bond_project

Charles Dotson

1 Case Study

Jack Dean, Charles Dotson, and Rodrigo Petricioli
ISE 447 - Financial Optimization
Lehigh Univerisity

[64]:
```
'''
Package Imports
'''
import pandas as pd
import numpy as np
import pulp
from pulp import *
import datetime
import matplotlib.pyplot as plt

import FinOpsCodeDeck as finops

from IPython.display import Markdown as md
# %matplotlib widget
```

1.1 Provided Data

A municipality sends the following liability stream (in millions of dollars)

Date Due	Amount
12/15/2022	11
6/15/2023	9
12/15/2023	8
6/15/2024	7
12/15/2024	9
6/15/2025	10
12/15/2025	9
6/15/2026	12
12/15/2026	9
6/15/2027	6
12/15/2027	5
6/15/2028	7

Date Due	Amount
12/15/2028	9
6/15/2029	7
12/15/2029	8
6/15/2030	7

1.2 Problem 1

Determine the current term structure of treasury rates (see textbook Section 3.4 or other resources that you can find), and find the present value, duration, and convexity of the stream of liabilities. Please explain the main steps followed in your calculations. Use real world data.

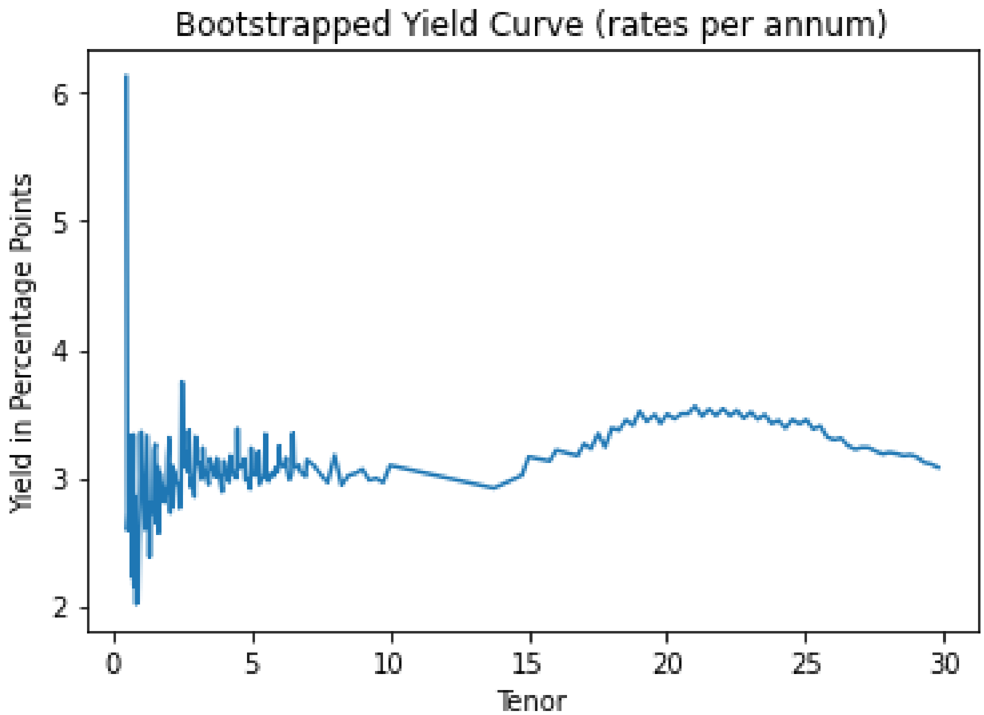
[65]:
```
'''
Data Import for Current Term Structure
---
Imports all active treasury bonds data, time indexes them by year
'''
data_prompt = pd.read_excel('Table.xlsx', sheet_name='PromptUse', index_col =␣
    ↪'DateDue')
data_prompt = data_prompt/1000
term_structure_df = pd.read_excel('TableNew.xlsx', sheet_name='d')
term_structure_df['px_last'] = [i if i>=5 else 100 - i for i in␣
    ↪term_structure_df['ASKED'].to_list()]
term_structure_df['ttm'] = [(i - datetime.datetime.now())/datetime.
    ↪timedelta(days=365) for i in term_structure_df['MATURITY']]
```

[66]:
```
'''
Bootstrap yield curve
---
begins with zero-coupon bonds to payout (ttm < 0.5 yrs) & calculates yield
moves to coupon bonds and uses calculated yields to bootstrap further
sorts all bonds into data frame indexed by ttm (by 100th of a year)
NOTE: Averages yields for the same time period
NOTE: assumes yield of period prior if yield for desired period does not exist
'''

'''short term rates'''
mats = []
round_to = 2
for bond_tenor in term_structure_df[term_structure_df['ttm'] <= 0.5].index:
    bond = term_structure_df.loc[bond_tenor]
    cpn = bond['COUPON']/2
    ttm = bond['ttm']
    px = bond['px_last']
    mats.append([np.round(ttm,round_to),np.log((100 + cpn) / bond['px_last']) /␣
    ↪bond['ttm']])
```

```
rates = pd.DataFrame(mats, columns=['ttm','rate']).set_index('ttm').
 ↪groupby('ttm').mean()

'''longer term rates'''
for bond_tenor in term_structure_df[term_structure_df['ttm']>=0.5].index:
    bond = term_structure_df.loc[bond_tenor]
    px = bond['px_last']
    ttm = bond['ttm']
    cpn = bond['COUPON']/2
    pmts = int(np.ceil(ttm * 2))
    cfs = [cpn if i+1<pmts else 100 + cpn for i in range(pmts)]
    cfs_idx = [np.round(ttm-i*0.5, round_to) for i in reversed(range(pmts))]
    known_rates = [rates[:cfs_idx[i]].iloc[-1,0] for i in range(pmts-1)]
    val = px - sum([cpn * np.exp((-1) * known_rates[i] * cfs_idx[i]) for i in␣
 ↪range(pmts-1)])
    yld = (-1) * (np.log(val / (100+cpn)) / cfs_idx[pmts-1])
    add_df = pd.DataFrame([np.round(ttm, round_to), yld], index=['ttm','rate']).
 ↪transpose().set_index('ttm')
    rates = pd.concat([rates,add_df],ignore_index=False)
    rates = rates.groupby('ttm').mean()
```

[67]:
```
'''
plot yield curve
---
plots yield curve in percentage points
'''
plt.plot(rates[0.5:] * 100)
plt.title('Bootstrapped Yield Curve (rates per annum)')
plt.xlabel('Tenor')
plt.ylabel('Yield in Percentage Points')
```

[67]: Text(0, 0.5, 'Yield in Percentage Points')



[68]:
```
# data_prompt = pd.read_excel('Table.xlsx', sheet_name='PromptUse', index_col =
 ↪'DateDue')
data_prompt['ttm'] = np.round((data_prompt.index - datetime.datetime.now()) /
 ↪datetime.timedelta(days=365), round_to)

r = []
for ttm in data_prompt['ttm']:
    r.append(rates[:ttm].iloc[-1,0])
data_prompt['rates'] = r

npv = sum([data_prompt.iloc[i,0]*np.exp((-1) * data_prompt.iloc[i,1] *
 ↪data_prompt.iloc[i,2]) for i in range(len(data_prompt))])
dur = sum([data_prompt.iloc[i,1]*data_prompt.iloc[i,0]*np.exp((-1) *
 ↪data_prompt.iloc[i,1] * data_prompt.iloc[i,2]) for i in
 ↪range(len(data_prompt))])
con = sum([data_prompt.iloc[i,1]*(data_prompt.iloc[i,1]+1)*data_prompt.
 ↪iloc[i,0]*np.exp((-1) * (data_prompt.iloc[i,1] + 2 ) * data_prompt.
 ↪iloc[i,2]) for i in range(len(data_prompt))])

print('The Net Present Value of the Liabilities is ${:.2f} million dollars'.
 ↪format(npv))
print('The Macauley Duration of the Liability stream is {:.2f} years'.
 ↪format(dur/npv))
```

```
print('The Convexity of the Liability stream is {:.2f}'.format(con/npv))
```

```
The Net Present Value of the Liabilities is $117.50 million dollars
The Macauley Duration of the Liability stream is 3.95 years
The Convexity of the Liability stream is 23.32
```

[69]:
```
data_prompt
```

[69]:

DateDue	Amount	ttm	rates
2022-12-15	11.0	0.59	0.027232
2023-06-15	9.0	1.09	0.028438
2023-12-15	8.0	1.59	0.026448
2024-06-15	7.0	2.09	0.027271
2024-12-15	9.0	2.59	0.030862
2025-06-15	10.0	3.09	0.031190
2025-12-15	9.0	3.59	0.031283
2026-06-15	12.0	4.09	0.031259
2026-12-15	9.0	4.59	0.030760
2027-06-15	6.0	5.09	0.030559
2027-12-15	5.0	5.59	0.030043
2028-06-15	7.0	6.09	0.031192
2028-12-15	9.0	6.59	0.031001
2029-06-15	7.0	7.09	0.031409
2029-12-15	8.0	7.59	0.030270
2030-06-15	7.0	8.09	0.031775

[70]:
```
# Using markdown outputs instead of just printing

md('''
<center>

### Solutions

The Net Present Value of the Liabilities is $\${:.2f}$ MM

The Macauley Duration of the Liability stream is ${:.2f}$ years

The Convexity of the Liability stream is ${:.2f}$


'''.format(npv,dur/npv,con/npv))
```

[70]:

1.2.1 Solutions

The Net Present Value of the Liabilities is \$117.50 MM

The Macauley Duration of the Liability stream is 3.95 years

The Convexity of the Liability stream is 23.32

1.3 Problem 2

Identify *at least* 30 fixed-income assets that are suitable to construct a dedicated bond portfolio for the municipality liabilities that you have been given. Use assets that are considered risk-free; for example, US government non-callable treasury bonds, treasury bills, or treasury notes. Display in an appropriate table the main characteristics of the bonds you choose. Namely, prices, coupon rates, maturity dates, face value).

[71]:

```
'''
code block
'''
ref_data = ['T ' + str(term_structure_df.iloc[bond,1]) + ' ' +␣
↪term_structure_df.iloc[bond,0].strftime('%m/%d/%y') for bond in␣
↪term_structure_df.index]
term_structure_df['ref_data'] = ref_data

bonds_clean = term_structure_df[['ref_data', 'px_last', 'ASKED YIELD']]
bonds_clean.columns = ['Bond', 'Price', 'Yield']
# bonds_clean = bonds_clean.assign(ttm = term_structure_df.ttm.round(2))
bonds_clean = bonds_clean.set_index('Bond')
```

[72]:

```
# You cannot tabe anything within the string literal for the markdown output,␣
↪See the cell titled "BAD MARKDOWN OUTPUT"
# Also when we finally convert to PDF I plan on using the metadata and hiding␣
↪the markdown output cell because it takes up to much space and is ugly
# Because of this, have all output cells in their own cell as done here
md('''
<center>

{}



'''.format(bonds_clean.to_markdown(colalign = ("center",)))
)
```

[72]:

Bond	Price	Yield
T 1.75 05/15/22	100.01	-1.9931
T 2.125 05/15/22	100.012	-2.5447
T 0.0 05/17/22	99.352	0.657
T 0.0 05/19/22	99.34	0.669
T 0.0 05/24/22	99.32	0.69
T 0.0 05/26/22	99.322	0.687
T 0.125 05/31/22	99.312	0.6946
T 1.75 05/31/22	100.014	0.608
T 1.875 05/31/22	100.016	0.544

Bond	Price	Yield
T 0.0 05/31/22	99.307	0.702
T 0.0 06/02/22	99.327	0.682
T 0.0 06/07/22	99.352	0.657
T 0.0 06/09/22	99.347	0.662
T 0.0 06/14/22	99.35	0.659
T 1.75 06/15/22	100.03	0.608
T 0.0 06/16/22	99.372	0.637
T 0.0 06/21/22	99.342	0.667
T 0.0 06/23/22	99.307	0.703
T 0.0 06/28/22	99.275	0.736
T 0.125 06/30/22	99.294	0.755
T 1.75 06/30/22	100.04	0.74
T 2.125 06/30/22	100.052	0.799
T 0.0 06/30/22	99.275	0.736
T 0.0 07/05/22	99.26	0.751
T 0.0 07/07/22	99.235	0.777
T 0.0 07/12/22	99.237	0.774
T 0.0 07/14/22	99.222	0.789
T 1.75 07/15/22	100.052	0.756
T 0.0 07/21/22	99.21	0.802
T 0.0 07/28/22	99.205	0.807
T 0.125 07/31/22	99.27	0.871
T 1.875 07/31/22	100.064	0.902
T 2.0 07/31/22	100.072	0.915
T 0.0 08/04/22	99.085	0.93
T 0.0 08/11/22	99.055	0.96
T 1.5 08/15/22	100.04	0.999
T 1.625 08/15/22	100.05	0.999
T 7.25 08/15/22	101.184	0.942
T 0.0 08/18/22	99.01	1.006
T 0.0 08/25/22	99.01	1.007
T 0.125 08/31/22	99.234	1.042
T 1.625 08/31/22	100.052	1.057
T 1.875 08/31/22	100.076	1.037
T 0.0 09/01/22	98.957	1.06
T 0.0 09/08/22	98.912	1.106
T 1.5 09/15/22	100.03	1.214
T 0.0 09/15/22	98.9	1.12
T 0.0 09/22/22	98.885	1.135
T 0.0 09/29/22	98.887	1.133
T 0.125 09/30/22	99.184	1.258
T 1.75 09/30/22	100.056	1.266
T 1.875 09/30/22	100.074	1.244
T 0.0 10/06/22	98.845	1.176
T 0.0 10/13/22	98.757	1.266
T 1.375 10/15/22	100.004	1.336

Bond	Price	Yield
T 0.0 10/20/22	98.732	1.292
T 0.0 10/27/22	98.697	1.329
T 0.125 10/31/22	99.14	1.365
T 1.875 10/31/22	100.072	1.375
T 2.0 10/31/22	100.09	1.379
T 0.0 11/03/22	98.675	1.352
T 0.0 11/10/22	98.605	1.424
T 1.625 11/15/22	100.022	1.483
T 7.625 11/15/22	103.032	1.346
T 0.125 11/30/22	99.08	1.522
T 2.0 11/30/22	100.084	1.505
T 0.0 12/01/22	98.64	1.389
T 1.625 12/15/22	100.012	1.557
T 0.0 12/29/22	98.515	1.52
T 0.137 12/31/22	99.016	1.653
T 2.125 12/31/22	100.09	1.669
T 1.5 01/15/23	99.274	1.712
T 0.0 01/26/23	98.452	1.586
T 0.125 01/31/23	98.27	1.771
T 1.75 01/31/23	100	1.749
T 2.375 01/31/23	100.142	1.739
T 1.375 02/15/23	99.22	1.795
T 2.0 02/15/23	100.05	1.788
T 7.125 02/15/23	104.006	1.708
T 0.0 02/23/23	98.332	1.713
T 0.125 02/28/23	98.206	1.855
T 1.5 02/28/23	99.232	1.849
T 2.625 02/28/23	100.192	1.853
T 0.5 03/15/23	98.276	1.879
T 0.0 03/23/23	98.282	1.772
T 0.125 03/31/23	98.134	1.954
T 1.5 03/31/23	99.19	1.97
T 2.5 03/31/23	100.146	1.964
T 0.25 04/15/23	98.13	2.016
T 0.0 04/20/23	98.127	1.933
T 0.125 04/30/23	98.056	2.057
T 1.625 04/30/23	99.19	2.056
T 2.75 04/30/23	100.206	2.061
T 0.125 05/15/23	98.03	2.066
T 1.75 05/15/23	99.22	2.068
T 0.125 05/31/23	97.306	2.115
T 1.625 05/31/23	99.156	2.12
T 2.75 05/31/23	100.204	2.124
T 0.25 06/15/23	97.316	2.136
T 0.125 06/30/23	97.23	2.19
T 1.375 06/30/23	99.034	2.181

Bond	Price	Yield
T 2.625 06/30/23	100.156	2.178
T 0.125 07/15/23	97.19	2.227
T 0.125 07/31/23	97.144	2.271
T 1.25 07/31/23	98.262	2.243
T 2.75 07/31/23	100.19	2.248
T 0.125 08/15/23	97.114	2.277
T 2.5 08/15/23	100.094	2.257
T 6.25 08/15/23	104.296	2.23
T 0.125 08/31/23	97.062	2.344
T 1.375 08/31/23	98.26	2.313
T 2.75 08/31/23	100.176	2.31
T 0.125 09/15/23	97.04	2.33
T 0.25 09/30/23	97.056	2.347
T 1.375 09/30/23	98.222	2.345
T 2.875 09/30/23	100.23	2.339
T 0.125 10/15/23	96.276	2.389
T 0.375 10/31/23	97.032	2.412
T 1.625 10/31/23	98.286	2.399
T 2.875 10/31/23	100.21	2.413
T 0.25 11/15/23	96.262	2.425
T 2.75 11/15/23	100.156	2.413
T 0.5 11/30/23	97.01	2.475
T 2.125 11/30/23	99.172	2.431
T 2.875 11/30/23	100.202	2.454
T 0.125 12/15/23	96.126	2.46
T 0.75 12/31/23	97.07	2.508
T 2.25 12/31/23	99.202	2.481
T 2.625 12/31/23	100.08	2.466
T 0.125 01/15/24	96.046	2.5
T 0.875 01/31/24	97.072	2.542
T 2.5 01/31/24	99.302	2.532
T 0.125 02/15/24	95.286	2.534
T 2.75 02/15/24	100.112	2.542
T 1.5 02/29/24	98.056	2.546
T 2.125 02/29/24	99.092	2.533
T 2.375 02/29/24	99.232	2.531
T 0.25 03/15/24	95.286	2.556
T 2.125 03/31/24	99.07	2.554
T 2.25 03/31/24	99.13	2.576
T 0.375 04/15/24	95.286	2.584
T 2.0 04/30/24	98.286	2.581
T 2.25 04/30/24	99.12	2.579
T 2.5 04/30/24	99.27	2.582
T 0.25 05/15/24	95.144	2.601
T 2.5 05/15/24	99.25	2.613
T 2.0 05/31/24	98.264	2.593

Bond	Price	Yield
T 0.25 06/15/24	95.066	2.628
T 1.75 06/30/24	98.06	2.632
T 2.0 06/30/24	98.214	2.647
T 0.375 07/15/24	95.064	2.669
T 1.75 07/31/24	98.014	2.665
T 2.125 07/31/24	98.264	2.674
T 0.375 08/15/24	94.31	2.693
T 2.375 08/15/24	99.092	2.702
T 1.25 08/31/24	96.25	2.708
T 1.875 08/31/24	98.042	2.721
T 0.375 09/15/24	94.224	2.735
T 1.5 09/30/24	97.07	2.717
T 2.125 09/30/24	98.21	2.713
T 0.625 10/15/24	95.034	2.723
T 1.5 10/31/24	97.02	2.745
T 2.25 10/31/24	98.274	2.733
T 0.75 11/15/24	95.054	2.752
T 2.25 11/15/24	98.244	2.765
T 7.5 11/15/24	111.176	2.685
T 1.5 11/30/24	96.292	2.766
T 2.125 11/30/24	98.142	2.763
T 1.0 12/15/24	95.204	2.758
T 1.75 12/31/24	97.134	2.775
T 2.25 12/31/24	98.22	2.771
T 1.125 01/15/25	95.25	2.777
T 1.375 01/31/25	96.114	2.779
T 2.5 01/31/25	99.09	2.776
T 1.5 02/15/25	96.184	2.798
T 2.0 02/15/25	97.296	2.786
T 7.625 02/15/25	112.286	2.725
T 1.125 02/28/25	95.186	2.779
T 2.75 02/28/25	99.286	2.787
T 1.75 03/15/25	97.052	2.798
T 0.5 03/31/25	93.222	2.798
T 2.625 03/31/25	99.176	2.787
T 2.625 04/15/25	99.164	2.799
T 0.375 04/30/25	93.04	2.815
T 2.875 04/30/25	100.066	2.8
T 2.125 05/15/25	98.016	2.806
T 2.75 05/15/25	99.282	2.791
T 0.25 05/31/25	92.186	2.811
T 2.875 05/31/25	100.066	2.802
T 0.25 06/30/25	92.116	2.819
T 2.75 06/30/25	99.266	2.805
T 0.25 07/31/25	92.046	2.826
T 2.875 07/31/25	100.06	2.813

Bond	Price	Yield
T 2.0 08/15/25	97.136	2.833
T 6.875 08/15/25	112.206	2.778
T 0.25 08/31/25	91.29	2.844
T 2.75 08/31/25	99.23	2.839
T 0.25 09/30/25	91.224	2.846
T 3.0 09/30/25	100.166	2.836
T 0.25 10/31/25	91.16	2.85
T 3.0 10/31/25	100.156	2.849
T 2.25 11/15/25	98.004	2.85
T 0.375 11/30/25	91.224	2.855
T 2.875 11/30/25	100.026	2.849
T 0.375 12/31/25	91.166	2.853
T 2.625 12/31/25	99.09	2.835
T 0.375 01/31/26	91.094	2.864
T 2.625 01/31/26	99.062	2.855
T 1.625 02/15/26	95.194	2.867
T 6.0 02/15/26	111.052	2.84
T 0.5 02/28/26	91.18	2.865
T 2.5 02/28/26	98.232	2.856
T 0.75 03/31/26	92.092	2.867
T 2.25 03/31/26	97.24	2.867
T 0.75 04/30/26	92.042	2.868
T 2.375 04/30/26	98.062	2.861
T 1.625 05/15/26	95.094	2.879
T 0.75 05/31/26	91.292	2.884
T 2.125 05/31/26	97.05	2.875
T 0.875 06/30/26	92.074	2.886
T 1.875 06/30/26	96.052	2.868
T 0.625 07/31/26	91.024	2.891
T 1.875 07/31/26	96.024	2.87
T 1.5 08/15/26	94.144	2.896
T 6.75 08/15/26	115.122	2.878
T 0.75 08/31/26	91.134	2.89
T 1.375 08/31/26	93.302	2.885
T 0.875 09/30/26	91.246	2.89
T 1.625 09/30/26	94.262	2.894
T 1.125 10/31/26	92.21	2.893
T 1.625 10/31/26	94.234	2.893
T 2.0 11/15/26	96.07	2.903
T 6.5 11/15/26	115.034	2.893
T 1.25 11/30/26	93.032	2.882
T 1.625 11/30/26	94.222	2.879
T 1.25 12/31/26	92.304	2.889
T 1.75 12/31/26	95.034	2.887
T 1.5 01/31/27	93.292	2.891
T 2.25 02/15/27	97.042	2.9

Bond	Price	Yield
T 6.625 02/15/27	116.114	2.912
T 1.125 02/28/27	92.052	2.888
T 1.875 02/28/27	95.172	2.878
T 0.625 03/31/27	89.244	2.891
T 2.5 03/31/27	98.09	2.88
T 0.5 04/30/27	89.002	2.897
T 2.75 04/30/27	99.126	2.881
T 2.375 05/15/27	97.164	2.913
T 0.5 05/31/27	88.24	2.916
T 0.5 06/30/27	88.184	2.916
T 0.375 07/31/27	87.274	2.903
T 2.25 08/15/27	96.24	2.922
T 6.375 08/15/27	116.224	2.92
T 0.5 08/31/27	88.06	2.926
T 0.5 10/31/27	87.262	2.932
T 2.25 11/15/27	96.186	2.927
T 6.125 11/15/27	116.04	2.928
T 0.625 11/30/27	88.082	2.937
T 0.625 12/31/27	88.02	2.944
T 0.75 01/31/28	88.166	2.949
T 2.75 02/15/28	99.004	2.937
T 1.125 02/29/28	90.136	2.934
T 1.25 03/31/28	90.284	2.95
T 1.25 04/30/28	90.24	2.955
T 2.875 05/15/28	99.192	2.948
T 1.25 05/31/28	90.2	2.956
T 1.25 06/30/28	90.146	2.965
T 1.0 07/31/28	88.296	2.965
T 2.875 08/15/28	99.162	2.961
T 5.5 08/15/28	114.156	2.944
T 1.125 08/31/28	89.164	2.964
T 1.25 09/30/28	90.026	2.968
T 1.375 10/31/28	90.226	2.966
T 3.125 11/15/28	100.312	2.959
T 5.25 11/15/28	113.192	2.935
T 1.5 11/30/28	91.112	2.964
T 1.375 12/31/28	90.194	2.946
T 1.75 01/31/29	92.234	2.951
T 2.625 02/15/29	98.01	2.948
T 5.25 02/15/29	114.03	2.933
T 1.875 02/28/29	93.152	2.942
T 2.375 03/31/29	96.14	2.951
T 2.875 04/30/29	99.172	2.949
T 2.375 05/15/29	96.134	2.945
T 1.625 08/15/29	91.192	2.918
T 6.125 08/15/29	120.284	2.908

Bond	Price	Yield
T 1.75 11/15/29	92.076	2.909
T 1.5 02/15/30	90.076	2.915
T 0.625 05/15/30	83.23	2.923
T 6.25 05/15/30	123.232	2.904
T 0.625 08/15/30	83.09	2.92
T 0.875 11/15/30	84.224	2.921
T 1.125 02/15/31	86.076	2.918
T 5.375 02/15/31	118.256	2.923
T 1.625 05/15/31	89.25	2.925
T 1.25 08/15/31	86.162	2.925
T 1.375 11/15/31	87.046	2.935
T 1.875 02/15/32	91.02	2.936
T 2.875 05/15/32	99.176	2.927
T 4.5 02/15/36	118.09	2.88
T 4.75 02/15/37	121.096	2.957
T 5.0 05/15/37	124.052	2.989
T 4.375 02/15/38	116.29	3.018
T 4.5 05/15/38	118.154	3.033
T 3.5 02/15/39	105.026	3.108
T 4.25 05/15/39	115.006	3.106
T 4.5 08/15/39	118.124	3.114
T 4.375 11/15/39	116.14	3.146
T 4.625 02/15/40	120.014	3.143
T 1.125 05/15/40	70.18	3.309
T 4.375 05/15/40	116.1	3.178
T 1.125 08/15/40	70.03	3.324
T 3.875 08/15/40	108.282	3.227
T 1.375 11/15/40	73.066	3.326
T 4.25 11/15/40	113.294	3.243
T 1.875 02/15/41	79.282	3.325
T 4.75 02/15/41	121.106	3.225
T 2.25 05/15/41	84.262	3.335
T 4.375 05/15/41	115.164	3.272
T 1.75 08/15/41	77.146	3.348
T 3.75 08/15/41	106.144	3.294
T 3.125 11/15/41	97.11	3.311
T 2.0 11/30/41	80.292	3.34
T 2.375 02/15/42	86.136	3.316
T 3.125 02/15/42	97.094	3.312
T 3.0 05/15/42	95.104	3.322
T 2.75 08/15/42	91.074	3.349
T 2.75 11/15/42	91.022	3.356
T 3.125 02/15/43	96.232	3.345
T 2.875 05/15/43	92.264	3.354
T 3.625 08/15/43	104.124	3.335
T 3.75 11/15/43	106.112	3.334

Bond	Price	Yield
T 3.625 02/15/44	104.09	3.346
T 3.375 05/15/44	100.094	3.356
T 3.125 08/15/44	96.11	3.359
T 3.0 11/15/44	94.084	3.366
T 2.5 02/15/45	86.084	3.369
T 3.0 05/15/45	94.094	3.358
T 2.875 08/15/45	92.116	3.35
T 3.0 11/15/45	94.174	3.337
T 2.5 02/15/46	86.076	3.344
T 2.5 05/15/46	86.074	3.338
T 2.25 08/15/46	82.016	3.335
T 2.875 11/15/46	92.234	3.31
T 3.0 02/15/47	94.266	3.307
T 3.0 05/15/47	94.282	3.302
T 2.75 08/15/47	90.276	3.285
T 2.75 11/15/47	90.302	3.277
T 3.0 02/15/48	95.21	3.25
T 3.125 05/15/48	98.04	3.232
T 3.0 08/15/48	95.242	3.241
T 3.375 11/15/48	103.004	3.205
T 3.0 02/15/49	96.142	3.199
T 2.875 05/15/49	94.084	3.194
T 2.25 08/15/49	82.304	3.191
T 2.375 11/15/49	85.126	3.175
T 2.0 02/15/50	78.066	3.189
T 1.25 05/15/50	64.106	3.184
T 1.375 08/15/50	66.176	3.177
T 1.625 11/15/50	71.07	3.166
T 1.875 02/15/51	75.31	3.152
T 2.375 05/15/51	85.156	3.141
T 2.0 08/15/51	78.106	3.137
T 1.875 11/15/51	76.014	3.124
T 2.25 02/15/52	83.134	3.108

1.4 Problem 3

Formulate a linear programming model to find the lowest cost bond dedicated portfolio that covers the stream of liabilities. To eliminate the possibility of any interest risk, assume that a 0% reinvestment rate on cash balances carried out from one date to the next. Assume no short selling of bonds is allowed. What is the cost of your portfolio? How does this cost compares with the NPV of the liabilities? What is the composition of the portfolio?

1.4.1 Mathematical Formulation

$$
\begin{aligned}
\min \quad & z_0 + \sum_{i=1}^{N} P_i x_i \\
\text{s.t.} \quad & \sum_{i=1,\dots,n:M_i \geq t-1} C_i x_i + \sum_{i=1,\dots,n:M_i \geq t} 100 x_i + z_{t-1} - z_t = L_t \\
& \text{All variables are non-negative}
\end{aligned}
$$

[73]:
```python
'''
Data Manipulation
'''
term_by_maturity = term_structure_df.set_index('MATURITY')
possibilities = term_by_maturity.drop(
    index=[i for i in term_by_maturity.index.to_list() if i > data_prompt.index.
 ↪to_list()[-1]],
    columns=['BID', 'ASKED', 'ASKED YIELD']#, 'ttm']
    )

'''List of bond maturities less than liability maturity'''
date_lists_to_change_to_periods = [
    [i for i in possibilities.index.to_list() if i <= t]
    for t in data_prompt.index.tolist()
    ]
'''Removing the duplicates from each one'''
for i in reversed(range(1,len(date_lists_to_change_to_periods))):
    for j in range(0,len(date_lists_to_change_to_periods[i-1])):
        date_lists_to_change_to_periods[i].
 ↪remove(date_lists_to_change_to_periods[i-1][j])

for i in range(0,len(date_lists_to_change_to_periods)):
    possibilities.loc[date_lists_to_change_to_periods[i],'period'] = i+1

possibilities['face'] = 100
possibilities['bond#'] = range(1,len(possibilities)+1)
possibilities = possibilities.set_index('bond#')

'''for labeling later'''
dec_var_names = possibilities['ref_data']
```

[74]:
```python
'''Getting data ready for the solver'''

'''Exmpty Array'''
cfs = np.zeros((len(possibilities),len(date_lists_to_change_to_periods)))

'''CF Matrix'''
'''Will make function later'''
for i in range(0, len(cfs)):
```

```
    for j in range(1, len(cfs[0])+1):
        if possibilities.loc[i+1,'period'] == j and possibilities.
 ↪loc[i+1,'COUPON'] == 0:
            cfs[i][j-1] = possibilities.loc[i+1,'face']
        elif possibilities.loc[i+1,'period'] == j and possibilities.
 ↪loc[i+1,'COUPON'] != 0:
            cfs[i][0:j-1] = possibilities.loc[i+1,'COUPON']/2
            cfs[i][j-1] = possibilities.loc[i+1,'face'] + possibilities.
 ↪loc[i+1,'COUPON']/2

cf_matrix = cfs.tolist()
prices = possibilities['px_last'].values.tolist()
liabilities = data_prompt['Amount'].values.tolist()
```

[75]:
```
'''Solving for the dedicated portfolio'''

# Making variable list of strings

periods = [i for i in range(0,len(cf_matrix[0])+1)]



# Dictionary of period constraints
period_dict = {}
for i in range(0,len(cf_matrix[0])):
    period_dict['Period {}'.format(i+1)] =␣
 ↪dict(zip(dec_var_names,[cf_matrix[j][i] for j in range(0,len(cf_matrix))]))

objective = dict(zip(dec_var_names, prices))

# Decision Vars
quantity = LpVariable.dict('', dec_var_names, lowBound=0)
excess = LpVariable.dict('carryover', periods, lowBound=0)

# Intializing the Problem
dedication_1 = LpProblem('Dedicated', LpMinimize)

# Objective function
dedication_1 += excess[0]+lpSum([objective[i]*quantity[i] for i in␣
 ↪dec_var_names])

# Constraints
for i in range(0,len(cf_matrix[0])):
    dedication_1 += lpSum([period_dict['Period {}'.format(i+1)][j]*quantity[j]␣
 ↪for j in dec_var_names]) + excess[i]- excess[i+1] == liabilities[i]

dedication_1.solve()
```

[75]: 1

[76]:
```
composition = pd.DataFrame(
    [v.varValue for v in dedication_1.variables() if v.name[0] != 'c' and v.
 ↪varValue > 0],
    index=[v.name[1:] for v in dedication_1.variables() if v.name[0] != 'c' and␣
 ↪v.varValue > 0],
    columns=['Quantity']
    )
```

[77]:
```
# Just have to match bonds with the tickers Jack created instead

'''Printing Solutions'''

md('''
##### <center> Dedication Portfolio Cost & Composition  </center>

<center>

Portfolio Cost $ = \${:.2f} $ MM <br>

</center>

<center>

{}

'''.format(dedication_1.objective.value(),composition.to_markdown(colalign =␣
 ↪("right",))))
```

[77]:

#####

Dedication Portfolio Cost & Composition

Portfolio Cost \$ = \$117.77 \$ MM

	Quantity
T_0.625_05_15_30	0.0697819
T_0.75_05_31_26	0.107674
T_1.25_05_31_28	0.0637436
T_1.5_11_30_24	0.0714316
T_2.0_05_31_24	0.0509223
T_2.875_04_30_29	0.0664559
T_5.5_08_15_28	0.084142
T_6.125_08_15_29	0.0774112
T_6.25_08_15_23	0.0590762
T_6.375_08_15_27	0.0423924
T_6.625_02_15_27	0.0507125

	Quantity
T_6.75_08_15_26	0.0780774
T_6.875_08_15_25	0.0750923
T_7.125_02_15_23	0.0667
T_7.25_08_15_22	0.0836671
T_7.625_02_15_25	0.0819673

1.5 Problem 4

Use the linear programming sensitivity analysis information to determine the term structure of interest rates implied by the optimal bond portfolio you found in the previous question. Use a plot to compare these rates with the current term structure of treasury rates you found in the first question.

[78]:
```
'''
Pull sensitivity analysis
---
https://s3.amazonaws.com/assets.datacamp.com/production/course_8835/slides/
↪chapter4.pdf
'''
o = [{'name':name, 'shadow price':c.pi} for name, c in dedication_1.constraints.
↪items()]
shadow_px = pd.DataFrame(o).set_index(data_prompt.index).drop('name',axis=1)
clean_shadow_px = pd.DataFrame(o).set_index(data_prompt.index.strftime('%m/%d/
↪%y')).drop('name',axis=1)
md('''
##### <center> Sensitivity Analysis - Shadow Prices  </center>

<center>

{}

'''.format(clean_shadow_px.to_markdown(colalign = ("right",)))
)
```

[78]: #####

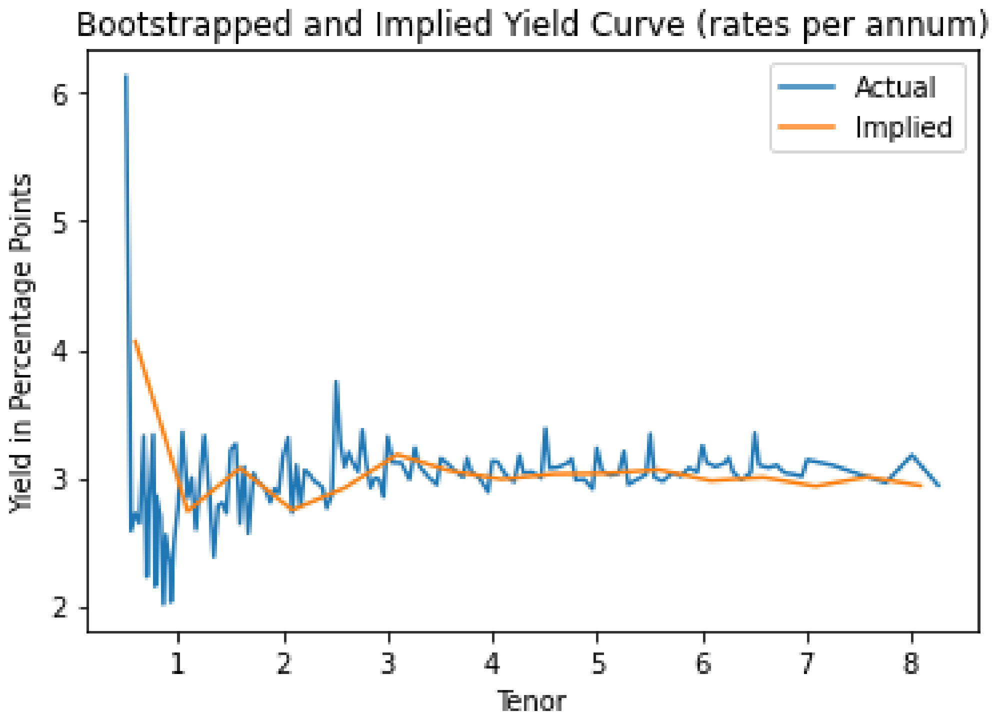
Sensitivity Analysis - Shadow Prices

DateDue	shadow price
12/15/22	0.976444
06/15/23	0.970693
12/15/23	0.952351
06/15/24	0.944203
12/15/24	0.927139
06/15/25	0.906415
12/15/25	0.896101

DateDue	shadow price
06/15/26	0.884951
12/15/26	0.870135
06/15/27	0.856877
12/15/27	0.8426
06/15/28	0.834112
12/15/28	0.820296
06/15/29	0.812113
12/15/29	0.795826
06/15/30	0.788304

[79]:
```
'''
Presents implied and actual yield curve as a plot
'''
shadow_ttm = ((shadow_px.index - datetime.datetime.now()) / datetime.
 ↪timedelta(days=365)).to_list()
shadow_factors = shadow_px['shadow price'].to_list()
implied_rates = [-np.log(shadow_factors[i]) / shadow_ttm[i] for i in␣
 ↪range(len(shadow_ttm))]

implied_rates_df = pd.DataFrame(
                        data = [shadow_ttm, implied_rates],
                        index=['ttm','implied_rate']
                    )

implied_rates_df = (implied_rates_df
                        .transpose()
                        .round({'ttm':round_to})
                        .set_index('ttm')
                    )

plt.plot(rates[0.5:8.5] * 100)
plt.plot(implied_rates_df * 100)
plt.title('Bootstrapped and Implied Yield Curve (rates per annum)')
plt.xlabel('Tenor')
plt.ylabel('Yield in Percentage Points')
plt.legend(['Actual', 'Implied'])
```

[79]: <matplotlib.legend.Legend at 0x246210a0f70>



[80]:
```
'''

'''

md('''
##### <center> Sensitivity Analysis - Shadow Prices  </center>

<center>

{}

'''.format((implied_rates_df*100).to_markdown(colalign = ("right",)))
)
```

[80]:

#####

Sensitivity Analysis - Shadow Prices

ttm	implied_rate
0.59	4.06243
1.09	2.74039
1.59	3.07675
2.09	2.7495
2.59	2.92146

ttm	implied_rate
3.09	3.18176
3.59	3.05616
4.09	2.98967
4.59	3.03095
5.09	3.0357
5.59	3.06399
6.09	2.978
6.59	3.00488
7.09	2.93498
7.59	3.00799
8.09	2.93998

1.6 Problem 5

Formulate a linear programming model to find the lowest cost bond immunized portfolio that matches the present value, duration, and convexity of a stream of liabilities. Assume that no short rates are allowed. What is the cost of your portfolio? How much would you save by using this immunization strategy instead of the dedication one? Is your portfolio immunized against non-parallel shifts in the term structure? Explain why or why not.

[81]:
```
'''
Aggregates cashflow matrix and ref data for immunization
---
Puts cashflow matrix into a dataframe for merging
merges possible bond ref data with cashflow matrix
cleans resulting dataframe

NOTE: MATH NEEDS WORK HERE BUT WE CAN FIGURE OUT
from here: use ttm and col_num against calculated curve to find appropriate␣
↪measure
        pv_factor = exp{-rt} = exp{- () * (ttm)}
'''
cf_df = pd.DataFrame(cf_matrix, index=dec_var_names)

cf_df = pd.merge(                                                            #␣
↪Combines possible bonds with cashflow matrix
    left = possibilities,                                                    #␣
↪possible bonds - SAME DF AS DEDICATION
    right = cf_df,                                                           #␣
↪Cashflow matrix - NP ARRAY FROM DEDICATION AS DF FOR MERGING
    how='inner',                                                             #␣
↪Catches any missed bonds on merge
    left_on='ref_data',                                                      #␣
↪possibilities not indexed by bond name - CHAZ IS THIS SOMETHING WE CAN␣
↪ADJUST OR NO????????????????????
```

```
    right_index=True                                                            #␣
 ↪Casflow df indexed by bond name
)

cf_df = (cf_df
            .drop(['COUPON','period','face'],axis=1)                            #␣
 ↪Drops unnecessary ref data
            .set_index('ref_data')                                              #␣
 ↪Sets index to bond name
            .round({'ttm':round_to})                                            #␣
 ↪rounds time to maturity to 2 decimal places -- allows use of derived term␣
 ↪structure (indexed by hundredths)
        )

cf_df
```

[81]:

```
                  px_last   ttm         0       1       2       3       4  \
ref_data
T 1.75 05/15/22   100.010  0.00  100.8750  0.0000  0.0000  0.0000  0.0000
T 2.125 05/15/22  100.012  0.00  101.0625  0.0000  0.0000  0.0000  0.0000
T 0.0 05/17/22     99.352  0.01  100.0000  0.0000  0.0000  0.0000  0.0000
T 0.0 05/19/22     99.340  0.01  100.0000  0.0000  0.0000  0.0000  0.0000
T 0.0 05/24/22     99.320  0.03  100.0000  0.0000  0.0000  0.0000  0.0000
…                    …    …         …       …       …       …       …
T 6.125 08/15/29  120.284  7.26    3.0625  3.0625  3.0625  3.0625  3.0625
T 1.75 11/15/29    92.076  7.51    0.8750  0.8750  0.8750  0.8750  0.8750
T 1.5 02/15/30     90.076  7.76    0.7500  0.7500  0.7500  0.7500  0.7500
T 0.625 05/15/30   83.230  8.01    0.3125  0.3125  0.3125  0.3125  0.3125
T 6.25 05/15/30   123.232  8.01    3.1250  3.1250  3.1250  3.1250  3.1250

                       5       6       7       8       9      10      11  \
ref_data
T 1.75 05/15/22   0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
T 2.125 05/15/22  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
T 0.0 05/17/22    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
T 0.0 05/19/22    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
T 0.0 05/24/22    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
…                   …       …       …       …       …       …       …
T 6.125 08/15/29  3.0625  3.0625  3.0625  3.0625  3.0625  3.0625  3.0625
T 1.75 11/15/29   0.8750  0.8750  0.8750  0.8750  0.8750  0.8750  0.8750
T 1.5 02/15/30    0.7500  0.7500  0.7500  0.7500  0.7500  0.7500  0.7500
T 0.625 05/15/30  0.3125  0.3125  0.3125  0.3125  0.3125  0.3125  0.3125
T 6.25 05/15/30   3.1250  3.1250  3.1250  3.1250  3.1250  3.1250  3.1250

                      12      13        14        15
ref_data
T 1.75 05/15/22   0.0000  0.0000    0.0000    0.0000
```

```
T 2.125 05/15/22  0.0000  0.0000    0.0000    0.0000
T 0.0 05/17/22    0.0000  0.0000    0.0000    0.0000
T 0.0 05/19/22    0.0000  0.0000    0.0000    0.0000
T 0.0 05/24/22    0.0000  0.0000    0.0000    0.0000
…                    …       …         …         …
T 6.125 08/15/29  3.0625  3.0625  103.0625    0.0000
T 1.75 11/15/29   0.8750  0.8750  100.8750    0.0000
T 1.5 02/15/30    0.7500  0.7500    0.7500  100.7500
T 0.625 05/15/30  0.3125  0.3125    0.3125  100.3125
T 6.25 05/15/30   3.1250  3.1250    3.1250  103.1250

[289 rows x 18 columns]
```

[82]:
```
'''
Create PResent Value, Duration, and Convexity factors for all possible time␣
 ↪index based on derived rates curve
'''
t = rates.index
r = rates['rate']
npv_factor = np.exp(-r*t)
dur_factor = (t / (1+r)) * np.exp(-r*t)
con_factor = ((t*(t+1))/ (1+r)**2) * np.exp(-r*t)
```

[83]:
```
'''
Calculates npv, duration, and convexity terms for all bonds consiuder in problem
'''
npvs=[]
durs=[]
cons=[]
for bond in cf_df.index:
    bond_df = cf_df.loc[bond]
    bond_ttm = bond_df.loc['ttm']
    bond_cf_stream = bond_df.loc[0:]
    eo_cfs = bond_cf_stream.idxmax()
    cpn_ttm = [(bond_ttm - 0.5*i).round(round_to) for i in range(eo_cfs+1)]
    bond_cf_ttm = pd.Series(data=bond_df.loc[0:eo_cfs].to_list(),␣
 ↪index=reversed(cpn_ttm))

    bond_npv = sum([bond_cf_ttm.loc[i] * npv_factor.loc[:i].iloc[-1] for i in␣
 ↪bond_cf_ttm.index])
    bond_dur = sum([bond_cf_ttm.loc[i] * dur_factor.loc[:i].iloc[-1] for i in␣
 ↪bond_cf_ttm.index])
    bond_con = sum([bond_cf_ttm.loc[i] * con_factor.loc[:i].iloc[-1] for i in␣
 ↪bond_cf_ttm.index])

    npvs.append(bond_npv)
    durs.append(bond_dur)
```

```
    cons.append(bond_con)

immunization_df = pd.DataFrame([npvs, durs, cons], columns=cf_df.index,␣
↪index=['npv','duration','convexity']).transpose()
```

[84]:
```
'''
Solves immunization portfolio
'''
bond_count = LpVariable.dicts('Bonds',dec_var_names,lowBound=0)

immunization = LpProblem('immunization', LpMinimize)

immunization += lpSum([cf_df['px_last'].loc[i] * bond_count[i] for i in␣
↪dec_var_names])
immunization += lpSum([immunization_df['npv'].loc[i] * bond_count[i] for i in␣
↪dec_var_names]) == npv
immunization += lpSum([immunization_df['duration'].loc[i] * bond_count[i] for i␣
↪in dec_var_names]) == dur
immunization += lpSum([immunization_df['convexity'].loc[i] * bond_count[i] for␣
↪i in dec_var_names]) == con

immunization.solve()
```

[84]: 1

[85]:
```
'''
Print Solution to Immunized portfolio
'''
bonds_fin = pd.DataFrame([[v.name[6:] for v in immunization.variables()], [v.
↪varValue for v in immunization.variables()]], index=['bond','amt']).
↪transpose().set_index('bond')
bonds_fin[bonds_fin['amt']>0]
```

[85]: #####

Portoflio Allocation - Immunization

Immunized Portfolio Value of $0.12 MM

bond	amt
T_6.125_08_15_29	0.169073
T_6.625_02_15_27	0.728901
T_7.25_08_15_22	0.117564

[ ]:
```
md('''
##### <center> Portoflio Allocation - Immunization  </center>
```

```
<center> Immunized Portfolio Value of ${:.2f} MM  </center> <br>


<center>

{}

'''.format(value(immunization.objective) / 1000, bonds_fin[bonds_fin['amt']>0].
 ↪to_markdown(colalign = ("right",)))
)
```

1.7 Problem 6

Combine a cash matching strategy (dedication) for the liabilities for the first three years and an immunization strategy based on matching present value, duration and convexity for the liabilities during the last five years. Compare the characteristics of the three bond portfolios you have obtained. Explain which one you think is the best one and why.

[86]:
```
'''
Immunization part
Calculates npv, duration, and convexity terms for all bonds considered in the␣
 ↪problem
FROM period 7-16
'''

ded_period = 6
imm_period = 5
imm_start_period = len(data_prompt) - imm_period*2
imm_end_period = len(data_prompt)
```

[87]:
```
'''
Solves combined portfolio
'''
bond_q = LpVariable.dicts('Bond',dec_var_names,lowBound=0)
excess_cf = LpVariable.dicts('ExcessCf', periods[:ded_period+1], lowBound=0)


combined = LpProblem('Combined', LpMinimize)

combined += lpSum([cf_df['px_last'][i] * bond_q[i] for i in dec_var_names]+␣
 ↪excess_cf[0])

for i in range(0,ded_period):
    combined += lpSum([cf_df[i][j]*bond_q[j] for j in dec_var_names]) +␣
 ↪excess_cf[i]- excess_cf[i+1] == liabilities[i]
```

```
combined += lpSum([immunization_df['npv'][i] * bond_q[i] for i in␣
 ↪dec_var_names]) == npv
combined += lpSum([immunization_df['duration'][i] * bond_q[i] for i in␣
 ↪dec_var_names]) == dur
combined += lpSum([immunization_df['convexity'][i] * bond_q[i] for i in␣
 ↪dec_var_names]) == con

combined.solve()
```

[87]: 1

[88]:
```
'''
Print Solution to Combined portfolio
'''
bonds_comb = pd.DataFrame(
             [v.varValue for v in combined.variables() if v.varValue > 0],
             index=[str(v.name[:-8].replace('Bond', '').replace('_', ' ') + v.
 ↪name[-8:].replace('_', '/'))
             if v.name[0] == 'B' else str(r'$\text{Excess}_' + v.name[-1] + '$')
             for v in combined.variables() if v.varValue > 0],
             columns=['Quantity'])

bonds_comb.index.name = 'Bonds/Excess Cashflows'


md('''
##### <center> Portoflio Allocation - Combined  </center>

<center> Combined Portfolio Value of ${:.2f} MM  </center> <br>



<center>

{}

'''.format(value(combined.objective) / 1000, bonds_comb.to_markdown(colalign =␣
 ↪("right",)))
)
```

[88]: #####

Portoflio Allocation - Combined

Combined Portfolio Value of $0.12 MM

bond	amt
_0.125_12_15_23	0.0760791
_0.25_05_31_25	0.0966403

bond	amt
_0.25_06_15_23	0.0859716
_0.25_06_15_24	0.0661266
_0.75_01_31_28	0.678449
_1.0_12_15_24	0.0862093
_1.25_04_30_28	0.111152
_1.625_12_15_22	0.0876304
Cf_0	1.76292

2 Part 7: Dedication Portfolio with Short Selling

The municipality would like to make a second bid (find a different portfolio of bonds). What is your best dedicated portfolio of risk-free bonds you can create *if short sales are allowed*? Did you find arbitrage opportunities? Did you take into consideration the bid-ask spread of the bonds? How would you take them in consideration and what is the result? Did you set limits in the transaction amounts? Discuss the practical feasibility of your solutions.

2.1 Without Transaction Limits

2.1.1 Mathematical Formulation

input here

2.1.2 Code

[89]:
```
'''Data Manipulation'''
data = term_structure_df
Bid_p = data['BID'].to_list()[0:len(cf_matrix)]
Ask_p = data['ASKED'].to_list()[0:len(cf_matrix)]
Bid_p_fix = [Bid_p[i]
             if Bid_p[i] > 10 else 100 - Bid_p[i]
             for i in range(0,len(Bid_p))]
Ask_p_fix = [Ask_p[i]
             if Ask_p[i] > 10 else 100 - Ask_p[i]
             for i in range(0,len(Ask_p))]

'''Solving for the dedicated portfolio'''

# Making variable list of strings
periods = [i for i in range(0,len(cf_matrix[0])+1)]
excess_from_arb_left = [0*i for i in periods]


# Dictionary of period constraints
period_dict = {}
for i in range(0,len(cf_matrix[0])):
```

```
    period_dict['Period {}'.format(i+1)] =␣
 ↪dict(zip(dec_var_names,[cf_matrix[j][i] for j in range(0,len(cf_matrix))]))

# Dictionary with tickers and prices for both bid and ask
bids = dict(zip(dec_var_names, Bid_p_fix))
asks = dict(zip(dec_var_names, Ask_p_fix))

# Decision Vars

quantity_s = LpVariable.dict('S', dec_var_names, lowBound=0)                #␣
 ↪Qauntity Short
quantity_l = LpVariable.dict('L', dec_var_names, lowBound=0)                #␣
 ↪Qauntity Long
excess_spent = LpVariable.dict('reserve_used', periods[1:], lowBound=0)     #␣
 ↪Reserves spent at time t


# Intializing the Problem
dedication_2 = LpProblem('Shorts', LpMinimize)

# Objective function
dedication_2 += lpSum([bids[i]*quantity_s[i] for i in dec_var_names]) -␣
 ↪lpSum([asks[i]*quantity_l[i] for i in dec_var_names])

# Constraints

for i in range(0,len(cf_matrix[0])):
    # Liabilties constraint
    dedication_2 += lpSum([period_dict['Period {}'.
 ↪format(i+1)][j]*quantity_l[j] for j in dec_var_names]) -␣
 ↪lpSum([period_dict['Period {}'.format(i+1)][j]*quantity_s[j] for j in␣
 ↪dec_var_names]) + excess_spent[i+1] == liabilities[i]
    # Constraint on the on the reserves at time t                              ␣
 ↪                                                      #sum of the "excess␣
 ↪spent" so far
    dedication_2 += lpSum([bids[j]*quantity_s[j] for j in dec_var_names]) -␣
 ↪lpSum([asks[j]*quantity_l[j] for j in dec_var_names]) -␣
 ↪lpSum([excess_spent[j+1] for j in range(0,i)]) >= 0
    # Constraint stating that the "excess spent" at time t must be <= the␣
 ↪current reserves.
    dedication_2 += excess_spent[i+1] <= lpSum([bids[i]*quantity_s[i] for i in␣
 ↪dec_var_names]) - lpSum([asks[i]*quantity_l[i] for i in dec_var_names]) -␣
 ↪lpSum([excess_spent[j+1] for j in range(0,i)])

dedication_2.solve()
```

[109]:

```
composition = pd.DataFrame(
    [v.varValue for v in dedication_2.variables() if v.varValue > 0],
    index=[str(v.name[:-8].replace('_', ' ')+v.name[-8:].replace('_', '/')) for␣
 ↪v in dedication_2.variables() if v.varValue > 0],
    columns=['Quantity']
    )

'''Manipulating the dataframe so that it will be sorted by maturity and look␣
 ↪nice'''
composition['Dates'] = [datetime.datetime.strptime(i[-8:], '%m/%d/%y') for i in␣
 ↪composition.index.to_list()]
composition.reset_index(inplace=True)
composition.set_index('Dates',inplace=True)
composition.sort_index(axis=0, inplace=True)
composition.set_index('index', inplace = True)

'''Printing Solutions'''
'''Note that the portfolio cost is a negative because our trade balance was␣
 ↪positive'''
md('''
##### <center> Dedication Portfolio Cost & Composition  </center>

<center>

Portfolio Cost $ = \${:.2f} $ <br>

</center>

<center>

{}

'''.format(dedication_2.objective.value()*-1000000,composition.
 ↪to_markdown(colalign = ("right",))))
```

[109]:

#####

Dedication Portfolio Cost & Composition

Portfolio Cost \$ = \$54.68 \$

index	Quantity
S T 0.0 06/16/22	2.04637
S T 0.137 12/31/22	63.1674
L T 7.125 02/15/23	61.101
L T 6.25 08/15/23	0.0570833
L T 2.0 05/31/24	0.0488672
L T 1.5 11/30/24	0.0693558

index	Quantity
L T 7.625 02/15/25	0.079876
L T 6.875 08/15/25	0.0729213
L T 0.75 05/31/26	0.105428
L T 6.75 08/15/26	0.0758233
L T 6.625 02/15/27	0.0483823
L T 6.375 08/15/27	0.039985
L T 1.25 05/31/28	0.0612595
L T 5.5 08/15/28	0.0816424
L T 5.25 02/15/29	0.0638876
L T 6.125 08/15/29	0.0755646
L T 6.25 05/15/30	0.0678788

2.1.3 Discussion

Answer to questions here.

2.2 With Transaction Limits

2.2.1 Mathematical Formulation

2.2.2 Code

2.2.3 Discussion

2.3 Problem 8

Consider proposing a new portfolio of bonds using any additional consideration or change to the model that you see fit. For example, can you do something to make your portfolio of bonds immune to nonparallel changes in the term structure. Is there a better way to combine the techniques you used before. Explain clearly what you do and your results.

[94]:
```
'''
code block
'''
```

[94]: '\ncode block\n'