DATA 609 - Homework 2: Applications of Least Squares

Eddie Xu

Instructions

Please submit a .qmd file along with a rendered pdf to the Brightspace page for this assignment. You may use whatever language you like within your qmd file, I recommend python, julia, or R.

Problem 1: Online Updating for Least Squares and Autoregressive Time Series Models

Many applications of least squares (and other statistical methods) involve *streaming data*, in which data is collected over a time period and the statistical model is updated as new data arrives. If the quantity of data arriving is very large, it may be inefficient or even impossible to refit the entire model on the entire dataset. Instead, we use techniques often referred to as *online learning* which take the current model as a starting point and update them to incorporate the new data.

The structure of least squares problems makes them amenable to online updating (sometimes this is called "recursive" least squares). The structure of the problem is as follows, at time t we receive a vector of observations $\mathbf{a}_t$ and an observation of our target variable b_t.

The full set of all observations and target variable data that we have received up to time t is contained in the following matrix and vector:

$$A_{(t)} = \begin{bmatrix} \cdots \, \mathbf{a}_1^T \, \cdots \\ \cdots \, \mathbf{a}_2^T \, \cdots \\ \vdots \\ \cdots \, \mathbf{a}_t^T \, \cdots \end{bmatrix}, \quad \mathbf{b}_{(t)} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_t \end{bmatrix}$$

which says that row j of the matrix $A_{(t)}$ is the jth observation $\mathbf{a}_j^T$.

The jth entry of $\mathbf{b}_{(t)}$ is b_j.

Here we assume that the vectors $\mathbf{a}$ each contain n observations, so that $A_{(t)}$ is a $t \times n$ matrix and the vector $\mathbf{b}_{(t)}$ is a t-vector.

As long as $t > n$, i.e. the number of time observations is greater than the number of features/data points in each observation, we can fit a linear model predicting the target as a function of the features $\mathbf{a}$ by solving the following system of equations:

$$(A_{(t)}^T A_{(t)})\mathbf{x}_{(t)} = A_{(t)}^T \mathbf{b}_{(t)}$$

As the length of the time series increases, the computational difficulty of solving this problem also increases. However, it is possible to re-use work done on the previous time step to avoid solving the full system at each step.

This algorithm is based on the fact that the *Gram Matrix* $A_{(t+1)}^T A_{(t+1)}$ can be calculated from the Gram matrix $A_{(t)}^T A_{(t)}$ from the previous time step.

$$A_{(t+1)}^T A_{(t+1)} = A_{(t)}^T A_{(t)} + \mathbf{a}_{t+1}\mathbf{a}_{t+1}^T$$

Similarly, the product $A_{(t+1)}^T \mathbf{b}_{(t+1)}$ can also be updated from thevalue on the previous time step:

$$A_{(t+1)}^T \mathbf{b}_{(t+1)} = A_{(t)}^T \mathbf{b}_{(t)} + b_{t+1}\mathbf{a}_{t+1}$$

We can write an efficient algorithm to compute the updated least squares solution as follows:

- Step 1: Pick an initial time t such that A_t is square or tall so that the least squares problem can be solved (i.e. wait for enough data to have built up before your start) and then calculate the Gram matrix and the product $A_t^T \mathbf{b}_t$

$$G_{(t)} = A_{(t)}^T A_{(t)}, \quad \mathbf{h}_{(t)} = A_t^T \mathbf{b}_t$$

- Step 2: Find the least squares solution at time t by solving the linear system:

$$G_{(t)}\mathbf{x}_{(t)} = \mathbf{h}_{(t+1)}$$

- Step 3: When the next data points $\mathbf{a}_{t+1}$ and b_{t+1}, update $G_{(t+1)}$ and $\mathbf{h}_{(t+1)}$:

$$G_{(t+1)} = G_{(t)} + \mathbf{a}_{t+1}\mathbf{a}_{t+1}^T, \quad \mathbf{h}_{(t+1)} = \mathbf{h}_{(t)} + b_{t+1}\mathbf{a}_{t+1}$$

Then you can repeat Step 2 to find $\mathbf{x}_{t+1}$. This algorithm can be improved upon slightly using the Matrix Inversion Lemma/Woodbury Formula, which could be a topic for a project (see note at the end which mentions Kalman filters).

(a) You are going to use this algorithm to make a linear, autoregressive model that predicts total day-ahead citibike trips from the daily high temperature and the number of daily citibike trips taken each of the past 7 days. The data is contained in the file daily_citibike_trips.csv.

Specifically, for each time point $t > 7$, fit the following model as a least squares estimation problem:

$$N_{trips,\tau} = \sum_{i=1}^{7} C_i N_{trips,\tau-i} + C_T T,$$

Here, each $N_{trips,\tau}$ stands for the number of citibike trips on the tth day of the time series, T stands for the forecast high temperature in New York City that day, and the coefficients C_i and C_T are the decision variables.

Find the coefficients $C_{i,t}$ and $C_{T,t}$ that minimize the mean square errors on all the observed citibike trips prior to time t. Use the recursive least squares optimization outlined in the preamble to this problem to calculate the coefficients for each time point, and plot how they and the R^2 of the model change over time.

What patterns do you notice in how the regression coefficients and R^2 change over time?

Tip: Be very cautious when coding about the dimensionality of matrices and arrays. In python, `a @ a.T` will be an 8x8 matrix if `a.shape = (8,1)`. However, by default `a.shape = (8,)`, indicating that `a` is not being treated as either a row or column vector. For this problem it is important that the vectors are either row or column vectors, and not arrays without such an orientation. In python, you can use `numpy.reshape` to adjust.

(b) We have included temperature as a variable because it probably influences the decision to ride a citibike. However, the relationship could be nonlinear, as both extreme high and low temperatures make bike riding less comfortable. With this idea in mind, create a new feature by choosing a nonlinear function of the temperature T that represents the potential for both high and low values of T to the same impact on predicted ridership.

Use least squares optimization to fit an autoregressive time series model, replacing T with the value of your new feature $f(T)$ in the time series. How does the temperature dependence coefficient differ between this model and the one you fit in (a)? Does the accuracy of the model improve or get worse using the new feature?

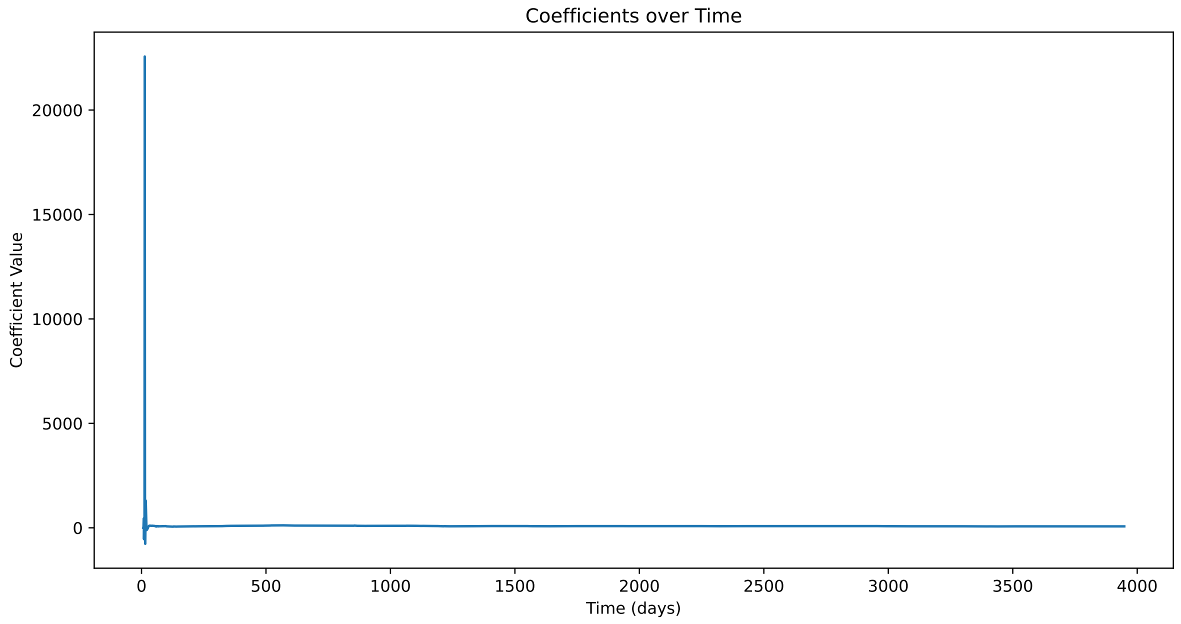
Problem 1 Solution

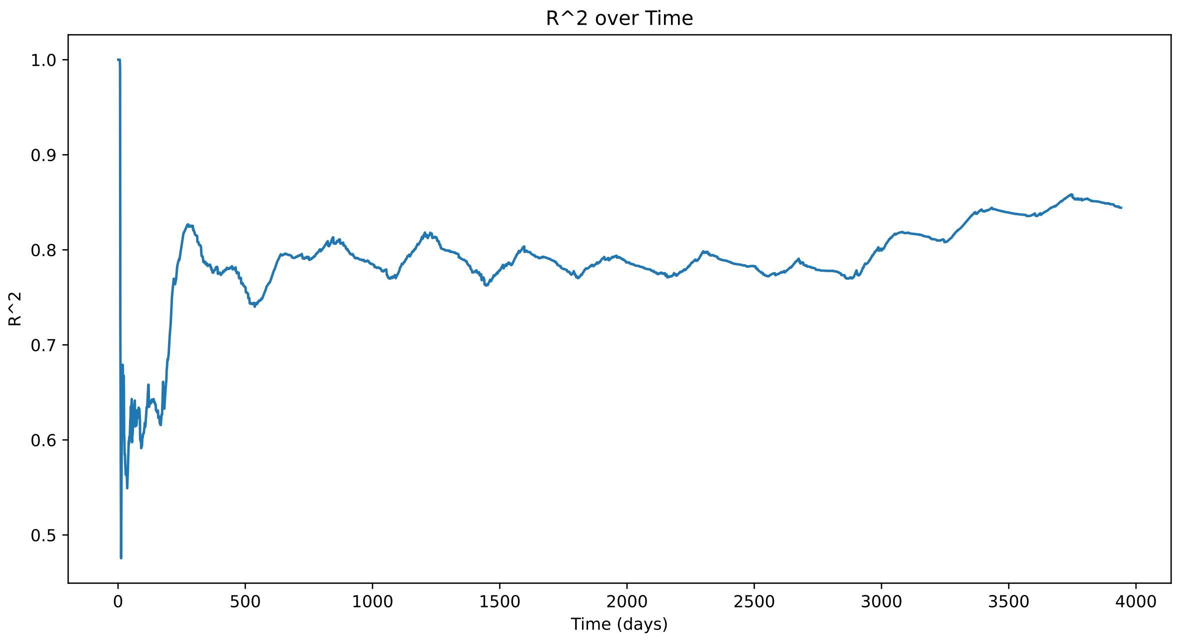
```python
# load dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# read in data
bike_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/refs/I
bike_data = pd.read_csv(bike_url)

### Problem 1(a)

# extract values and define variables
temp_length = len(bike_data)
trips = bike_data['daily_trips'].values
temp = bike_data['TMAX'].values
days = 7

# set up gram matrix and vectors
G = np.zeros((days + 1 , days + 1))
h = np.zeros(days + 1)

# define other variables for RLS
coefficients, r_squared = [], []

# perform the RLS
for t in range(days, len(trips)):
    a_t = np.concatenate([trips[t-days:t][::-1], [temp[t]]])
    b_t = trips[t]

    if t == days:
        # append the matrix and vector
        G = np.outer(a_t, a_t)
        h = b_t * a_t
    else:
        G += np.outer(a_t, a_t)
        h += b_t * a_t


    # solve for coefficients
    x_t = np.linalg.solve(G, h)
    coefficients.append(x_t)
    # solve for r_squared
    predictions = []

    # collect prediction based on previous points
    for i in range(days, t+1):
        a_i = np.concatenate([trips[i-days:i][::-1], [temp[i]]])
        predictions.append(np.dot(a_i, x_t))

    predictions = np.array(predictions)
    residual = trips[days:t+1] - predictions

    # calculate the ss_total based on all riderships until current points
    ss_total = np.sum((trips[days:t+1] - np.mean(trips[days:t+1]))**2)
    ss_residual = np.sum(residual**2)
    r_squared.append(1 - (ss_residual / ss_total))

# save both coefficients and r squared in an array
coefficients = np.array(coefficients)
r_squared = np.array(r_squared)

# plot results
plt.figure(figsize=(12, 6))
plt.plot(range(days, len(trips)), coefficients[:, days])
plt.xlabel('Time (days)')
plt.ylabel('Coefficient Value')
plt.title('Coefficients over Time')
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(r_squared, label='R^2')
plt.xlabel('Time (days)')
plt.ylabel('R^2')
plt.title('R^2 over Time')
plt.show()

### Problem 1(b)

# calculate the nonlinear temperature
nonlinear_temp = (temp - 25) ** 2

# define new gram matrix and vector
G_nonlinear = np.zeros((days + 1 , days + 1))
```

```python
h_nonlinear = np.zeros(days + 1)

# define other variables for RLS
coefficients_nonlinear, r_squared_nonlinear = [], []

# perform the RLS for nonlinear temperature
for t in range(days, len(trips)):
    a_t_nl = np.concatenate([trips[t-days:t][::-1], [nonlinear_temp[t]]])
    b_t_nl = trips[t]

    if t == days:
        # append the nonlinear matrix and vector
        G_nonlinear = np.outer(a_t_nl, a_t_nl)
        h_nonlinear = b_t_nl * a_t_nl
    else:
        G_nonlinear += np.outer(a_t_nl, a_t_nl)
        h_nonlinear += b_t_nl * a_t_nl


    # solve for coefficients
    x_t_nl = np.linalg.solve(G_nonlinear, h_nonlinear)
    coefficients_nonlinear.append(x_t_nl)

    # solve for r_squared
    predictions_nonlinear = []

    # collect prediction based on previous points
    for i in range(days, t+1):
        a_i_nl = np.concatenate([trips[i-days:i][::-1], [nonlinear_temp[i]]])
        predictions_nonlinear.append(np.dot(a_i_nl, x_t_nl))

    predictions_nonlinear = np.array(predictions_nonlinear)
    residual_nl = trips[days:t+1] - predictions_nonlinear

    # calculate the ss_total based on all riderships until current points
    ss_total_nl = np.sum((trips[days:t+1] - np.mean(trips[days:t+1]))**2)
    ss_residual_nl = np.sum(residual_nl**2)
    r_squared_nonlinear.append(1 - (ss_residual_nl / ss_total_nl))

# save both nonlinear coefficients and r squared in an array
coefficients_nonlinear = np.array(coefficients_nonlinear)
r_squared_nonlinear = np.array(r_squared_nonlinear)
```

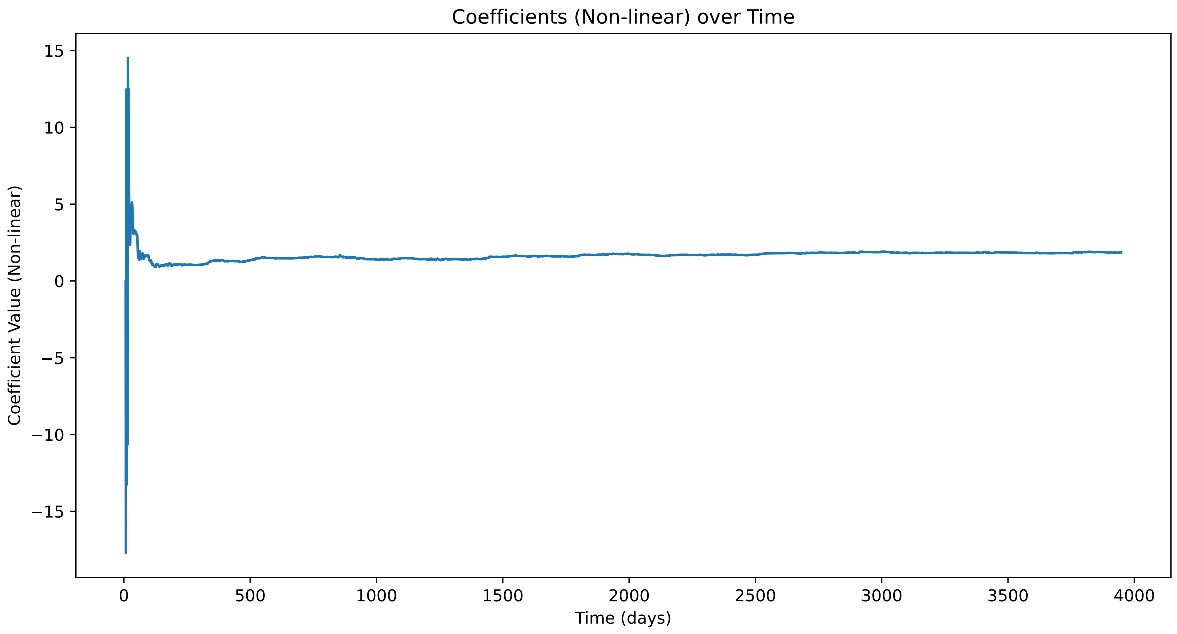
```
# plot results
plt.figure(figsize=(12, 6))
plt.plot(range(days, len(trips)), coefficients_nonlinear[:, days])
plt.xlabel('Time (days)')
plt.ylabel('Coefficient Value (Non-linear)')
plt.title('Coefficients (Non-linear) over Time')
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(r_squared, label='R^2')
plt.xlabel('Time (days)')
plt.ylabel('R^2 (Non-linear)')
plt.title('R^2 (Non-lienear) over Time')
plt.show()
```

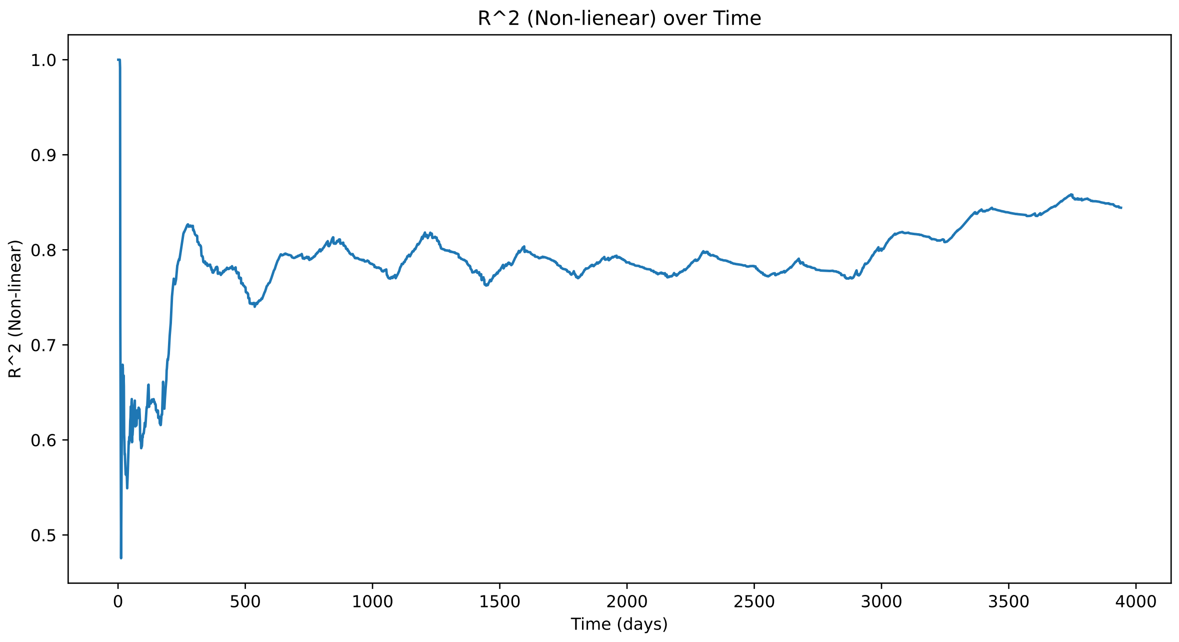
```
/var/folders/h4/zjq554hs0b57vqfcrc5738wh0000gn/T/ipykernel_28661/3008040661.py:57: RuntimeWa

invalid value encountered in scalar divide
```





```
/var/folders/h4/zjq554hs0b57vqfcrc5738wh0000gn/T/ipykernel_28661/3008040661.py:122: RuntimeWa

invalid value encountered in scalar divide
```





Problem 2: Weighted Least Squares

The file social-mobility.csv contains data on the fraction of individuals born in the years 1980-1982 to parents in the bottom 20% of the income distribution who reach the top 20% of the income distribution by the time they turn 30 in a large number of municipalities throughout the United States. The dataset also contains additional variables that describe other socio-economic differences between the cities in the dataset.

(a) Make a scatter-plot of mobility versus population (use a log-scale for population). What do you notice about the variance of social mobility as a function of population? This is a common feature of nearly every dataset containing geographic regions with widely different populations.

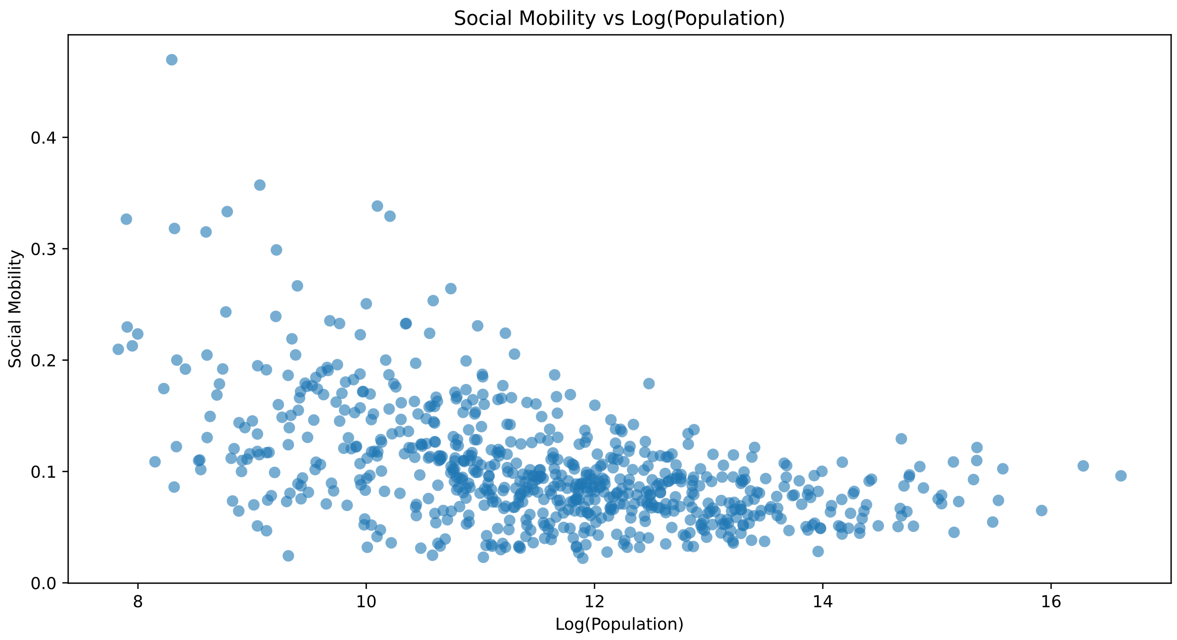
(b) Assume that the number of children born in families making below the 20th percentile of the income distribution in each city is linearly proportional to the city population. Write down a formula for how the variance of each measurement of the social mobility should depend on the measured social mobility and the population. Hint: start with either the formula for the variance of binomial counts or look up the variance of a proportion derived from a binomial distribution. Don't worry about constant factors when deriving this formula.

(c) Use weighted least squares to calculate an estimate of how social mobility depends on commute time and student-teacher ratio, using weights calculated based on the variance estimate derived in (b).

Compare the coefficients to those derived from ordinary least squares with no weights.

Problem 2(a) and 2(b) Solution

```
# load dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# read in data
social_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/refs
social_mobility_data = pd.read_csv(social_url)

## Problem 2(a)

# log tranform the population and define the variable for mobility
log_pop = np.log(social_mobility_data['Population'])
mobility = social_mobility_data['Mobility']

# plot
plt.figure(figsize=(12, 6))
plt.scatter(log_pop, mobility, alpha=0.6)
plt.xlabel('Log(Population)')
plt.ylabel('Social Mobility')
plt.title('Social Mobility vs Log(Population)')
plt.show()
```



The binomial distribution formula is defined below:

$$Variance = n * p * (1 - p)$$

Problem 2(c) Solution

```
# load dependencies
import statsmodels.api as sm
import pandas as pd

# read in data
social_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/refs
social_mobility_data = pd.read_csv(social_url)

social_mobility_data = social_mobility_data[["Commute","Student_teacher_ratio","Mobility","Po
social_mobility_data = social_mobility_data.dropna()

# define variable
commute = social_mobility_data['Commute']
ratio = social_mobility_data['Student_teacher_ratio']
population = social_mobility_data['Population']
mobility = social_mobility_data['Mobility']
```

```
# define both X and Y
X = pd.DataFrame({'commute':commute, 'ratio':ratio})
X = sm.add_constant(X)
print(X)
Y = mobility
weights = (mobility * (1 - mobility))/population

# OLS model
ols_model = sm.OLS(Y, X).fit()

# WLS model
wls_model = sm.WLS(Y, X, weights=weights).fit()

print("OLS Coefficients")
print(ols_model.summary())

print("WLS Coefficients")
print(wls_model.summary())
```

```
     const  commute  ratio
0      1.0    0.359   15.1
1      1.0    0.292   15.4
2      1.0    0.305   16.7
3      1.0    0.289   16.2
4      1.0    0.325   12.3
..     ...      ...    ...
706    1.0    0.579   15.1
707    1.0    0.628   18.3
708    1.0    0.418   21.1
709    1.0    0.486   19.5
710    1.0    0.240   21.0

[699 rows x 3 columns]
OLS Coefficients
                            OLS Regression Results
==============================================================================
Dep. Variable:               Mobility   R-squared:                       0.356
Model:                            OLS   Adj. R-squared:                  0.354
Method:                 Least Squares   F-statistic:                     192.6
Date:                Tue, 01 Apr 2025   Prob (F-statistic):           2.72e-67
Time:                        00:57:46   Log-Likelihood:                 1213.6
```

```
No. Observations:                 699   AIC:                            -2421.
Df Residuals:                     696   BIC:                            -2408.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0360      0.016      2.214      0.027       0.004       0.068
commute        0.2118      0.013     16.349      0.000       0.186       0.237
ratio         -0.0019      0.001     -2.471      0.014      -0.003      -0.000
==============================================================================
Omnibus:                      250.548   Durbin-Watson:                   1.366
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1772.747
Skew:                           1.424   Prob(JB):                         0.00
Kurtosis:                      10.264   Cond. No.                         201.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
WLS Coefficients
                            WLS Regression Results
==============================================================================
Dep. Variable:               Mobility   R-squared:                       0.178
Model:                            WLS   Adj. R-squared:                  0.175
Method:                 Least Squares   F-statistic:                     75.24
Date:                Tue, 01 Apr 2025   Prob (F-statistic):           2.61e-30
Time:                        00:57:46   Log-Likelihood:                 254.19
No. Observations:                 699   AIC:                            -502.4
Df Residuals:                     696   BIC:                            -488.7
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1261      0.031      4.121      0.000       0.066       0.186
commute        0.2637      0.030      8.809      0.000       0.205       0.322
ratio         -0.0078      0.001     -5.786      0.000      -0.010      -0.005
==============================================================================
Omnibus:                      860.129   Durbin-Watson:                   1.620
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           169885.547
Skew:                           5.855   Prob(JB):                         0.00
Kurtosis:                      78.471   Cond. No.                         196.
==============================================================================
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Problem 3: Markowitz Portfolio Optimization

In this problem you will use *Markowitz Portfolio Optimization* to construct a set of portfolios that aim to achieve target expected rates of return while minimizing risk. The file stock_returns.csv contains information on daily asset returns from 2020-2024 for a group of assets, consisting mostly of large-cap stocks but also a handful of exchange traded funds that correspond to US Treasury Bonds and Notes with varying maturities.

You will divide the data into two time periods, a training period (2020-2022) and a testing period (2022-2024). The data in the `stock_returns.csv` is stored in a long format, with the following variables:

1. `Company` - The ticker symbol that identifies the stock
2. `date` - The date to which the data corresponds
3. `adjusted`- the closing price of the stock, adjusted for special events like dividends and stock splits
4. `return` - the ratio of the current adjusted close to the adjusted close on the previous trading day
5. `log_return`- the natural logarithm of the return

(a) Construct a vector (we call it μ) containing the annualized rate of return over the training period (for the ith stock, you can use the formula: $\mu_i = \exp\left(0.5 \sum_t \text{log\_return}_i(t)\right)$), or the square root of the total return over the first two years of data, and daily return covariance Γ.

Hint: Possible workflow if working in python: convert your data to a wide format using `pandas`, extract the values into a `numpy ndarray`, and then use the function `np.cov`.

Then solve the following constrained least squares problem to calculate optimal portfolios achieving a fixed rate of return with minimum variance:

$$\begin{aligned} \min_w \mathbf{x}^T \Gamma \mathbf{x}&, \\ \mathbf{w}^T &= r, \\ \sum_i w_i &= 1 \end{aligned}$$

Here $\mathbf{w}$ is a vector containing the investment allocations into different assets, and r is the target rate of return.

Calculate optimal portfolios based on the 2020-2022 data for $r = 1.05$, $r = 1.10$, and $r = 1.20$.

(b) Plot the cumulative value of each portfolio over time, assuming that an initial investment is made at the start of the period and that there is no rebalancing of the portfolio, i.e. $r_T = \sum_{i=1}^{n} w_i \Pi_{t=1}^{T} r_{it}$, where r_{it} is the return of asset i on trading day t.

(hint: `np.cumprod` allows you to efficiently calculate this quantity).

Make the plot for both the training and test sets of returns.

For each of the 3 portfolios also report:

- The annualized return on the training and test sets;
- The risk on the training and test sets, defined as the realized variance of the daily return;
- The asset with the maximum allocation weight, and its weight;
- The initial leverage, defined as $\sum_{i=1}^{n} |w_i|$. This number is always at least one, and it is exactly one only if the portfolio has no short positions.

Comment briefly on your observations about the different portfolios and the difference between their training and testing performance.

(c) It is well known that optimal portfolios constructed using the Markowitz procedure perform much more poorly out of sample compared to in sample. This is due to a variety of reasons, one of which is that the procedure assumes that future returns are equal to past returns, another that the correlation structure of the market might change over time, and finally, when there are many assets there is the potential for overfitting. Repeat the previous problem but introduce a ridge regression/l_2 norm penalty term to the objective function, with a hyperparameter λ governing the size of the penalty term.

Specifically, you will select 10 positive values of λ on a log scale between $1e-1$ and 10 and for each value of λ solve the following penalized regression problem:

$$\min_w w^T(\Gamma + \lambda I)w,$$
$$w^T \mu = r,$$
$$\sum_{i=1}^{n} w_i = 1$$

for just the single value of $r = 20\%$. Then calculate the performance of each of these regularized Markowitz strategies on both the training and test datasets and plot the the return of the portfolios over both the training and testing period.

For each portfolio, also report:

- The annualized return on the training and test sets;

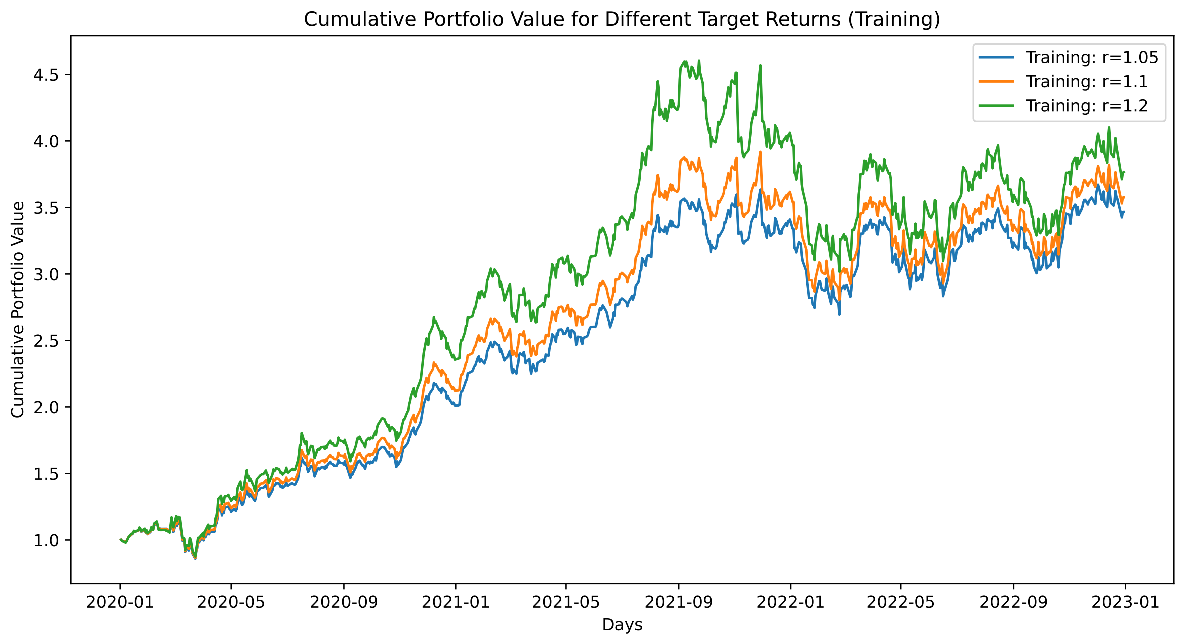
- The risk on the training and test sets;
- The maximum allocation weight and the asset with maximum allocation;
- The initial leverage

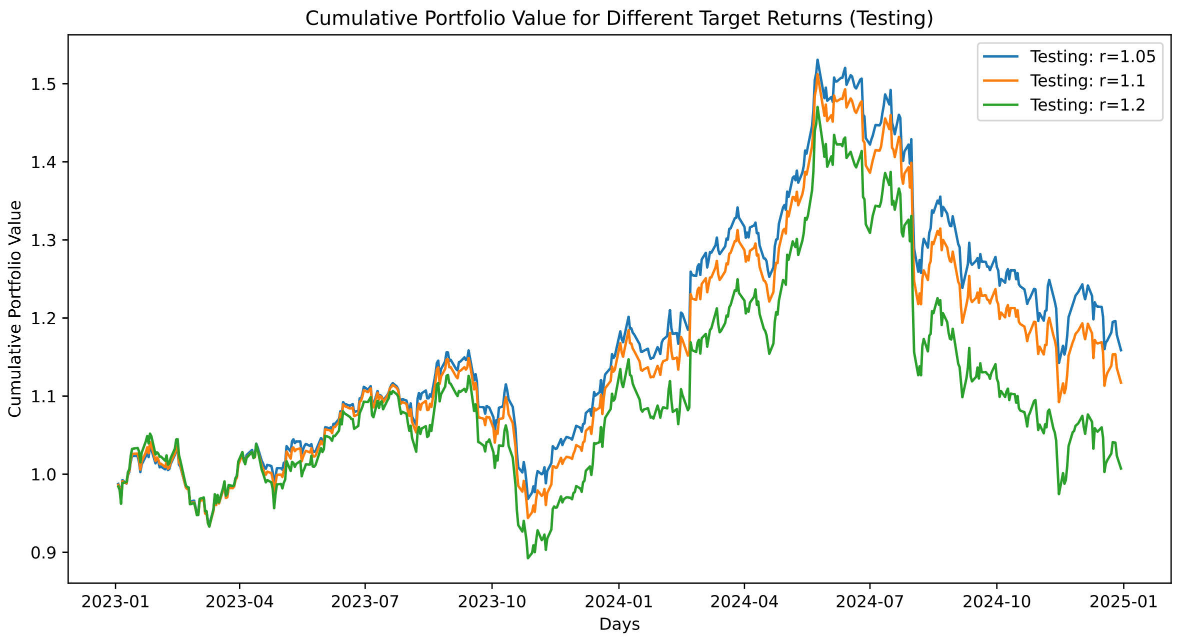
Comment on how the different values of λ changed the optimal portfolios and the difference between in-sample and out-of-sample return and variance.

Problem 3 Solution

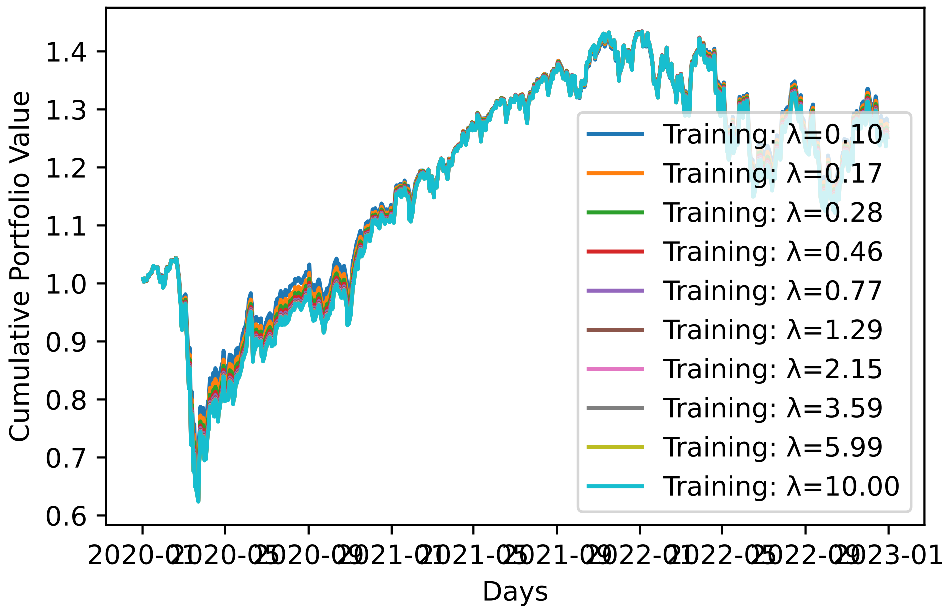
```python
# load dependencies
import numpy as np
import pandas as pd
import cvxpy as cp
import matplotlib.pyplot as plt

## Problem 3(a) Solution

# read in data
stock_url = 'https://media.githubusercontent.com/media/georgehagstrom/DATA609Spring2025/refs/
stock_return_data = pd.read_csv(stock_url)
stock_return_data['date'] = pd.to_datetime(stock_return_data['date'])

# filter data into training (2020-2022) and testing (2022-2024) periods
train_period = stock_return_data[(stock_return_data['date'] >= '2020-01-01') & (stock_return_
test_period = stock_return_data[(stock_return_data['date'] > '2022-12-31')]

# convert the data with the log return
train_data = train_period.pivot(index='date', columns='Company', values='log_return')
test_data = test_period.pivot(index='date', columns='Company', values='log_return')

# calculate the annual mean and covariance matrix
mu = np.exp(0.5 * train_data.sum()) - 1
gamma = train_data.cov().values

## Problem 3(b) Solution

# define variables
r_targets = [1.05, 1.10, 1.20]
optimal_weights = {}
portfolio_cum_returns_train = {}
portfolio_cum_returns_test = {}
```

```python
for r in r_targets:
    # define the variable
    w = cp.Variable(len(mu))

    # define the objective function: 0.5 * w^T * Gamma * w
    objective = cp.Minimize(0.5 * cp.quad_form(w, gamma))

    # define the constraints
    constraints = [cp.sum(w) == 1, w.T @ mu == r, w >= 0]

    # Solve the problem
    problem = cp.Problem(objective, constraints)
    problem.solve()

    # store the optimal weight
    optimal_weights[r] = w.value

    # Calculate portfolio returns for training period
    portfolio_returns_train = train_data.dot(w.value)
    portfolio_cum_returns_train[r] = np.cumprod(1 + portfolio_returns_train)

    # Calculate portfolio returns for testing period
    portfolio_returns_test = test_data.dot(w.value)
    portfolio_cum_returns_test[r] = np.cumprod(1 + portfolio_returns_test)

# Plot cumulative returns for training and testing periods
plt.figure(figsize=(12, 6))

for r in r_targets:
    plt.plot(portfolio_cum_returns_train[r], label=f'Training: r={r}')

plt.title('Cumulative Portfolio Value for Different Target Returns (Training)')
plt.legend()
plt.xlabel('Days')
plt.ylabel('Cumulative Portfolio Value')
plt.show()

plt.figure(figsize=(12, 6))

for r in r_targets:
    plt.plot(portfolio_cum_returns_test[r], label=f'Testing: r={r}')
plt.title('Cumulative Portfolio Value for Different Target Returns (Testing)')
plt.legend()
plt.xlabel('Days')
plt.ylabel('Cumulative Portfolio Value')
plt.show()

## Problem 3(c) Solution

# set the target return
r = 0.20

# define the lambda range
lambda_values = np.logspace(-1, 1, 10)

# define dicts for storing results
optimal_weights_ridge = {}
portfolio_cum_returns_train_ridge = {}
portfolio_cum_returns_test_ridge = {}


for lam in lambda_values:
    # define the variable
    w = cp.Variable(len(mu))

    # define the objective function: 0.5 * w^T * (Gamma + lambda * I) * w
    objective = cp.Minimize(0.5 * cp.quad_form(w, cp.psd_wrap(gamma + lam * np.eye(len(mu)))

    # define the constraints
    constraints = [cp.sum(w) == 1, w.T @ mu == r, w >= 0]

    # define the optimization problem
    problem = cp.Problem(objective, constraints)

    # Solve the problem
    problem.solve()

    # store the optimal weights
    optimal_weights_ridge[lam] = w.value

    # calculate portfolio returns for training data
    portfolio_returns_train = train_data.dot(w.value)
    portfolio_cum_returns_train_ridge[lam] = np.cumprod(1 + portfolio_returns_train)
    # calculate portfolio returns for testing data
    portfolio_returns_test = test_data.dot(w.value)
    portfolio_cum_returns_test_ridge[lam] = np.cumprod(1 + portfolio_returns_test)

# plot the lamda training period
for lam in lambda_values:
    plt.plot(portfolio_cum_returns_train_ridge[lam], label=f'Training:  ={lam:.2f}')

#plt.title('Cumulative Portfolio Value with Ridge Regularization (  values - Training)')
plt.legend()
plt.xlabel('Days')
plt.ylabel('Cumulative Portfolio Value')
plt.show()

# plot the lamda testing period
for lam in lambda_values:
    plt.plot(portfolio_cum_returns_test_ridge[lam], label=f'Testing:  ={lam:.2f}')

plt.title('Cumulative Portfolio Value with Ridge Regularization (  values - Testing)')
plt.legend()
plt.xlabel('Days')
plt.ylabel('Cumulative Portfolio Value')
plt.show()
```

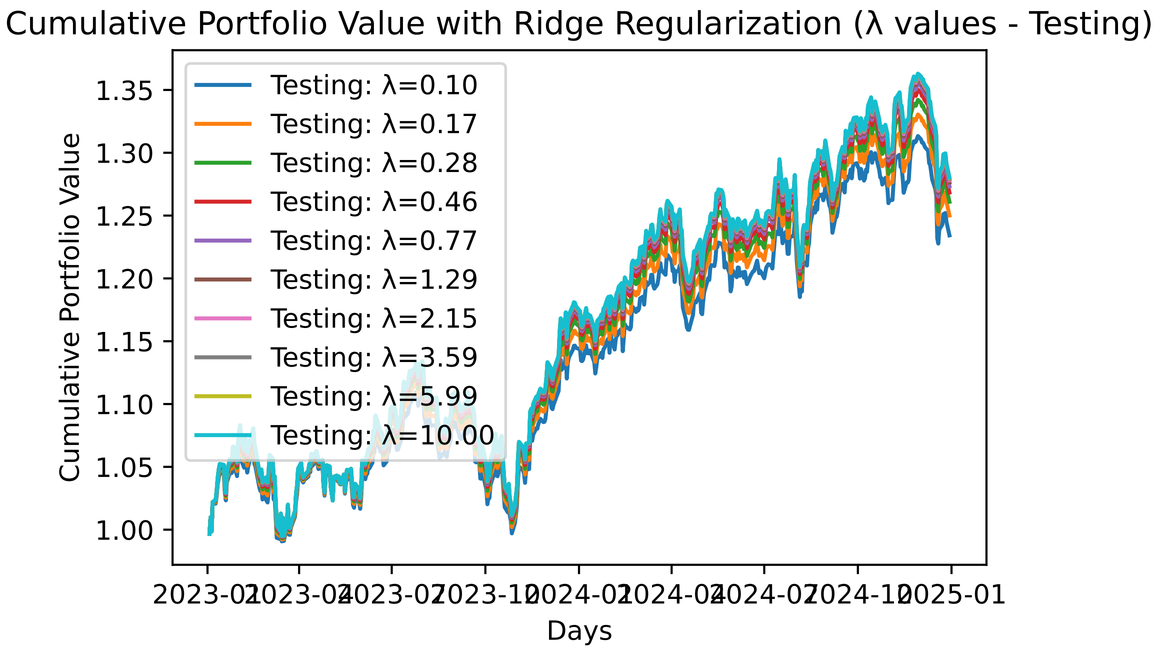




Cumulative Portfolio Value for Different Target Returns (Testing)
Testing: r=1.05
Testing: r=1.1
Testing: r=1.2
Cumulative Portfolio Value
Days
1.5
1.4
1.3
1.2
1.1
1.0
0.9
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
2024-07
2024-10
2025-01



Training: λ=0.10
Training: λ=0.17
Training: λ=0.28
Training: λ=0.46
Training: λ=0.77
Training: λ=1.29
Training: λ=2.15
Training: λ=3.59
Training: λ=5.99
Training: λ=10.00
Cumulative Portfolio Value
Days
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
2020-01
2020-05
2020-09
2021-01
2021-05
2021-09
2022-01
2022-05
2022-09
2023-01



Potential Projects Based on this Homework:

Recursive least squares is a gateway to several important techniques that would lead to good projects. The Kalman Filter or Bayes Filter is an algorithm that recursively estimates a statistical model while allowing the underlying coefficients (or state) of that model to undergo dynamical evolution (as a simple example, think of correcting noisy measurements of the GPS location of a drone using Newtownian physics). These are some of the most useful prediction algorithms and can be applied to many areas, including econometrics, web traffic prediction, and vehicle location. For a more math related project, studying the accuracy of the Woodbury formula could be interesting.