Empirical Asset Pricing
Assignment 01

Morteza Aghajanzadeh* Ge Song†

January 28, 2024

Question 1

(a) Let's define the variables that we need to use in the estimation.

$$f(v_t,\theta)=\begin{bmatrix} R_{t1}-\mu_1 \\ R_{t2}-\mu_2 \\ (R_{t1}-\mu_1)^2-\sigma_1^2 \\ (R_{t2}-\mu_2)^2-\sigma_2^2 \end{bmatrix} \quad , \quad \theta=\begin{bmatrix}\mu_1\\ \mu_2\\ \sigma_1^2\\ \sigma_2^2\end{bmatrix}$$

$$g_T(\theta)= \frac{1}{T}\sum_{t=1}^{T} f(v_t,\theta)$$

We know from the lecture that we need to calculate the $\frac{\partial f}{\partial \theta'}$ to get the $\hat{D_T}$:

$$\frac{\partial f(v_t,\theta)}{\partial \theta'}=\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ -2(R_{t1}-\mu_1) & 0 & -1 & 0 \\ 0 & -2(R_{t2}-\mu_2) & 0 & -1 \end{bmatrix}$$

$$\Rightarrow \hat{D_T}=\frac{1}{T}\sum_{t=1}^{T}\frac{\partial f(v_t,\theta)}{\partial \theta'}=\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}=-I$$

We also know that $A_T = I$ and $A_T g_T(\theta) = 0$. Therefore, we can calculate the $\hat{\theta}$:

$$A_T g_T(\theta)=0 \Rightarrow \quad g_T(\theta)=0$$

$$g_T(\theta)=\begin{bmatrix} \frac{\sum R_{t1}}{T}-\mu_1 \\ \frac{\sum R_{t2}}{T}-\mu_2 \\ \frac{\sum (R_{t1}-\mu_1)^2}{T}-\sigma_1^2 \\ \frac{\sum (R_{t2}-\mu_2)^2}{T}-\sigma_2^2 \end{bmatrix}=0 \Rightarrow \begin{bmatrix} \frac{\sum_{t=1}^{T} R_{t1}}{T} \\ \frac{\sum_{t=1}^{T} R_{t2}}{T} \\ \frac{\sum_{t=1}^{T} (R_{t1}-\hat{\mu_1})^2}{T} \\ \frac{\sum_{t=1}^{T} (R_{t2}-\hat{\mu_2})^2}{T} \end{bmatrix}=\begin{bmatrix}\hat{\mu_1}\\ \hat{\mu_2}\\ \hat{\sigma_1^2}\\ \hat{\sigma_2^2}\end{bmatrix}=\hat{\theta}$$

Our calculated $\hat{\theta}$ based on the given data is:

$$\hat{\theta}=\begin{bmatrix}0.0162\\ 0.0045\\ 0.0212\\ 0.0167\end{bmatrix}$$

*Department of Finance, Stockholm School of Economics. Email: morteza.aghajanzadeh@phdstudent.hhs.se
†Department of Finance, Stockholm School of Economics. Email: ge.song@phdstudent.hhs.se

```
    mu_1 = sum(df['Stock1'])/len(df['Stock1'])
    mu_2 = sum(df['Stock2'])/len(df['Stock2'])
    sigma_1 = sum((df.Stock1 - mu_1)**2)/(len(df.Stock1))
    sigma_2 = sum((df.Stock2 - mu_2)**2)/(len(df.Stock2))
```

Listing 1: Python code for calculating $\hat{\theta}$

(b) Still we assume that there is no serial correlation in the moments. Therefore, we can calculate the $\hat{S_T}$ as follows:

$$\begin{aligned}
\hat{S_T} =& \frac{1}{T}\sum_{t=1}^{T} f(v_t,\theta)f(v_t,\theta)' \\
=& \frac{1}{T}\sum_{t=1}^{T}\begin{bmatrix} (R_{t1}-\mu_1)^2 & (R_{t1}-\mu_1)(R_{t2}-\mu_2) & (R_{t1}-\mu_1)^3-\sigma_1^2(R_{t1}-\mu_1) & (R_{t1}-\mu_1)(R_{t2}-\mu_2)^2-\sigma_2^2(R_{t1}-\mu_1) \\ (R_{t1}-\mu_1)(R_{t2}-\mu_2) & (R_{t2}-\mu_2)^2 & (R_{t1}-\mu_1)^2(R_{t2}-\mu_2)-\sigma_1^2(R_{t2}-\mu_2) & (R_{t2}-\mu_2)^3-\sigma_2^2(R_{t2}-\mu_2) \\ (R_{t1}-\mu_1)^3-\sigma_1^2(R_{t1}-\mu_1) & (R_{t1}-\mu_1)^2(R_{t2}-\mu_2)-\sigma_1^2(R_{t2}-\mu_2) & (R_{t1}-\mu_1)^4-2\sigma_1^2(R_{t1}-\mu_1)^2+\sigma_1^4 & (R_{t1}-\mu_1)^2(R_{t2}-\mu_2)^2-\sigma_1^2(R_{t1}-\mu_1)\sigma_2^2(R_{t2}-\mu_2) \\ (R_{t1}-\mu_1)(R_{t2}-\mu_2)^2-\sigma_2^2(R_{t1}-\mu_1) & (R_{t2}-\mu_2)^3-\sigma_2^2(R_{t2}-\mu_2) & (R_{t1}-\mu_1)^2(R_{t2}-\mu_2)^2-\sigma_1^2(R_{t1}-\mu_1)\sigma_2^2(R_{t2}-\mu_2) & (R_{t2}-\mu_2)^4-2\sigma_2^2(R_{t2}-\mu_2)^2+\sigma_2^4 \end{bmatrix} \\
=& \begin{bmatrix} \hat{\sigma_1}^2 & Cov(\hat{R_1},\hat{R_2}) & \hat{m}_1^{(3)} & \hat{k}_{12}^{(2)} \\ Cov(\hat{R_1},\hat{R_2}) & \hat{\sigma_2}^2 & \hat{k}_{21}^{(2)} & \hat{m}_2^{(3)} \\ \hat{m}_1^{(3)} & \hat{k}_{12}^{(2)} & \hat{m}_1^{(4)}-\hat{\sigma}_1^4 & \hat{k}_{12}^{(3)} \\ \hat{k}_{21}^{(2)} & \hat{m}_2^{(3)} & \hat{k}_{12}^{(3)} & \hat{m}_2^{(4)}-\hat{\sigma}_2^4 \end{bmatrix}
\end{aligned}$$

Here the definition of $\hat{m}_1^{(j)}$ is the one in the lecture notes. For the $\hat{k}_{mn}^{(j)}$, I do not have a good way to write it in the matrix form. Therefore, I just write it as a way to sum up the terms in the matrix.

As we know that two process are independent, and normally distributed, we can calculate the $\hat{S_T}$ as follows:

$$\hat{S_T} = \begin{bmatrix} \hat{\sigma_1}^2 & 0 & 0 & 0 \\ 0 & \hat{\sigma_2}^2 & 0 & 0 \\ 0 & 0 & 2\hat{\sigma_1}^4 & 0 \\ 0 & 0 & 0 & 2\hat{\sigma_2}^4 \end{bmatrix}$$

Our calculated $\hat{S_T}$ based on the given data is:

$$\hat{S_T} = \begin{bmatrix} 0.0212 & 0 & 0 & 0 \\ 0 & 0.0167 & 0 & 0 \\ 0 & 0 & 0.0347 & 0 \\ 0 & 0 & 0 & 0.0011 \end{bmatrix}$$

```
theta = np.array([mu_1,mu_2, sigma_1, sigma_2])
def f_v(theta,x):
    mu_1 = theta[0]
    mu_2 = theta[1]
    sigma_1 = theta[2]
    sigma_2 = theta[3]
    x_1 = x[0]
    x_2 = x[1]
    f = np.array([x_1-mu_1,x_2-mu_2,(x_1-mu_1)**2-sigma_1,(x_2-mu_2)**2-
        sigma_2]).reshape(len(theta),1)
    return f
def s(theta,x):
    f = f_v(theta,x)
    return f @ f.T
x = np.array([df[['Stock1','Stock2']]])[0]
s_hat = sum([s(theta,i) for i in x])/len(x)
# set non-diagonal elements to zero
```

```
s_hat = s_hat * np.eye(4)
```

Listing 2: Python function for calculating standard error of estimation

(c) Now we want to adjust the standard errors by Newey-West estimator. Therefore, we need to calculate the $\hat{\Gamma_1}$ by using the fact that two distributions are independent:

$$\hat{\Gamma_1} = \begin{bmatrix} \frac{\sum_2^T (R_t^1-\mu_1)(R_{t-1}^1-\mu_1)}{T-1} & 0 & \frac{\sum_2^T (R_t^1-\mu_1)(R_{t-1}^1-\mu_1)^2}{T-1} & 0 \\ 0 & \frac{\sum_2^T (R_t^2-\mu_2)(R_{t-1}^2-\mu_2)}{T-1} & 0 & \frac{\sum_2^T (R_t^2-\mu_2)(R_{t-1}^2-\mu_2)^2}{T-1} \\ \frac{\sum_2^T (R_{t-1}^1-\mu_1)(R_{t-1}^1-\mu_1)^2}{T-1} & 0 & \frac{\sum_2^T (R_{t-1}^1-\mu_1)^2(R_{t-1}^1-\mu_1)^2}{T-1} + \sigma_1^4 & 0 \\ 0 & \frac{\sum_2^T (R_{t-1}^2-\mu_2)(R_{t-1}^2-\mu_2)^2}{T-1} & 0 & \frac{\sum_2^T (R_{t-1}^2-\mu_2)^2(R_{t-1}^2-\mu_2)^2}{T-1} + \sigma_2^4 \end{bmatrix}$$

and then we can calculate the $\hat{S_T}$ as follows:

$$\hat{S_T} = \begin{bmatrix} \hat{\sigma_1}^2 & 0 & 0 & 0 \\ 0 & \hat{\sigma_2}^2 & 0 & 0 \\ 0 & 0 & 2\hat{\sigma_1}^4 & 0 \\ 0 & 0 & 0 & 2\hat{\sigma_2}^4 \end{bmatrix} + \frac{1}{2}(\hat{\Gamma}_1 + \hat{\Gamma}_1')$$

Our calculated $\hat{S_T}$ based on the given data is:

$$\hat{S_T} = \begin{bmatrix} 0.0162 & 0 & 0 & 0 \\ 0 & 0.0160 & 0 & 0 \\ 0 & 0 & 0.04 & 0 \\ 0 & 0 & 0 & 0.0013 \end{bmatrix}$$

```
theta = np.array([mu_1,mu_2, sigma_1, sigma_2])
lag = 1
def gamma(theta,x,lag):
    gamma = {}
    for i in range(1,lag +1):
        lag = np.array(df[['Stock1','Stock2']].shift(i).dropna())
        tempt = []
        for num,j in enumerate(x[i:]):
            tempt.append(f_v(theta,j) @ f_v(theta,lag[num]).T)
        gamma[i] = sum(tempt)/len(tempt)
    gamma = [gamma[i] for i in gamma]
    return sum(gamma)

def s_newywest(theta,x):
    gamma_hat = gamma(theta,x,lag)
    return sum([s(theta,i) for i in x])/len(x) + 0.5 * (gamma_hat +
        gamma_hat.T)
s_hat_newywest = s_newywest(theta,x)
s_hat_newywest * np.eye(4)
```

Listing 3: Python function for calculating Newy-West standard error

(d) Now we want to compare the Sharpe ratio of two stocks. Therefore, we need to test the hypothesis that:

$$\begin{cases} H_0 : \frac{\mu_1}{\sigma_1} = \frac{\mu_2}{\sigma_2} \\ H_1 : \frac{\mu_1}{\sigma_1} \neq \frac{\mu_2}{\sigma_2} \end{cases} \Rightarrow \begin{cases} H_0 : \mu_1\sigma_2 - \mu_2\sigma_1 = 0 \\ H_1 : \mu_1\sigma_2 - \mu_2\sigma_1 \neq 0 \end{cases}$$

now we can define the $R(\theta)$ as follows:

$$R(\theta) = \mu_1\sigma_2 - \mu_2\sigma_1$$

and then rewrite the hypothesis as follows:

$$H_0 : R(\theta) = 0$$
$$H_1 : R(\theta) \neq 0$$

Now we can use the Delta method to find the distribution of $R(\hat{\theta})$:

$$\sqrt{T}(R(\hat{\theta}) - R(\theta)) \xrightarrow{d} N(0, \frac{\partial R(\theta)}{\partial \theta'} V_\theta \frac{\partial R(\theta)}{\partial \theta'})$$
$$\sqrt{T}(R(\hat{\theta})) \xrightarrow{d} N(0, \frac{\partial R(\theta)}{\partial \theta'} V_\theta \frac{\partial R(\theta)}{\partial \theta'})$$

where V_θ is the variance of $\hat{\theta}$. Now we can calculate the $\frac{\partial R(\theta)}{\partial \theta'}$ as follows:

$$\frac{\partial R(\theta)}{\partial \theta'} = \begin{bmatrix} \sigma_2 & -\sigma_1 & -\mu_2 & \mu_1 \end{bmatrix}$$

and we know that $V_\theta = \hat{S_T}$. Therefore, we can find the distribution of $R(\hat{\theta})$ as follows:

$$\frac{\partial R(\theta)}{\partial \theta'} V_\theta \frac{\partial R(\theta)}{\partial \theta'} = \begin{bmatrix} \sigma_2 & -\sigma_1 & -\mu_2 & \mu_1 \end{bmatrix} \hat{S_T} \begin{bmatrix} \sigma_2 \\ -\sigma_1 \\ -\mu_2 \\ \mu_1 \end{bmatrix} = \hat{V}_T$$

Now we can calculate the test statistic as follows:

$$TR(\hat{\theta})' \hat{V}_T^{-1} R(\hat{\theta}) \xrightarrow{d} \chi_1^2$$
$$\frac{T(\mu_1\sigma_2 - \mu_2\sigma_1)^2}{\hat{S}_T} \xrightarrow{d} \chi_1^2$$

Let's calculate the test statistic:

$$\hat{V}_T = \begin{bmatrix} 0.0167 & -0.0212 & -0.0045 & 0.0162 \end{bmatrix} \begin{bmatrix} 0.0212 & 0 & 0 & 0 \\ 0 & 0.0167 & 0 & 0 \\ 0 & 0 & 0.0347 & 0 \\ 0 & 0 & 0 & 0.0011 \end{bmatrix} \begin{bmatrix} 0.0167 \\ -0.0212 \\ -0.0045 \\ 0.0162 \end{bmatrix} = 0.001449$$

therefore, the test statistic is 1.2602 and the p-value is 0.2612. Therefore, we cannot reject the null hypothesis under 5% significance level. Therefore, we can conclude that there is no significant difference between the Sharpe ratios of two stocks.

```
R_theta = mu_1*sigma_2 - mu_2*sigma_1
R_prime = np.array([sigma_2, -sigma_1, -mu_2, mu_1])
V_T = R_prime @ s_hat @ R_prime.T
test_stat = len(df) * (R_theta)**2 / V_T
```

Listing 4: Python code for calculating the test statistics

(e) Now we need to recalculate the standard error with the results in part (c). Our $\hat{V}_T$ is equal to 0.000005 and test statistics is 3.1692. The test statistics has been increased but it is still under the critical value of 3.841, which means that we cannot reject the null hypothesis.

(f) Now we use the bootstrap method to find a distribution of the Sharpe ratios. As we conducted the sample, we realized that our 95% confidence interval contains zero, which means that still we cannot reject the null hypothesis and two stocks' Sharpe ratio are statistically indifferent.

```
# Function to calculate Sharpe ratio
def calculate_sharpe_ratio(returns):
    return np.mean(returns) / np.std(returns)

# Parameters
n_iterations = 10000
confidence_level = 0.95

# Calculate observed Sharpe ratio difference
```

```
sharpe_ratio_diff_observed = calculate_sharpe_ratio(R1) -
    calculate_sharpe_ratio(R2)

# Bootstrap procedure

sharpe_ratio_diff_bootstrap = []
np.random.seed(123)
for _ in range(n_iterations):
    # Generate bootstrapped samples
    bootstrap_sample_stock1 = np.random.choice(R1, size=len(R1), replace=
        True)
    bootstrap_sample_stock2 = np.random.choice(R2, size=len(R2), replace=
        True)

    # Calculate Sharpe ratios for bootstrapped samples
    sharpe_ratio_stock1 = calculate_sharpe_ratio(bootstrap_sample_stock1)
    sharpe_ratio_stock2 = calculate_sharpe_ratio(bootstrap_sample_stock2)

    # Store the Sharpe ratio difference
    sharpe_ratio_diff_bootstrap.append(sharpe_ratio_stock1 -
        sharpe_ratio_stock2)

# Calculate 95% confidence interval
lower_bound, upper_bound = np.percentile(sharpe_ratio_diff_bootstrap, [(1
    - confidence_level) * 100 / 2, confidence_level * 100 - (1 -
    confidence_level) * 100 / 2])

print(f'Observed Sharpe Ratio Difference: {sharpe_ratio_diff_observed:.4f
    }')
print(f'95% Confidence Interval: [{lower_bound:.4f}, {upper_bound:.4f}]')
# The sharp ratio difference is not significant at 5% level.
```

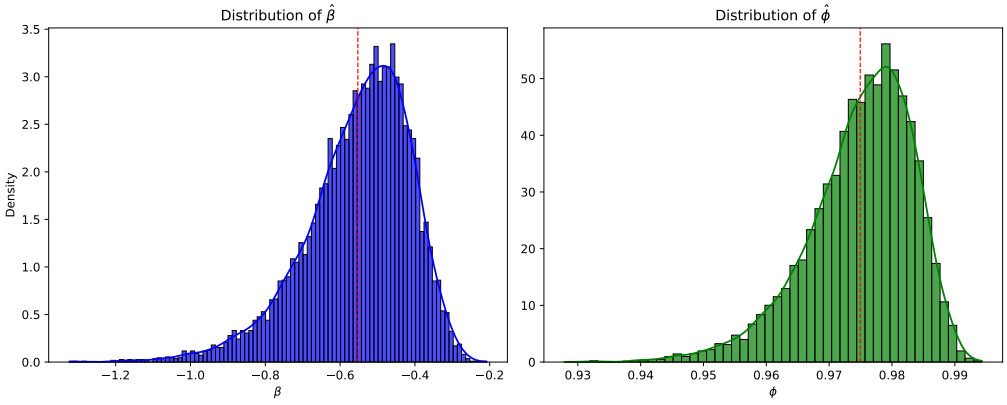
Listing 5: Python code for Bootstrapping

Question 2

Optional

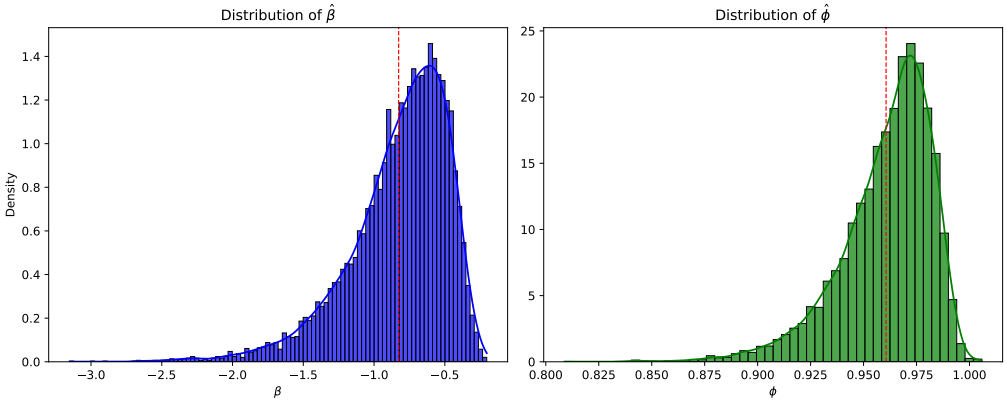
Question 3

(a) N=840



(b) N=240

Compared with the estimation with n=840, the estimation with n=240 is more biased and



away from the true value. This is probably because the sample size is too small to estimate the true value.

```
    def simulate_and_estimate():
    # Initialize arrays to store parameter estimates
    beta_hat_array = np.zeros(n_replications)
    phi_hat_array = np.zeros(n_replications)

    for rep in range(n_replications):
        beta_hat_array[rep], phi_hat_array[rep] = simulate()

    return beta_hat_array, phi_hat_array

def simulate():
    r, x = generate_data()
        # Estimate parameters with OLS
    beta, phi = estimate_parameters(r, x)
    return beta, phi

def generate_data():
    # Simulate data
    # Correlation matrix
    corr_mat= np.array([[1.0, corr_uv],
                        [corr_uv, 1.0]])

    # Compute the (upper) Cholesky decomposition matrix
    upper_chol = cholesky(corr_mat)

    # Generate 3 series of normally distributed (Gaussian) numbers
    rnd = np.random.normal(0.0, 1.0, size=(n_obs, 2))

    # Finally, compute the inner product of upper_chol and rnd
    errors = rnd @ upper_chol
    u, v = errors[:, 0], errors[:, 1]
    # Scale errors
    u *= sigma_u
    v *= sigma_v

    x = np.zeros(n_obs)
    r = np.zeros(n_obs)
    for t in range(1, n_obs):
        x[t] = theta + phi * x[t-1] + v[t]
        r[t] = alpha + beta * x[t-1] + u[t]
```

```
    return r, x

def estimate_parameters(r, x):
    # Estimate parameters with OLS
    model1 = sm.OLS(r, sm.add_constant(x))
    model2 = sm.OLS(x[1:], sm.add_constant(x[:-1]))
    results1 = model1.fit()
    results2 = model2.fit()
    # Store estimates
    beta_hat = results1.params[1]
    phi_hat = results2.params[1]
    return beta_hat, phi_hat
```

Listing 6: Python code for simulation

(c) At the confidence level of 5%, we rejected 50.51% of 10,000 simulations when when assuming IID errors, suggesting that we reject the null hypothesis. However, we should be expecting lower rejection rate since the real value of beta is 0.
This difference might be due to the fact that the errors are not independent.
New results [0.5051, 0.3815, 0.2619] for significance level 5%, 1%, 0.1% respectively

```
    def estimate_new_model(r,x):
    df = pd.DataFrame({'r':r,'x':x})
    df['12_month_return'] = df['r'].rolling(12).sum()
    # get the pvalue of the regression
    return sm.OLS(df['12_month_return'][11:].to_numpy(), sm.add_constant(
        df['x'][:-11]).to_numpy()).fit().pvalues[1]
def new_simulation():
    pvalue = np.zeros(n_replications)
    for rep in range(n_replications):
        r, x = generate_data()
        pvalue[rep] = estimate_new_model(r,x)
    return pvalue
```

Listing 7: Python code for new model

(d) We rejected 9.13% of the simulations out of 10,000 runs when using Newey-West standard errors, suggesting that there is autocorrelation present in the errors, and the standard errors are adjusted to account for this.
At the 5% significant level, we would not reject the null hypothesis: H0: beta=0. New results : [0.0913, 0.0326, 0.0067] for significance level 5%, 1%, 0.1% respectively

```
    def estimate_new_model_neweywest(r,x):
    df = pd.DataFrame({'r':r,'x':x})
    df['12_month_return'] = df['r'].rolling(12).sum()
    # get the pvalue of the regression
    return sm.OLS(df['12_month_return'][11:].to_numpy(), sm.add_constant(
        df['x'][:-11]).to_numpy()).fit(cov_type='HAC', cov_kwds={'maxlags
        ': 11}).pvalues[1]
def new_simulation_neweywest():
    pvalue = np.zeros(n_replications)
    for rep in range(n_replications):
        r, x = generate_data()
        pvalue[rep] = estimate_new_model(r,x)
    return pvalue
```

Listing 8: Python code for new model with Newey-West

Question 4

(a) We predict the out-of-sample returns based on three different models:

i. Using dividend-price ratio:

$$\hat{R}^e_{t,DP} = \hat{\alpha} + \hat{\beta}_{t-1} dp_{t-1}$$

ii. Using all the variables from Dong et al. (2022):

$$\hat{R}^e_{t,OLS} = \hat{\alpha} + \sum_{i=1}^{K} \hat{\beta}_{i,t-1} X_{i,t-1}$$

iii. Using combination-mean forecast:

$$\hat{R}^e_{t,CM} = \frac{1}{K} \sum_{i=1}^{K} \hat{R}^e_{t,i}$$

where, for each i,

$$\hat{R}^e_{t,i} = \hat{\alpha}_i + \hat{\beta}_{i,t-1} dp_{t-1}$$

where $\hat{\alpha}$ and $\hat{\beta}$ are estimated using OLS regression for the in-sample data which is an expanding window from 1970/01 until the previous month of the out-of-sample period. The out-of-sample period is from 1985/01 until 2017/12. We compare each model using the out-of-sample R^2 which is defined as:

$$R^2_{oc} = 1 - \frac{\sum_{t=1}^{T} (R^e_t - \hat{R}^e_t)^2}{\sum_{t=1}^{T} (R^e_t - \bar{R}^e_t)^2}$$

where R^e_t is the realized excess return and $\hat{R}^e_t$ is the predicted excess return. The out-of-sample R^2 for each model :

Model	R^2_{oc}
DP	-0.0237
OLS	-0.6856
CM	0.0128

Table 1: Out-of-sample R^2 for each model

As we can see, the out-of-sample R^2 for the DP and OLS model is negative which means that the DP model is not able to predict better than the benchmark which is the historical mean and the OLS model is worse than the DP model. However, the CM model has a positive out-of-sample R^2 which means that it is able to predict better than the benchmark.

```
data = pd.read_excel("Assignment1Data_G1.xlsx", sheet_name="
    Predictability")
data = data.dropna()
years = range(1970,2018)
periods = [int(str(i) + "0" + str(j)) for i in years for j in
    range(1,10) if len(str(j)) == 1]
periods.extend([int(str(i) + str(j)) for i in years for j in
    range(10,13) if len(str(j)) == 2 ])
periods.sort()
# %% Estimation
def prediction(X,y):
    beta = sm.OLS(y,X).fit().params.to_numpy()
    return X.iloc[-1].to_numpy() @ beta
BM_results = {}
DP_results = {}
OLS_results = {}
```

```
CM_results = {}
for prediction_period in tqdm([j for j in periods if j >=
    198501]):
    in_sample_period = [i for i in periods if i <
        prediction_period]
    in_sample_data = data[data["Month"].isin(in_sample_period)]
    X = sm.add_constant(in_sample_data["dp"])
    y = in_sample_data["ExcessRet"]
    BM_results[prediction_period] = y.mean()
    DP_results[prediction_period] = prediction(X,y)
    columns = list(data)
    columns.remove('Month')
    columns.remove('ExcessRet')
    columns.remove('Rfree')
    columns.remove('dp')
    X = sm.add_constant(in_sample_data[columns])
    OLS_results[prediction_period] = prediction(X,y)
    CM_list= []
    for i in columns:
        X = sm.add_constant(in_sample_data[i])
        CM_list.append(prediction(X,y))
    CM_results[prediction_period] = np.mean(CM_list)
```

Listing 9: Python code for prediction

(b) We need to perform the Diebold-Mariano test to test for the statistical significance of the difference between the out-of-sample R^2 of the models. The null hypothesis is that the difference between the out-of-sample R^2 is zero. The test statistic is defined as:

$$DM = \frac{\bar{d}}{\sqrt{\frac{1}{T}\sum_{t=1}^{T}(d_t - \bar{d}_t)^2}}$$

where $d_t = \hat{\varepsilon}_t^2 - \tilde{\varepsilon}_t^2$ and $\bar{d}_t = \frac{1}{T_{os}\sum d_t}$. Also, we use the Clark-West test which has a different definition for the d_t:

$$d_t = \hat{\varepsilon}_t^2 - [\tilde{\varepsilon}_t^2 - (\tilde{R}_t - \hat{R}_t)^2]$$

The calculated test statistics for each model are:

Model	DM	CW
DP	-1.418	-0.0606
OLS	-5.3738	1.321
CM	0.5262	2.0521

Table 2: Out-of-sample R^2 for each model

The critical values for the Diebold-Mariano test are -1.96 and 1.96 for the two-tailed test. Since the test statistics for the DP and CM models are not statistically significant, we cannot reject the null hypothesis that the difference between the out-of-sample R^2 is zero. On the other hand, the test statistic for the OLS model is statistically significant which means that the out-of-sample R^2 of the OLS model is statistically predict less than benchmark model due to the negative sign of the test statistic.

In addition, we can see that the test statistics for the Clark-West test are different from the Diebold-Mariano test. The critical values for the Clark-West test are the same as before. The test statistic for the DP and OLS model are not statistically significant which means that the null hypothesis cannot be rejected. However, the test statistic for the CM model is positive and statistically significant which means that the model is statistically predicts better than benchmark model.

```
    def DM_test(y_tilde, y_hat):
T = len(y_hat)
d = y_tilde**2 - y_hat**2
delta_hat = np.mean(d)
# sigma_hat = np.sqrt(np.sum((d - delta_hat)**2)/(T-1))
# Newey-West correction with on lag
sigma_hat = np.sqrt(np.sum((d - delta_hat)**2)/(T-1) + 2*np.sum([d[i
    ]*d[i-1] for i in range(1,T)])/(T-1))

DM = delta_hat/sigma_hat * np.sqrt(T)
return DM

def Clark_West_test(y_tilde, y_hat, R_tilde, R_hat):
T = len(y_hat)
d = y_tilde**2 - (y_hat**2 - (R_tilde - R_hat)**2)
delta_hat = np.mean(d)
# sigma_hat = np.sqrt(np.sum((d - delta_hat)**2)/(T-1))
sigma_hat = np.sqrt(np.sum((d - delta_hat)**2)/(T-1) + 2*np.sum([d[i
    ]*d[i-1] for i in range(1,T)])/(T-1))
CW = delta_hat/sigma_hat * np.sqrt(T)
return CW
```

Listing 10: Python code for Diebold-Mariano test

(c) Now we want to compare the cumulative returns of the portfolios constructed based on the predicted excess returns of each model. The cumulative returns are calculated as:

$$R_{t+1} = R_t + \hat{\omega}^*_{t,j} R^e_t$$

where $\hat{\omega}^*_{t,j}$ is the optimal weight of the portfolio at time t and j is the model. The optimal weights are calculated as:

$$\hat{\omega}^*_{t,j} = \begin{cases} 2 & \text{if } \hat{\omega}_{t,j} \geq 2 \\ \hat{\omega}_{t,j} & \text{if } -1 < \hat{\omega}_{t,j} < 2 \\ -2 & \text{if } \hat{\omega}_{t,j} \leq -1 \end{cases} \quad \text{where} \quad \hat{\omega}_{t,j} = \frac{1}{\gamma} \frac{\hat{R}^e_{t,j}}{\hat{\sigma}^2_{R^e,t}}$$

where γ is the risk aversion parameter and $\hat{\sigma}^2_{R^e,t}$ is the variance of the predicted excess returns over the last 60 months. The cumulative returns of the portfolios are shown in the following figure. We can see that the cumulative returns of the portfolios based on the DP and OLS models could not beat the benchmark which is the historical mean. However, the cumulative returns of the portfolio based on the CM model is able to beat the benchmark.

It is interesting to see that the prediction models during the dot com bubble cannot beat the benchmark. In addition, the DP model lose its power to predict after the dot com bubble which also become worse than OLS model.

```
prediction_data['rolling_var'] = prediction_data.ExcessRet.rolling
    (60).var()
gamma = 2
for prediction_model in ['BM',"DP", "OLS", "CM"]:
    prediction_data['omega_hat'] = prediction_data[prediction_model]/
        prediction_data['rolling_var']/gamma
    prediction_data['omega_hat_'+prediction_model] = prediction_data[
        'omega_hat']
    prediction_data.loc[prediction_data.omega_hat >= 2,'omega_hat_'+
        prediction_model] = 2
    prediction_data.loc[prediction_data.omega_hat <= -1,'omega_hat_'+
        prediction_model] = -1
    prediction_data['r_p_'+prediction_model] = prediction_data['
        ExcessRet'] + prediction_data['omega_hat_'+prediction_model]*
        prediction_data['ExcessRet']
```

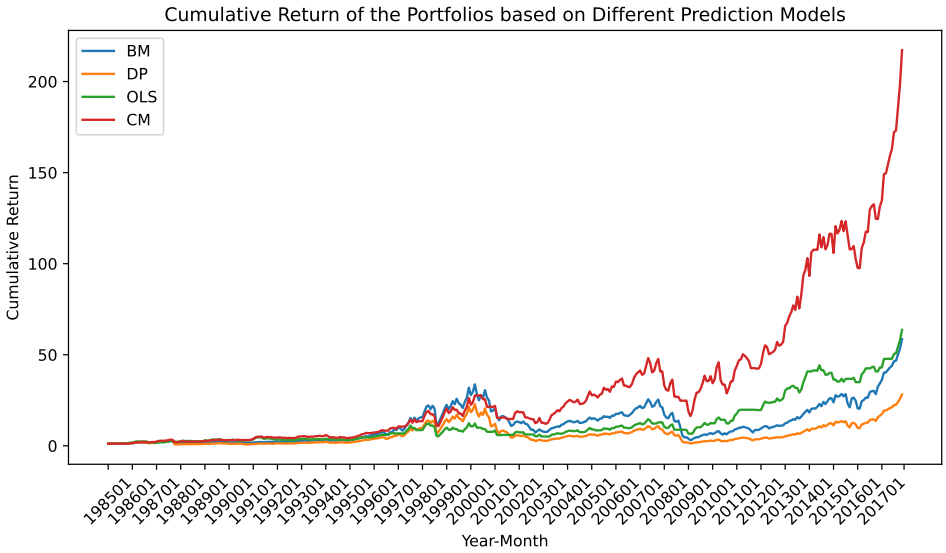


Figure 1: Cumulative returns of the portfolios

Listing 11: Python code for portfolio weights

(d) Now we want to plot the optimal weights of the portfolios. The optimal weights are calculated as mentioned before. The optimal weights of the portfolios are shown in following Figure. We can see that the optimal weights of the portfolios based on the OLS and CM models are not stable and they change over time. However, the optimal weights of the portfolio based on the BM and DP are stable and they do not change over time. It is interesting to see that the weights from the DP model and the BM model are highly correlated.

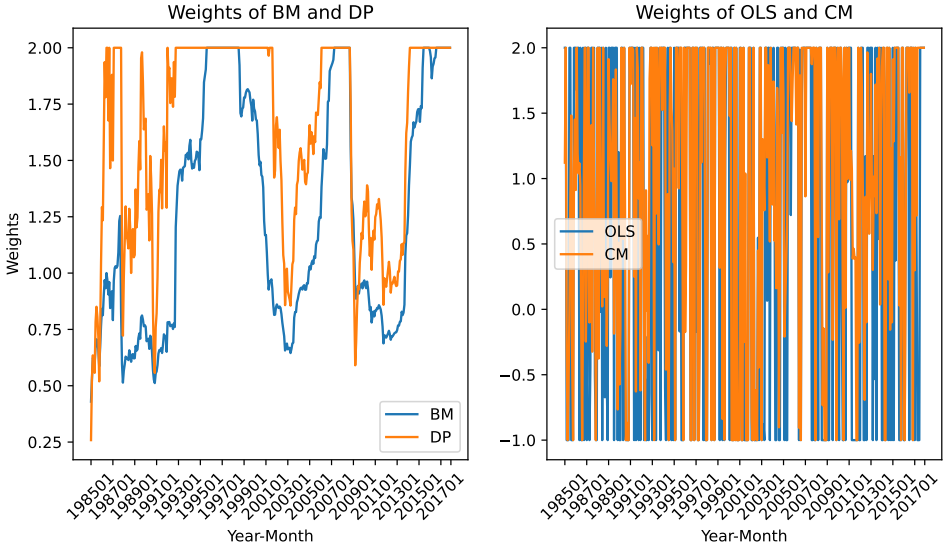


Figure 2: Optimal weights of the portfolios

(e) Now we define a utility function for the investor as:

$$U(\hat{R}^j_{p,t}) = \hat{E}\left[\hat{R}^j_{p,t}\right] - \frac{\gamma}{2}\hat{Var}\left(\hat{R}^j_{p,t}\right)$$

and we want to compare the utility of the portfolios based on the predicted excess returns of each model with the utility of the benchmark which is the historical mean. The utility gain is defined as:

$$UG = U(\hat{R}^{j}_{p,t}) - U(\hat{R}^{BM}_{p,t})$$

The utility gain for each model is shown in the following table. We can see that the utility gain for the DP model is negative which means that the utility of the portfolio based on the DP model is less than the utility of the benchmark. However, the utility gain for the OLS and CM models are positive which means that the utility of the portfolios based on these models are higher than the utility of the benchmark.

In this comparison, we can see that the utility gain for the CM model is the highest which means that the portfolio based on the CM model is the best portfolio among the other portfolios.

Model	UG
DP	-0.003
OLS	0.0013
CM	0.0043

Table 3: Utility gain for each model

Question 6

In this question we will redo the portfolio analysis of the previous question, but this time we will use the historical average to predict the future returns and use two different methods to estimate the variance. The first method is calculating the sample variance over the past 60 months (RW) and the second method is the $GARCH(1,1)$ model ($GARCH$)over the expanding window from the beginning of the sample to the current month.
We used the same method as in question 4 to calculate the optimal weights of the portfolios. The results are shown in the following figures. Our calculation for the utility gain shows that the utility gain of the $GARCH$ model is 0.0029 and the RW model is 0.006 with respect to the historical average. We can see that the $GARCH$ model has a better performance than the RW model but due to the higher volatility of the $GARCH$ portfolio the utility gain is lower than the RW model.

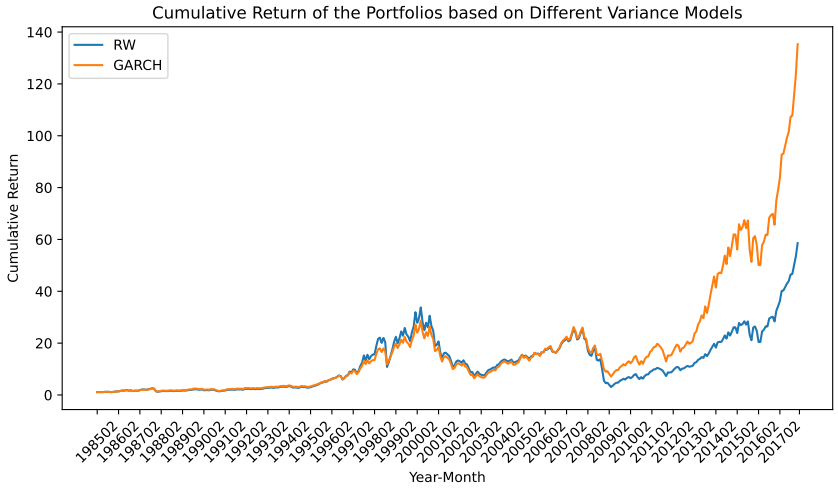


Figure 3: Optimal weights of the portfolios

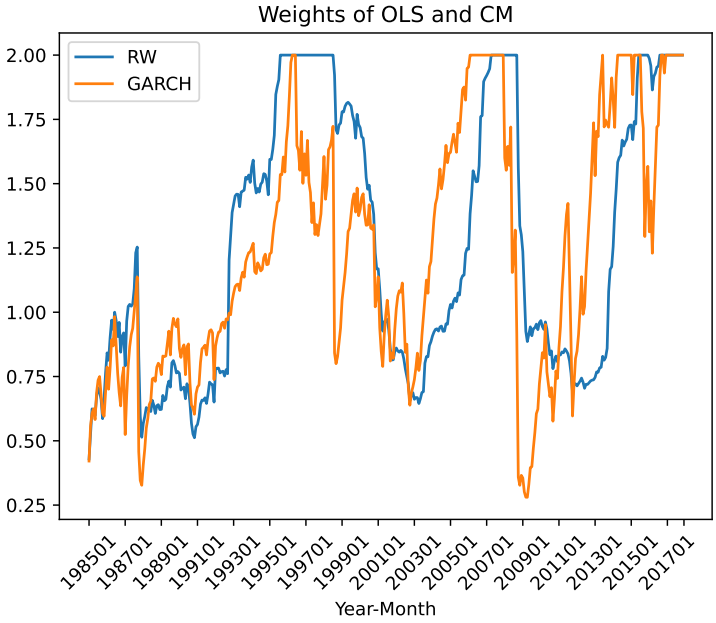


Figure 4: Optimal weights of the portfolios