FA Track Project 1 Write Up

Team Members:

Kolton Fowler
Mario Gonzalez
Brandt Green
Adam Hoard

Objective

Our company is facing a stream of liabilities that need to be paid using cash reserves today. To solve this problem, we have decided to purchase a portfolio of bonds and forwards to hedge against interest rate risks.

Main Assumptions

- Assume the following column names in the bond data: "StartTime", "Maturity", "Coupon", "Price".
- The maturity date of the longest tenured bond is greater than or equal to the furthest liability year.

Procedure

The first step of this problem is to upload the data to python.

```
# Read data
data = pd.read_csv('bonds.csv')
liability_df = pd.read_csv('liabilities.csv')
```

Now we want to make several adjustments to our bond data frame so that our data is easier to analyze:

- Add an indicator column called 'forward' which lets us know if the bond is a forward or not
- Create new columns on our data which correspond to the yearly expected cash flows of each bond. The price paid for the bond is represented as a negative cash flow.

Cleaning and Creating our Bond Dataframe

- Creating Bonds Projected Cash Flows

```python
def create_bond_df(data:pd.DataFrame):
    bond_df = data.copy()

    # Add indicator column for forward
    bond_df['forward'] = bond_df['StartTime'].apply(lambda x: 1 if x > 0 else 0)

    longest_maturity = bond_df['Maturity'].max()
    cf_col_names = [f'CF_{x}' for x in list(range(longest_maturity+1))]

    cfs = bond_df.apply(create_cf_vector,axis=1, args=(longest_maturity,)) # Get the cash flow series
    cf_matrix = np.array([np.array(row) for row in cfs]) # convert it to a matrix
    df = pd.DataFrame(columns=cf_col_names,data=cf_matrix) # Put it into a dataframe
    bond_df = bond_df.merge(df,left_index=True, right_index=True) # Merge this data frame onto the existing bond df

    return bond_df

bond_df = create_bond_df(data)
```

- Creating the Bond Dataframe with the New Cash Flow Values

```python
: create_cf_vector(row, longest_maturity: int):
"""This function is used to create a bonds projected cash flows based on its coupon, start time, end time, and price.
"""
start_year = row['StartTime']
maturity_year = row['Maturity']
coupon = row['Coupon']
bond_price = row['Price']

cf_vector = np.zeros(longest_maturity + 1) # vector of zeros to represent cash flows of this bond to be filled in below
for year in range(len(cf_vector)):
    if year == start_year:
        cf_vector[year] = -bond_price
    elif year == maturity_year:
        cf_vector[year] = coupon + 100
    elif year > start_year and year < maturity_year:
        cf_vector[year] = coupon
    else:
        cf_vector[year] = 0
return cf_vector
```

Bond Dataframe

- Now we can examine in further detail the characteristics of our available bonds
- The price represents the market value of the bond and the cash flows are displayed to the right

Bond	Price	Coupon	StartTime	Maturity	forward	CF_0	CF_1	CF_2	CF_3	CF_4	CF_5	CF_6	CF_7	CF_8
1	102	5.0	0	1	0	-102.0	105.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	100	3.0	1	2	1	0.0	-100.0	103.0	0.0	0.0	0.0	0.0	0.0	0.0
3	99	3.5	0	2	0	-99.0	3.5	103.5	0.0	0.0	0.0	0.0	0.0	0.0
4	101	4.0	0	2	0	-101.0	4.0	104.0	0.0	0.0	0.0	0.0	0.0	0.0
5	98	2.5	0	3	0	-98.0	2.5	2.5	102.5	0.0	0.0	0.0	0.0	0.0
6	98	4.0	0	4	0	-98.0	4.0	4.0	4.0	104.0	0.0	0.0	0.0	0.0
7	98	2.0	2	4	1	0.0	0.0	-98.0	2.0	102.0	0.0	0.0	0.0	0.0
8	104	9.0	0	5	0	-104.0	9.0	9.0	9.0	9.0	109.0	0.0	0.0	0.0
9	100	6.0	0	5	0	-100.0	6.0	6.0	6.0	6.0	106.0	0.0	0.0	0.0
10	101	8.0	0	6	0	-101.0	8.0	8.0	8.0	8.0	8.0	108.0	0.0	0.0
11	102	9.0	0	7	0	-102.0	9.0	9.0	9.0	9.0	9.0	9.0	109.0	0.0
12	94	7.0	0	8	0	-94.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	107.0
13	91	3.0	3	8	1	0.0	0.0	0.0	-91.0	3.0	3.0	3.0	3.0	103.0

Liability Dataframe

- And the liabilities we are seeking to replicate:

Year	Liability
1	1200000
2	1800000
3	2000000
4	2000000
5	1600000
6	1500000
7	1200000
8	1000000

Optimization Problem

The problem we would like to solve is how many of each type of bond do we need to purchase so that our cash inflows exactly offset our cash outflows(liabilities).

We want to solve the equation:

$$Ax=b$$

Where x equals the number of each bond to purchase, b equals our liabilities, and A is a matrix where each row represents a bond and each column is the cash flow of a bond for that year. We would also like to solve this problem while paying the lowest possible amount today.

The function below is used to solve this optimization problem, with the results displayed afterward.

```python
def get_optimal_results(bond_df:pd.DataFrame, liability_df:pd.DataFrame):
    """This function is used to find the optimal portfolio. Send in a bond dataframe and liability dataframe.
    This function will then use garobi to optimize and it will return two values:
    the total price paid to create the portfolio today and a dataframe which is exactly
    like the inputted bond_df except there is an additional column 'optimal_quantity' containing the number
    of that bond which is to be purchased.
    """
    cf_col_names = [f'CF_{x}' for x in list(range(bond_df['Maturity'].max()+1))] # Get the names of cash flow columns

    # Get the A matrix, which is all cash flows excluding the year 0 cash flow and anything beyond the liabilities
    bond_df_transposed = bond_df[cf_col_names].iloc[:,1:len(liability_df)+1].T

    # Now do the fun garobi stuff

    # This vector represents our cash outflow today. We want like to maximize this number because this means paying
    # the least possible amount of cash
    obj = bond_df['CF_0']

    # all constraints are '=' to and will be equal to the number of years in our liabilities df.
    sense = np.array(['='] * len(liability_df))

    bondMod = gp.Model() # initialize an empty model
    bondModX = bondMod.addMVar(len(bond_df)) # Number of variables we have is equal to the number of bonds we have available

    # must define the variables before adding constraints because variables go into the constraints
    # add the constraints to the model
    bondModCon = bondMod.addMConstr(bond_df_transposed, bondModX, sense, liability_df['Liability'])

    # Again, we want to maximize this number because our cash outflows are going to be negative at time 0.
    bondMod.setMObjective(None,obj,0,sense=gp.GRB.MAXIMIZE)
    bondMod.Params.OutputFlag = 0 # tell gurobi to shut up!!

    bondMod.optimize() # solve the LP

    price_paid_today = bondMod.ObjVal # Cash paid today
    bond_df['optimal_quantity'] = bondModX.x

    return price_paid_today, bond_df
```

Finding Price of the Bond Today & Bond Portfolio

```python
price_paid_today, bond_df_optimal = get_optimal_results(bond_df, liability_df)
```

```
Restricted license - for non-production use only - expires 2022-01-13
```

```python
print(f'The cash outflow today to create the replicating portfolio is: ${price_paid_today:,.2f}')
```

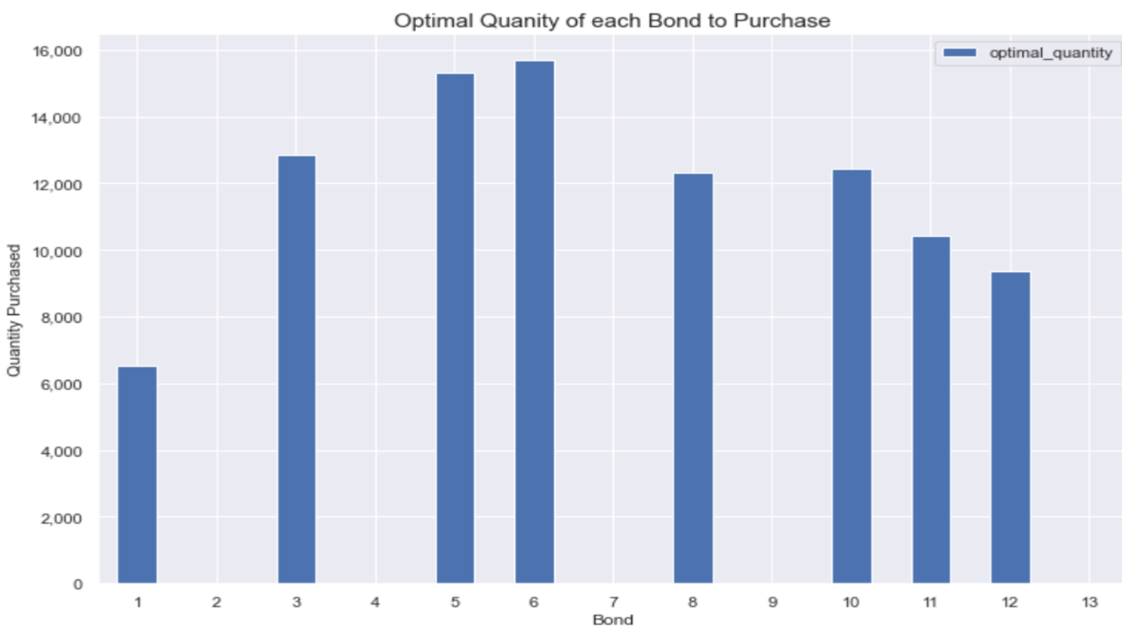
```
The cash outflow today to create the replicating portfolio is: $-9,447,500.76
```

This portfolio will cost us $9,447,500.76 to create today.

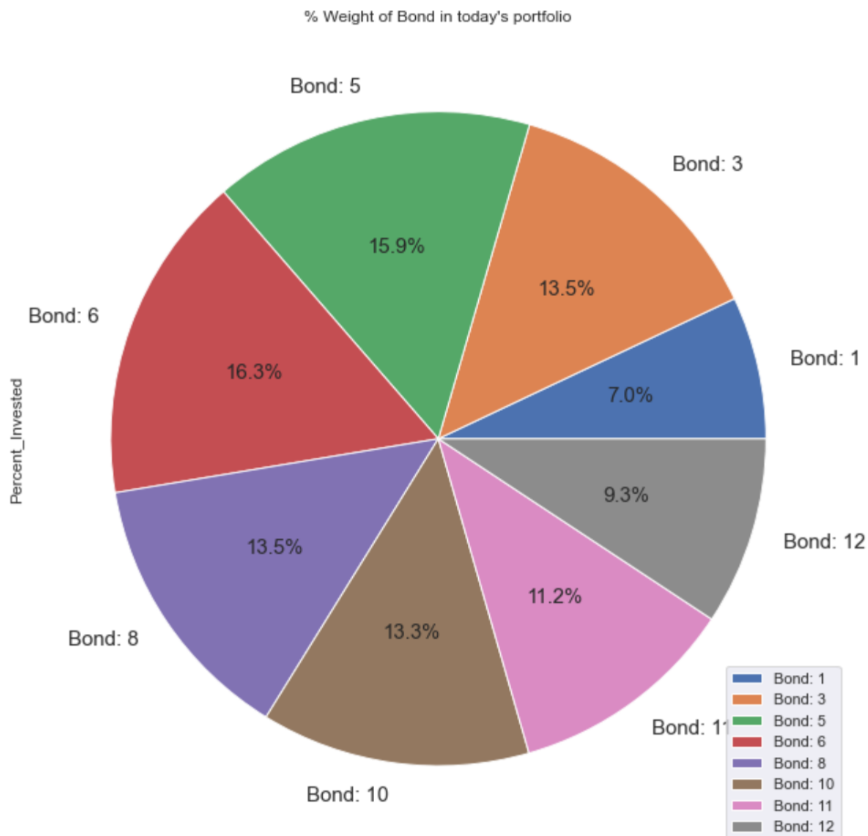
Below is the Optimal Quantity for Each Bond

Bond	Price	Coupon	StartTime	Maturity	optimal_quantity
1	102	5.0	0	1	6522.491727
2	100	3.0	1	2	0.000000
3	99	3.5	0	2	12848.616313
4	101	4.0	0	2	0.000000
5	98	2.5	0	3	15298.317884
6	98	4.0	0	4	15680.775832
7	98	2.0	2	4	0.000000
8	104	9.0	0	5	12308.006865
9	100	6.0	0	5	0.000000
10	101	8.0	0	6	12415.727483
11	102	9.0	0	7	10408.985681
12	94	7.0	0	8	9345.794393
13	91	3.0	3	8	0.000000

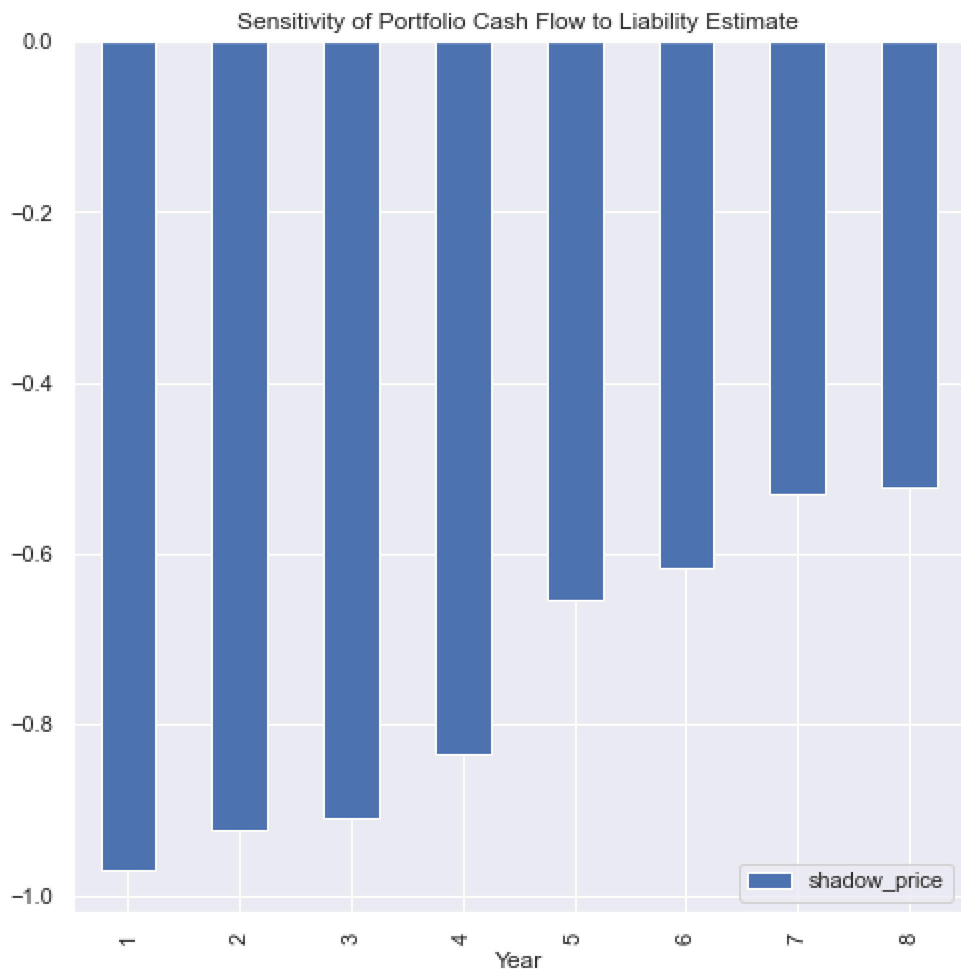
The Bond Allocations Displayed



Pie Chart Displaying Portfolio Allocation Percentages


% Weight of Bond in today's portfolio
Percent_Invested
Bond: 1
Bond: 3
Bond: 5
Bond: 6
Bond: 8
Bond: 10
Bond: 11
Bond: 12
7.0%
13.5%
15.9%
16.3%
13.5%
13.3%
11.2%
9.3%
Bond: 1
Bond: 3
Bond: 5
Bond: 6
Bond: 8
Bond: 10
Bond: 11
Bond: 12

Potential Liability Uncertainty:
We recognize that these expected liabilities are uncertain, so we would like to understand how the cost of our portfolio changes if our liability assumptions change. To carry out this analysis, we calculate the 'Shadow Prices' of each liability which tells us exactly what we want. The calculated shadow prices are displayed graphically below:



We can see that as our liabilities increase, our initial cash outflow becomes even more negative, with the closest liabilities having a stronger impact than those that are further away. With this knowledge in hand, we may take special care to ensure our analysis of the shorter term liabilities is accurate.

Replicating with Real Bond Price from the Wall Street Journal

- The prices were downloaded from the Wall Street Journal on 10/6/21
- The csv file has three columns: "maturity_date","Coupon", and "Price"

```python
# Read data
data_wsj = pd.read_csv('wsj_data.csv')
```

- There are several adjustments we need to make to the Wall Street Journal Datal

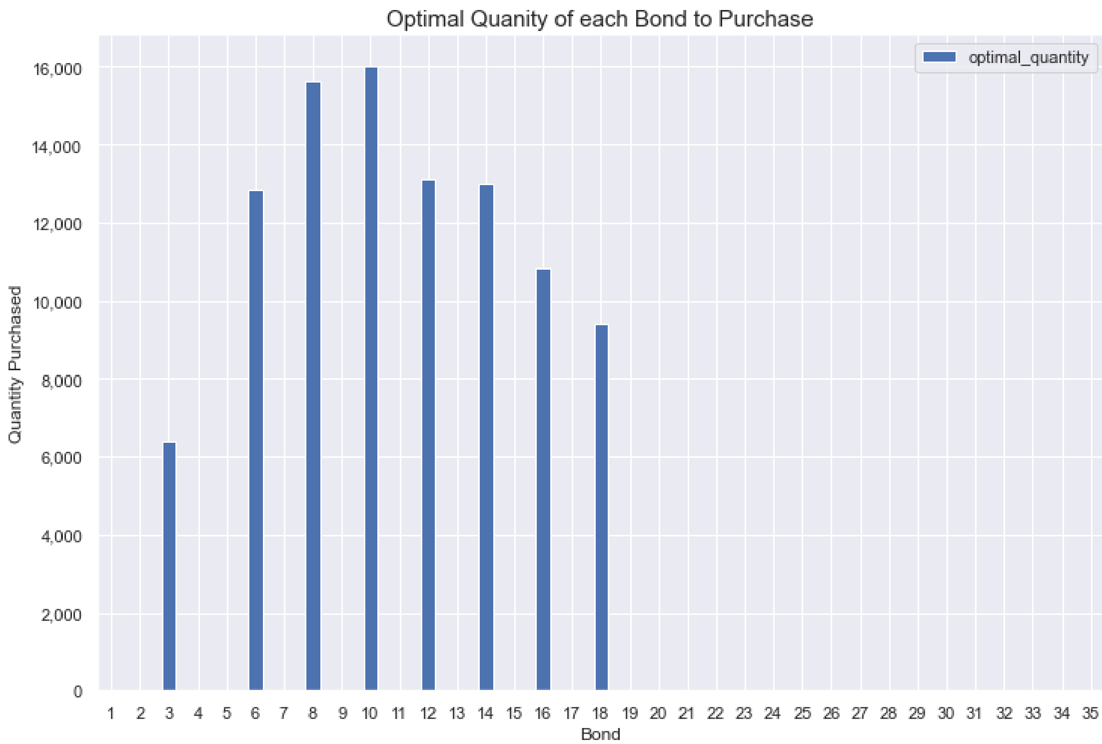
```python
# Make sure wsj maturity column is in datetime format
data_wsj_cleaned = data_wsj.copy()
data_wsj_cleaned['maturity_date'] = pd.to_datetime(data_wsj_cleaned['maturity_date'])

# Filter out bonds with maturity dates that do not end on 8/15
data_wsj_cleaned = data_wsj_cleaned[(data_wsj_cleaned['maturity_date'].dt.month == liability_month) & (data_wsj_cleaned['

# Create a "Maturity" column like in our old bond_df
data_wsj_cleaned['Maturity'] = data_wsj_cleaned['maturity_date'].dt.year - portfolio_creation_date.year
data_wsj_cleaned.drop(columns=['maturity_date'], inplace=True)
data_wsj_cleaned['StartTime'] = 0
data_wsj_cleaned['Bond'] = range(1,len(data_wsj_cleaned) + 1)
data_wsj_cleaned.index = range(0,len(data_wsj_cleaned))
```

Wall Street Journal Bond Dataframe

	Coupon	Price	Maturity	StartTime	Bond	forward	CF_0	CF_1	CF_2	CF_3	...	CF_21	CF_22	CF_23	CF_24	CF_25	CF_26	CF_27	CF_2
0	1.500	101.066	1	0	1	0	-101.066	101.500	0.000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1	1.625	101.104	1	0	2	0	-101.104	101.625	0.000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	7.250	106.032	1	0	3	0	-106.032	107.250	0.000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.125	99.242	2	0	4	0	-99.242	0.125	100.125	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	2.500	104.050	2	0	5	0	-104.050	2.500	102.500	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

5 rows × 37 columns

- The Price, Maturity and Coupon for the first 5 bonds can be seen above
- After filtering out the bonds that do not end on August 15th for a given year, there are a total of 35 bonds available to consider for the optimization.
- With the data cleaned, we simply use the function previously created for the original problem to find the optimal bond amounts for the WSJ bonds
- **The cash outflow needed to create the replicating portfolio is $-11,717,495.64**
- The number of each bond that is purchased is displayed below.



Optimal Quanity of each Bond to Purchase
optimal_quantity
Quantity Purchased
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Bond

Optimal Quantity for Each Bond

	Bond	Price	Coupon	StartTime	Maturity	optimal_quantity
0	1	101.066	1.500	0	1	0.000000
1	2	101.104	1.625	0	1	0.000000
2	3	106.032	7.250	0	1	6376.455787
3	4	99.242	0.125	0	2	0.000000
4	5	104.050	2.500	0	2	0.000000
5	6	111.036	6.250	0	2	12838.748831
6	7	99.204	0.375	0	3	0.000000
7	8	105.106	2.375	0	3	15641.170633
8	9	104.240	2.000	0	4	0.000000
9	10	123.110	6.875	0	4	16012.648436
10	11	102.176	1.500	0	5	0.000000
11	12	127.162	6.750	0	5	13113.518016
12	13	106.110	2.250	0	6	0.000000
13	14	129.284	6.375	0	6	12998.680482
14	15	110.164	2.875	0	7	0.000000
15	16	127.262	5.500	0	7	10827.346362
16	17	101.280	1.625	0	8	0.000000
17	18	135.262	6.125	0	8	9422.850412
18	19	93.016	0.625	0	9	0.000000
19	20	97.166	1.250	0	10	0.000000
20	21	140.144	4.500	0	18	0.000000
21	22	86.032	1.125	0	19	0.000000
22	23	131.046	3.875	0	19	0.000000
23	24	95.194	1.750	0	20	0.000000
24	25	129.222	3.750	0	20	0.000000
25	26	112.202	2.750	0	21	0.000000
26	27	128.142	3.625	0	22	0.000000
27	28	119.236	3.125	0	23	0.000000
28	29	115.132	2.875	0	24	0.000000
29	30	103.132	2.250	0	25	0.000000
30	31	113.196	2.750	0	26	0.000000
31	32	119.086	3.000	0	27	0.000000
32	33	103.242	2.250	0	28	0.000000
33	34	84.164	1.375	0	29	0.000000
34	35	98.114	2.000	0	30	0.000000

- Below is a graph containing the same information

- Only 8 of the bonds are used to replicate the tracking portfolio, similar to the original problem, where many of the bonds were not used