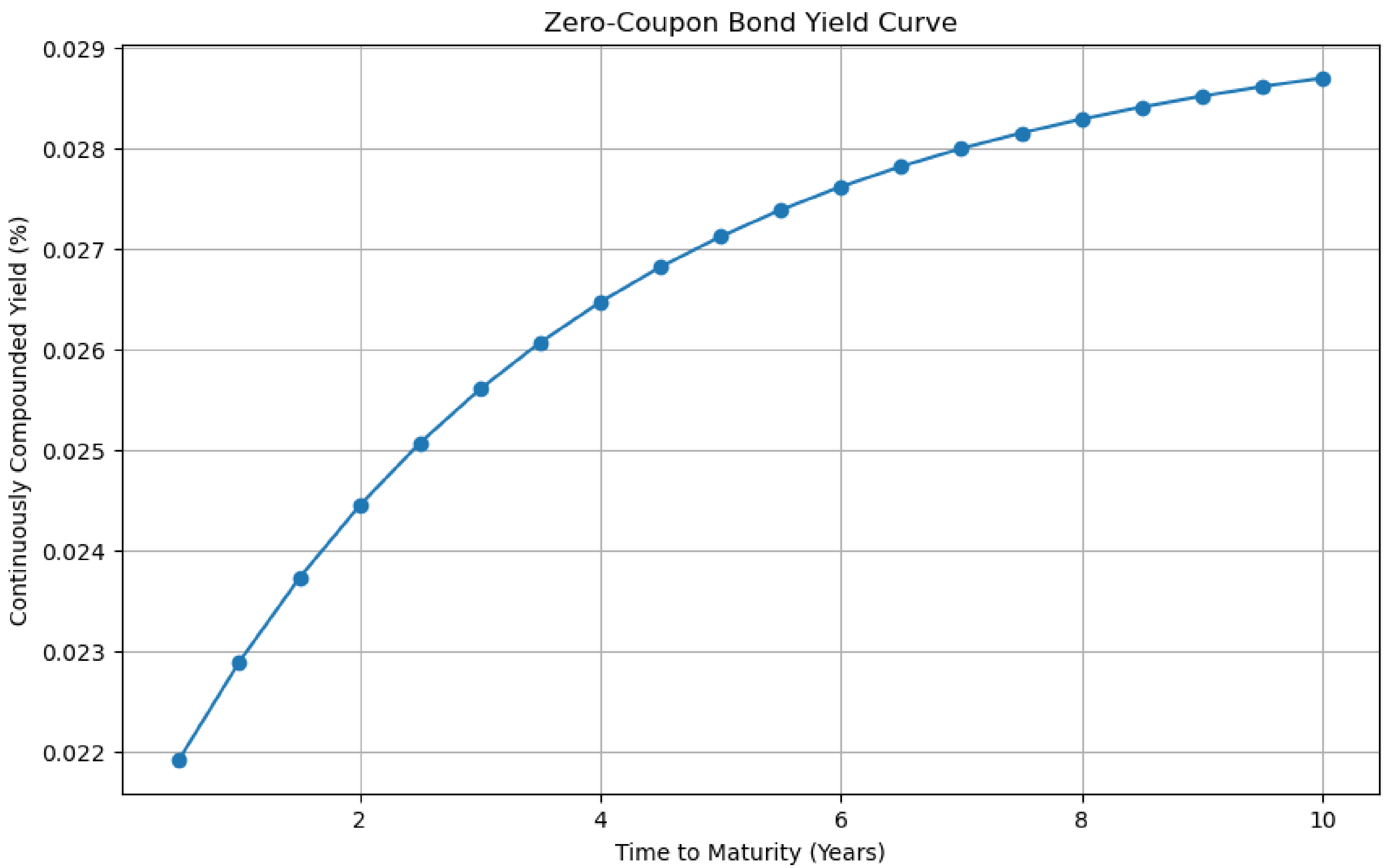
In [11]:
```
### Next task is to se the estimated model parameters to calculate the zero-coupon bond yield for each maturi
## We extract our optimized parameters so it could be later used
optimized_params = result.x
theta_0, theta_1, theta_2, lambda_ = optimized_params
```

In [12]:
```
# Extract the unique maturity values
unique_maturities = df_NS["TIME_TO_MATURITY"].unique()

# We then calculate yeuekds for each maturity
zero_cp_yields = {T: nelson_siegel_yield(theta_0, theta_1, theta_2, lambda_, T) for T in unique_maturities}

# We convert the dictionary to a DF
df_NS_model = pd.DataFrame(list(zero_cp_yields.items()), columns = ["Maturity", "Continously_Compounded_Yield
df_NS_model.head()
```

Out[12]:

	Maturity	Continously_Compounded_Yield_in%
0	0.5	0.021928
1	1.0	0.022898
2	1.5	0.023732
3	2.0	0.024451
4	2.5	0.025071

In [13]:
```
# We can then plot the term structure of interest rates obtained in our calculation

plt.figure(figsize=(10, 6))
plt.plot(df_NS_model['Maturity'], df_NS_model['Continously_Compounded_Yield_in%'], marker='o')
plt.title('Zero-Coupon Bond Yield Curve')
plt.xlabel('Time to Maturity (Years)')
plt.ylabel('Continuously Compounded Yield (%)')
plt.grid(True)
plt.show()
```



Bootstrap Method

In [14]:
```
df_bot = pd.read_csv("treasury_data_bootstrap.csv")
df_bot.head()
```

Out[14]:

	CUSIP	ISSUE_DATE	MATURITY_DATE	COUPON_RATE	BID_PRICE	ASK_PRICE	TIME_TO_MATURITY
0	912810EB	22/11/1988	15/11/2018	9.000	103.093750	103.218750	0.5
1	912828R4	16/05/2016	15/05/2019	0.875	98.679688	98.718750	1.0
2	912828LY	16/11/2009	15/11/2019	3.375	101.382812	101.429688	1.5
3	912828X9	15/05/2017	15/05/2020	1.500	98.218750	98.257812	2.0
4	9128283G	15/11/2017	15/11/2020	1.750	98.195312	98.234375	2.5

In [15]:
```
df_bot["Mid_Price"] = (df_bot.ASK_PRICE + df_bot.BID_PRICE)/2
df_bot.head()
```

Out[15]:

	CUSIP	ISSUE_DATE	MATURITY_DATE	COUPON_RATE	BID_PRICE	ASK_PRICE	TIME_TO_MATURITY	Mid_Price
0	912810EB	22/11/1988	15/11/2018	9.000	103.093750	103.218750	0.5	103.156250
1	912828R4	16/05/2016	15/05/2019	0.875	98.679688	98.718750	1.0	98.699219
2	912828LY	16/11/2009	15/11/2019	3.375	101.382812	101.429688	1.5	101.406250
3	912828X9	15/05/2017	15/05/2020	1.500	98.218750	98.257812	2.0	98.238281
4	9128283G	15/11/2017	15/11/2020	1.750	98.195312	98.234375	2.5	98.214844

In [16]:
```
## Step 1. Creating the Cash Flow Matrix
unique_maturities = sorted(df_bot["TIME_TO_MATURITY"].unique())

# Initialize the cash flow matrix with zeroes
cash_flow_matrix = np.zeros((len(df_bot), len(unique_maturities)))

# Iterative process to create a vector row of each bonds casf flows at different maturities
for i, bond in df_bot.iterrows():
    for j, maturity in enumerate(unique_maturities):
        # Coupon payment for maturities before, and at the bonds maturity
        cash_flow = 100*bond["COUPON_RATE"] / 100 / 2  # Assuming semiannual
        if maturity == bond["TIME_TO_MATURITY"]:
            cash_flow += 100    # Adding principal
        cash_flow_matrix[i, j] = cash_flow

# We convert the cash flow matrix to Dataframe
df_cash_flow_matrix = pd.DataFrame(cash_flow_matrix, columns = unique_maturities)
df_cash_flow_matrix.head()
```

Out[16]:

	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5	
0	104.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5000	4.5(
1	0.4375	100.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4375	0.4:
2	1.6875	1.6875	101.6875	1.6875	1.6875	1.6875	1.6875	1.6875	1.6875	1.6875	1.6875	1.6875	1.6875	1.6875	1.6875	1.6ł
3	0.7500	0.7500	0.7500	100.7500	0.7500	0.7500	0.7500	0.7500	0.7500	0.7500	0.7500	0.7500	0.7500	0.7500	0.7500	0.7!
4	0.8750	0.8750	0.8750	0.8750	100.8750	0.8750	0.8750	0.8750	0.8750	0.8750	0.8750	0.8750	0.8750	0.8750	0.8750	0.8

In [17]:
```
## Step 2. Prepare the Price Vector
price_vector = df_bot["Mid_Price"].values
```

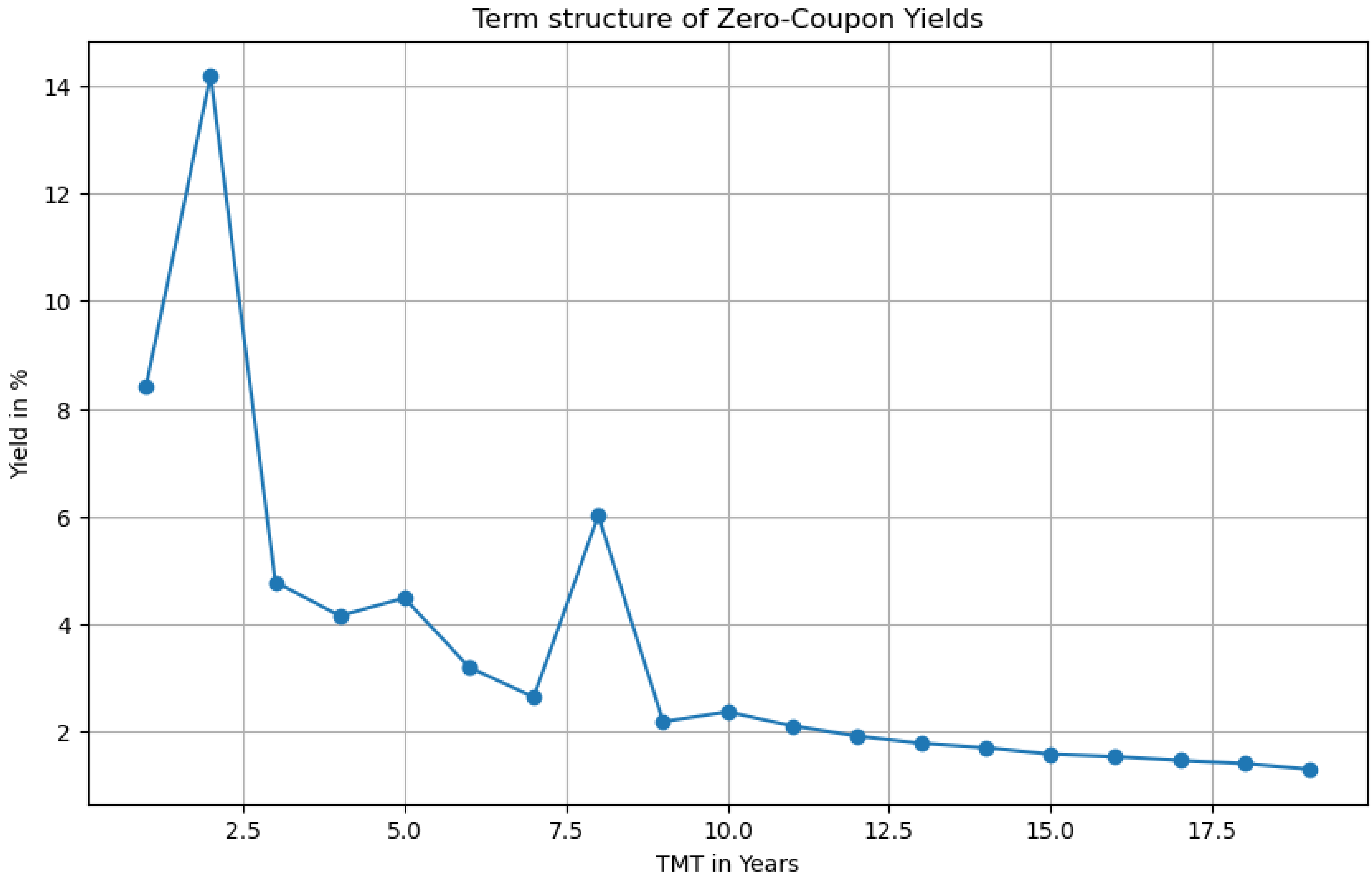
In [18]:

```
## Step 3.Calculate Discount Factors

# Invert the cash flow matrix
cash_flow_matrix_inv = inv(cash_flow_matrix)

# Calculate the discount factors by multiplying the inverted matrix by the price vector
discount_factors = np.dot(cash_flow_matrix_inv, price_vector)

# Convert disocunt factors to Dataframe
df_discount_factors = pd.DataFrame(discount_factors)
df_discount_factors["Discount_Factors"] = df_discount_factors[0]
df_bootstrap = df_discount_factors.drop(df_discount_factors.columns[0], axis = 1)
df_bootstrap.head()
```

Out[18]:

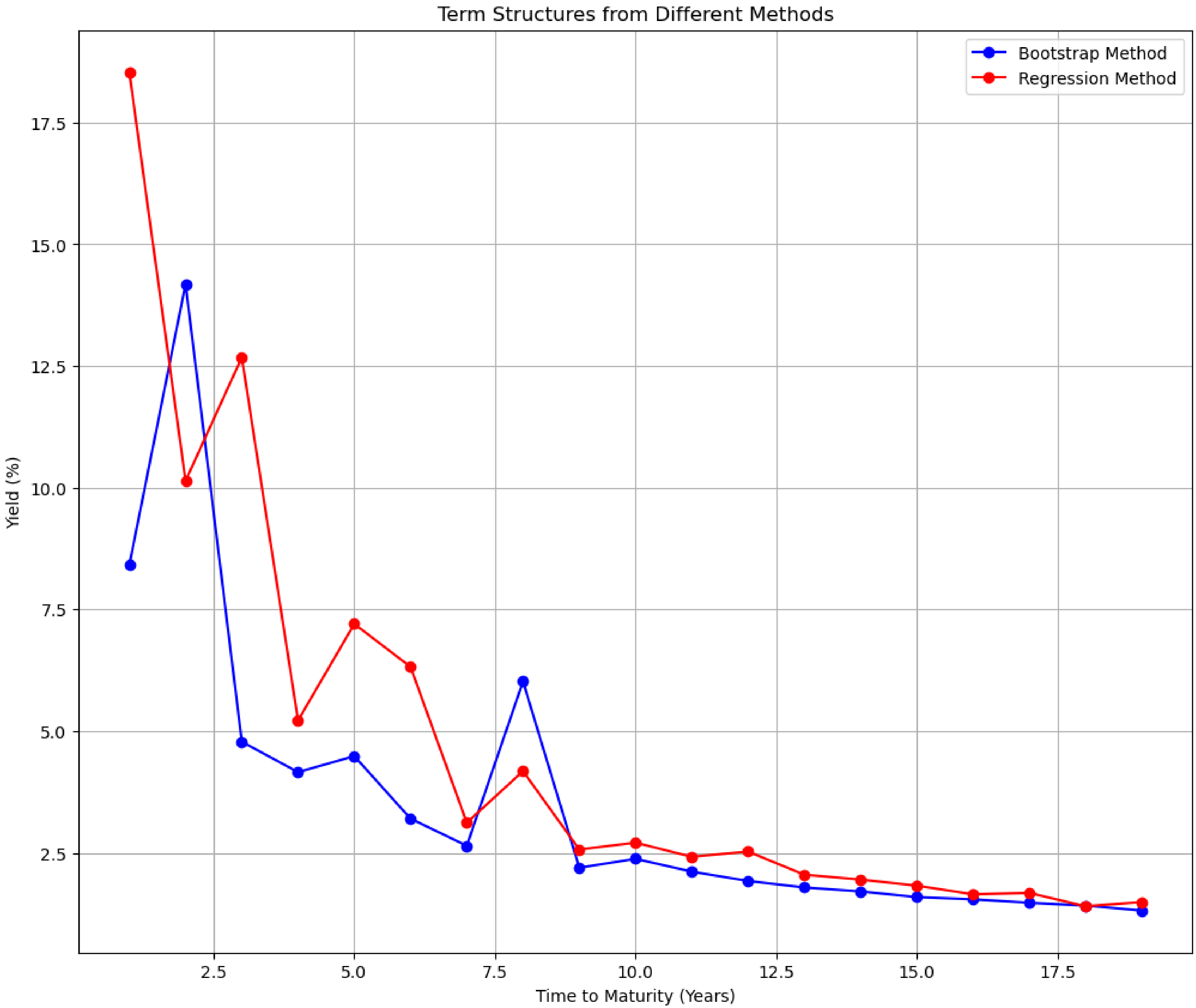
	Discount_Factors
0	0.335418
1	0.919311
2	0.753008
3	0.866359
4	0.846787

In [19]:
```
## Task 2 is to calculate the continously compounded zero-coupon bond yields for each maturity

# We will be using the same Discount_Factors as calculated above
df_bootstrap["Continously_Compounded_Yield"] = -np.log(df_bootstrap["Discount_Factors"]) / df_bootstrap.index

# Convert the yield to percent
df_bootstrap["Continously_Compounded_Yield_in%"] = df_bootstrap.Continously_Compounded_Yield * 100

# Plot the term structure
plt.figure(figsize = (10, 6))
plt.plot(df_bootstrap.index, df_bootstrap["Continously_Compounded_Yield_in%"], marker = "o")
plt.title("Term structure of Zero-Coupon Yields")
plt.xlabel("TMT in Years")
plt.ylabel("Yield in %")
plt.grid(True)
plt.show()
```


Term structure of Zero-Coupon Yields
Yield in %
TMT in Years
2
4
6
8
10
12
14
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Plotting BootStrap and Regression

In [20]:
```
# Initialize the figure
plt.figure(figsize=(12,10))

# Plotting the Bootstrap
plt.plot(df_bootstrap.index, df_bootstrap["Continously_Compounded_Yield_in%"], label = "Bootstrap Method", ma

# Plotting Regression
plt.plot(df_reg_discount.index, df_reg_discount["Continously_Compounded_Yield_in%"], label = "Regression Meth

# Adding titles and labels
plt.title('Term Structures from Different Methods')
plt.xlabel('Time to Maturity (Years)')
plt.ylabel('Yield (%)')
plt.legend()
plt.grid(True)

# Show plot
plt.show()
```


Term Structures from Different Methods
Bootstrap Method
Regression Method
Yield (%)
Time to Maturity (Years)
17.5
15.0
12.5
10.0
7.5
5.0
2.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5