US Treasury Par Yield Curve

October 12, 2023

1. get data through fiscaldata api and clean it
2. get zero rate curve: boostrapping
3. get par yield curve from zero rate curve
4. compare my calculated par yield curve with offical curve

Notes: - use linear interpolation to get zero rate of critial data points(1m, 2m, 3m, 4m, 6m, 1yr, 2yr,3yr, 5yr, 7yr, 10yr, 20yr, 30yr) - assume zero rate remain constant between critical data points. For example I assume zero rate is constant in (1 year, 2 year] - assume continuous compounding - use bisection to find root - curious about par rate calculation for terms less than 0.5 year

[2]:
```
import requests
import pandas as pd



# Define the FRED API URL
base_url = 'https://api.fiscaldata.treasury.gov/services/api/fiscal_service/v1/
 ↪accounting/od/auctions_query'

# Define the parameters for the API request
params = {
    'format': 'json',
    'fields':
 ↪'record_date,security_type,security_term,auction_date,price_per100,issue_date,maturity_date
    'filter':'record_date:gte:2023-01-01'
}

# Make the API request
response = requests.get(base_url, params=params)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    data = response.json()

    # Extract the data into a Pandas DataFrame
else:
    print(f'Failed to fetch data. Status code: {response.status_code}')
```

[3]:
```
# sort by auction date
```

```
clean = pd.DataFrame(data['data']).sort_values(by = 'auction_date', ascending =
 ↪False)
```

[4]:
```
clean.head()
```

[4]:
```
   record_date security_type  … int_payment_frequency  int_rate
60  2023-01-01          Bill  …                  None      null
61  2023-01-01          Bill  …                  None      null
7   2023-01-01          Bill  …                  None      null
8   2023-01-01          Bill  …                  None      null
11  2023-01-01          Note  …           Semi-Annual  4.625000

[5 rows x 9 columns]
```

[5]:
```
clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100 entries, 60 to 17
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   record_date            100 non-null    object
 1   security_type          100 non-null    object
 2   security_term          100 non-null    object
 3   auction_date           100 non-null    object
 4   price_per100           100 non-null    object
 5   issue_date             100 non-null    object
 6   maturity_date          100 non-null    object
 7   int_payment_frequency  100 non-null    object
 8   int_rate               100 non-null    object
dtypes: object(9)
memory usage: 7.8+ KB
```

[6]:
```
# replace string null to None
clean.replace('null', None, inplace = True)
clean.replace('None', None, inplace = True)
```

[7]:
```
# convert column data type
clean['record_date'] = pd.to_datetime(clean['record_date'])
clean['auction_date'] = pd.to_datetime(clean['auction_date'])
clean['issue_date'] = pd.to_datetime(clean['issue_date'])
clean['maturity_date'] = pd.to_datetime(clean['maturity_date'])

clean['price_per100'] = clean['price_per100'].astype(float)
clean['int_rate'] = clean['int_rate'].astype(float)
clean['security_type'] = clean['security_type'].astype(str)
clean['security_term'] = clean['security_term'].astype(str)
```

```python
clean['int_payment_frequency'] = clean['int_payment_frequency'].astype(str)
```

[8]:
```python
clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100 entries, 60 to 17
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   record_date            100 non-null    datetime64[ns]
 1   security_type          100 non-null    object
 2   security_term          100 non-null    object
 3   auction_date           100 non-null    datetime64[ns]
 4   price_per100           98 non-null     float64
 5   issue_date             100 non-null    datetime64[ns]
 6   maturity_date          100 non-null    datetime64[ns]
 7   int_payment_frequency  100 non-null    object
 8   int_rate               21 non-null     float64
dtypes: datetime64[ns](4), float64(2), object(3)
memory usage: 7.8+ KB
```

notice there are missing values for int_rate and price_per100

[9]:
```python
# price should not be na
```

[10]:
```python
clean.dropna(subset = ['price_per100'], inplace = True)
```

[11]:
```python
# Regarding int_rate, it is reasonable for T bill have a NA int rate since they␣
↪are zero coupon bonds
```

[12]:
```python
clean.loc[clean['security_type'] == 'Bill'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 75 entries, 7 to 47
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   record_date            75 non-null     datetime64[ns]
 1   security_type          75 non-null     object
 2   security_term          75 non-null     object
 3   auction_date           75 non-null     datetime64[ns]
 4   price_per100           75 non-null     float64
 5   issue_date             75 non-null     datetime64[ns]
 6   maturity_date          75 non-null     datetime64[ns]
 7   int_payment_frequency  75 non-null     object
 8   int_rate               0 non-null      float64
dtypes: datetime64[ns](4), float64(2), object(3)
memory usage: 5.9+ KB
```

[13]:
```python
# but for T bonds and T notes, int_rate should not be a NA value
clean.loc[clean['security_type'] != 'Bill'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23 entries, 11 to 17
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   record_date            23 non-null     datetime64[ns]
 1   security_type          23 non-null     object
 2   security_term          23 non-null     object
 3   auction_date           23 non-null     datetime64[ns]
 4   price_per100           23 non-null     float64
 5   issue_date             23 non-null     datetime64[ns]
 6   maturity_date          23 non-null     datetime64[ns]
 7   int_payment_frequency  23 non-null     object
 8   int_rate               21 non-null     float64
dtypes: datetime64[ns](4), float64(2), object(3)
memory usage: 1.8+ KB
```

[14]:
```python
# we notice there 2 records without int_rate. They are dirty data
filtering = (clean['security_type'] != 'Bill') & (clean['int_rate'].isna())
clean.loc[filtering]
```

[14]:
```
   record_date security_type  … int_payment_frequency int_rate
10  2023-01-01          Note  …             Quarterly      NaN
48  2023-01-01          Note  …             Quarterly      NaN

[2 rows x 9 columns]
```

[15]:
```python
clean.drop(clean.loc[filtering].index, inplace = True)
```

[16]:
```python
clean.loc[clean['security_type'] != 'Bill'].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21 entries, 11 to 17
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   record_date            21 non-null     datetime64[ns]
 1   security_type          21 non-null     object
 2   security_term          21 non-null     object
 3   auction_date           21 non-null     datetime64[ns]
 4   price_per100           21 non-null     float64
 5   issue_date             21 non-null     datetime64[ns]
 6   maturity_date          21 non-null     datetime64[ns]
 7   int_payment_frequency  21 non-null     object
 8   int_rate               21 non-null     float64
```

```
dtypes: datetime64[ns](4), float64(2), object(3)
memory usage: 1.6+ KB
```

[17]:
```python
from collections import Counter
Counter(clean['int_payment_frequency'])
```

[17]:
```
Counter({'None': 75, 'Semi-Annual': 21})
```

[18]:
```python
Counter(clean['security_type'])
```

[18]:
```
Counter({'Bill': 75, 'Note': 16, 'Bond': 5})
```

[19]:
```python
clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96 entries, 7 to 17
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   record_date            96 non-null     datetime64[ns]
 1   security_type          96 non-null     object
 2   security_term          96 non-null     object
 3   auction_date           96 non-null     datetime64[ns]
 4   price_per100           96 non-null     float64
 5   issue_date             96 non-null     datetime64[ns]
 6   maturity_date          96 non-null     datetime64[ns]
 7   int_payment_frequency  96 non-null     object
 8   int_rate               21 non-null     float64
dtypes: datetime64[ns](4), float64(2), object(3)
memory usage: 7.5+ KB
```

[20]:
```python
clean.describe()
```

[20]:

	price_per100	int_rate
count	96.000000	21.000000
mean	98.743169	3.702381
std	1.288804	0.882536
min	94.813000	1.250000
25%	98.213751	3.500000
50%	99.174000	3.875000
75%	99.589333	4.125000
max	104.489600	4.750000

[21]:
```python
clean.loc[clean['security_type'] == 'Bill'].describe()
```

[21]:

	price_per100	int_rate
count	75.000000	0.0
mean	98.523294	NaN

```
std         1.129351        NaN
min        94.813000        NaN
25%        97.631472        NaN
50%        98.719681        NaN
75%        99.406458        NaN
max        99.985944        NaN
```

Now the data looks good to me: - int_rate are not null for all T note and T bonds - prices for all bonds are not null - T bill price is less than 100

[22]:
```
Counter(clean['security_term'])
```

[22]:
```
Counter({'4-Week': 14,
         '8-Week': 8,
         '5-Year': 3,
         '17-Week': 12,
         '29-Year 11-Month': 1,
         '42-Day': 5,
         '26-Week': 17,
         '13-Week': 10,
         '3-Year': 2,
         '7-Year': 4,
         '2-Year': 4,
         '10-Year': 2,
         '52-Week': 4,
         '19-Year 11-Month': 2,
         '9-Year 11-Month': 1,
         '44-Day': 2,
         '1-Day': 1,
         '19-Year 10-Month': 2,
         '7-Day': 1,
         '12-Day': 1})
```

[23]:
```
replace_dict = {
    '4-Week': 4 * 7,
    '8-Week': 8 * 7,
    '17-Week': 17 * 7,
    '5-Year': 365 * 5,
    '29-Year 11-Month': 29 * 365 + 11 * 30,
    '26-Week': 26 * 7,
    '13-Week': 13 * 7,
    '3-Year': 3 * 365,
    '42-Day': 42,
    '7-Year': 7 * 365,
    '10-Year': 10 * 365,
    '52-Week': 52 * 7,
    '19-Year 11-Month': 19 * 365 + 11 * 30,
```

```
    '9-Year 11-Month': 9 * 365 + 11 * 30,
    '44-Day': 44,
    '1-Day': 1,
    '2-Year': 2 * 365,
    '19-Year 10-Month': 19 * 365 + 10 * 30,
    '7-Day': 7,
    '12-Day': 12}

clean.replace(replace_dict, inplace = True)
```

[24]:
```
Counter(clean['security_term'])
```

[24]:
```
Counter({28: 14,
         56: 8,
         1825: 3,
         119: 12,
         10915: 1,
         42: 5,
         182: 17,
         91: 10,
         1095: 2,
         2555: 4,
         730: 4,
         3650: 2,
         364: 4,
         7265: 2,
         3615: 1,
         44: 2,
         1: 1,
         7235: 2,
         7: 1,
         12: 1})
```

1 zero rate calculation

[25]:
```
clean.head()
```

[25]:
```
   record_date security_type  …  int_payment_frequency int_rate
7   2023-01-01          Bill  …                   None      NaN
8   2023-01-01          Bill  …                   None      NaN
11  2023-01-01          Note  …            Semi-Annual    4.625
9   2023-01-01          Bill  …                   None      NaN
89  2023-01-01          Bond  …            Semi-Annual    4.125

[5 rows x 9 columns]
```

[26]:
```python
clean['rk'] = clean.groupby(['security_type', 'security_term'])['auction_date'].
    ↪rank(ascending = False, method = 'first')
clean.head()
```

[26]:
```
   record_date security_type  …  int_rate   rk
7   2023-01-01          Bill  …       NaN  1.0
8   2023-01-01          Bill  …       NaN  1.0
11  2023-01-01          Note  …     4.625  1.0
9   2023-01-01          Bill  …       NaN  1.0
89  2023-01-01          Bond  …     4.125  1.0

[5 rows x 10 columns]
```

[27]:
```python
filtering = clean['rk'] == 1
clean = clean.loc[filtering, clean.columns != 'rk'].sort_values(by =␣
    ↪'security_term')
clean
```

[27]:
```
   record_date security_type  …  int_payment_frequency int_rate
81  2023-01-01          Bill  …                   None      NaN
91  2023-01-01          Bill  …                   None      NaN
4   2023-01-01          Bill  …                   None      NaN
7   2023-01-01          Bill  …                   None      NaN
98  2023-01-01          Bill  …                   None      NaN
76  2023-01-01          Bill  …                   None      NaN
8   2023-01-01          Bill  …                   None      NaN
49  2023-01-01          Bill  …                   None      NaN
9   2023-01-01          Bill  …                   None      NaN
93  2023-01-01          Bill  …                   None      NaN
44  2023-01-01          Bill  …                   None      NaN
99  2023-01-01          Note  …            Semi-Annual    4.750
20  2023-01-01          Note  …            Semi-Annual    4.375
11  2023-01-01          Note  …            Semi-Annual    4.625
94  2023-01-01          Note  …            Semi-Annual    4.000
63  2023-01-01          Note  …            Semi-Annual    3.375
3   2023-01-01          Note  …            Semi-Annual    1.375
80  2023-01-01          Bond  …            Semi-Annual    3.875
59  2023-01-01          Bond  …            Semi-Annual    3.875
89  2023-01-01          Bond  …            Semi-Annual    4.125

[20 rows x 9 columns]
```

[28]:
```python
clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20 entries, 81 to 89
Data columns (total 9 columns):
```

```
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   record_date            20 non-null     datetime64[ns]
 1   security_type          20 non-null     object
 2   security_term          20 non-null     int64
 3   auction_date           20 non-null     datetime64[ns]
 4   price_per100           20 non-null     float64
 5   issue_date             20 non-null     datetime64[ns]
 6   maturity_date          20 non-null     datetime64[ns]
 7   int_payment_frequency  20 non-null     object
 8   int_rate               9 non-null      float64
dtypes: datetime64[ns](4), float64(2), int64(1), object(2)
memory usage: 2.1+ KB
```

1.1 get zero rates from T bills directly

[29]:
```
import math
def get_zero_rate_bill(security_term):
  r = -math.log(clean.loc[clean['security_term'] == security_term,␣
↪'price_per100'] /100) * 365/ security_term * 100
  return r
```

[30]:
```
def get_interpolated_zero_rate_bill(security_term_1, security_term_2,␣
↪target_term):
  price_1, price2 = clean.loc[clean['security_term'] == security_term_1,␣
↪'price_per100'], clean.loc[clean['security_term'] == security_term_2,␣
↪'price_per100']
  r = get_zero_rate_bill(security_term_1) +␣
↪(get_zero_rate_bill(security_term_2) - get_zero_rate_bill(security_term_1)) /
↪ (security_term_2 - security_term_1) * (target_term - security_term_1)
  return r
```

[31]:
```
# 1 Mo zero rate
rate_1m = get_interpolated_zero_rate_bill(28, 42, 30)
rate_1m
```

[31]: 5.37459590005075

[32]:
```
# 2 Mo zero rate
rate_2m = get_interpolated_zero_rate_bill(56, 91, 60)
rate_2m
```

[32]: 5.429352551217961

[33]:
```
# 3 Mo zero rate
rate_3m = get_interpolated_zero_rate_bill(56, 91, 91)
rate_3m
```

[33]: 5.451038619332669

[34]:
```
# 4 Mo zero rate
rate_4m = get_interpolated_zero_rate_bill(119,182,120)
rate_4m
```

[34]: 5.467350637920455

[35]:
```
# 6 Mo zero rate
rate_6m = get_interpolated_zero_rate_bill(119, 182, 180)
rate_6m
```

[35]: 5.4475732925713665

[36]:
```
# 1 Yr zero rate
rate_1yr = get_zero_rate_bill(364)
rate_1yr
```

[36]: 5.340998405042217

1.2 Bootstrapping for coupon bearing bonds

using bisection to find roots - pros: 100% converge - cons: need to find a and b s.t f(a) * f(b) < 0

[37]:
```
# 2 year
# I need to interpolate 0.5 year, 1 year, 1.5 year and 2 year zero rate
# assume (1, 2] zero rate stays constant
```

[38]:
```
import math
from scipy.optimize import root_scalar


def equation(rate_2yr):
    return 3.875/2 * math.e ** (-rate_6m/100 * 1/2) + \
           3.875/2 * math.e ** (-rate_1yr/100 * 2/2) + \
           3.875/2 * math.e ** (-rate_2yr/100 * 3/2) + \
           (3.875/2 + 100) * math.e ** (-rate_2yr/100 * 4/2) - 99.820997


solution = root_scalar(equation, bracket=[0, 100])
rate_2yr_solution = solution.root

rate_2yr = rate_2yr_solution
rate_2yr
```

[38]: 3.9089618015469574

[39]:
```python
# 3 year
# I need to interpolate 0.5 year, 1 year, 1.5 year, 2 year, 2.5 year, 3 year␣
↪zero rate
# assume (2, 3] zero rate stays constant
```

[40]:
```python
# Define the equation as a function
def equation(rate_3yr):
    return 4.375/2 * math.e ** (-rate_6m/100 * 1/2) + \
           4.375/2 * math.e ** (-rate_1yr/100 * 2/2) + \
           4.375/2 * math.e ** (-rate_2yr/100 * 3/2) + \
           4.375/2 * math.e ** (-rate_2yr/100 * 4/2) + \
           4.375/2 * math.e ** (-rate_3yr/100 * 5/2) + \
           (4.375/2 + 100) * math.e ** (-rate_3yr/100 * 6/2) - 99.936014



solution = root_scalar(equation, bracket=[0, 100])
rate_3yr_solution = solution.root

rate_3yr = rate_3yr_solution
rate_3yr
```

[40]: 4.349989582049794

[41]:
```python
# 5 year
# assume (3, 5] zero rate stays constant
```

[42]:
```python
def equation(rate_5yr):
    return 4.625/2 * math.e ** (-rate_6m/100 * 1/2) + \
           4.625/2 * math.e ** (-rate_1yr/100 * 2/2) + \
           4.625/2 * math.e ** (-rate_2yr/100 * 3/2) + \
           4.625/2 * math.e ** (-rate_2yr/100 * 4/2) + \
           4.625/2 * math.e ** (-rate_3yr/100 * 5/2) + \
           4.625/2 * math.e ** (-rate_3yr/100 * 6/2) + \
           4.625/2 * math.e ** (-rate_5yr/100 * 7/2) + \
           4.625/2 * math.e ** (-rate_5yr/100 * 8/2) + \
           4.625/2 * math.e ** (-rate_5yr/100 * 9/2) + \
           (4.625/2 + 100) * math.e ** (-rate_5yr/100 * 10/2) - 99.849471



solution = root_scalar(equation, bracket=[0, 100])
rate_5yr_solution = solution.root

rate_5yr = rate_5yr_solution
rate_5yr
```

[42]: 4.6185430316877625

[43]:
```
# 7 year
# assume (5, 7] zero rate stays constant
7 * 365
```

[43]: 2555

[44]:
```
# Define the equation as a function
def equation(rate_7yr):
    return 4/2 * math.e ** (-rate_6m/100 * 1/2) + \
           4/2 * math.e ** (-rate_1yr/100 * 2/2) + \
           4/2 * math.e ** (-rate_2yr/100 * 3/2) + \
           4/2 * math.e ** (-rate_2yr/100 * 4/2) + \
           4/2 * math.e ** (-rate_3yr/100 * 5/2) + \
           4/2 * math.e ** (-rate_3yr/100 * 6/2) + \
           4/2 * math.e ** (-rate_5yr/100 * 7/2) + \
           4/2 * math.e ** (-rate_5yr/100 * 8/2) + \
           4/2 * math.e ** (-rate_5yr/100 * 9/2) + \
           4/2 * math.e ** (-rate_5yr/100 * 10/2) + \
           4/2 * math.e ** (-rate_7yr/100 * 11/2) + \
           4/2 * math.e ** (-rate_7yr/100 * 12/2) + \
           4/2 * math.e ** (-rate_7yr/100 * 13/2) + \
           (4/2 + 100) * math.e ** (-rate_7yr/100 * 14/2) - 99.474987



solution = root_scalar(equation, bracket=[0, 100])
rate_7yr_solution = solution.root

rate_7yr = rate_7yr_solution
rate_7yr
```

[44]: 4.006853242005691

[45]:
```
# 10 year
# assume (7, 10] zero rate stays constant
10 * 365
```

[45]: 3650

[46]:
```
clean.loc[clean['security_term'] == 10 * 365, :]
```

[46]:
```
  record_date security_type  …  int_payment_frequency int_rate
3  2023-01-01          Note  …            Semi-Annual    1.375

[1 rows x 9 columns]
```

[47]:
```
def equation(rate_10yr):
    return 1.375/2 * math.e ** (-rate_6m/100 * 1/2) + \
           1.375/2 * math.e ** (-rate_1yr/100 * 2/2) + \
           1.375/2 * math.e ** (-rate_2yr/100 * 3/2) + \
           1.375/2 * math.e ** (-rate_2yr/100 * 4/2) + \
           1.375/2 * math.e ** (-rate_3yr/100 * 5/2) + \
           1.375/2 * math.e ** (-rate_3yr/100 * 6/2) + \
           1.375/2 * math.e ** (-rate_5yr/100 * 7/2) + \
           1.375/2 * math.e ** (-rate_5yr/100 * 8/2) + \
           1.375/2 * math.e ** (-rate_5yr/100 * 9/2) + \
           1.375/2 * math.e ** (-rate_5yr/100 * 10/2) + \
           1.375/2 * math.e ** (-rate_7yr/100 * 11/2) + \
           1.375/2 * math.e ** (-rate_7yr/100 * 12/2) + \
           1.375/2 * math.e ** (-rate_7yr/100 * 13/2) + \
           1.375/2 * math.e ** (-rate_7yr/100 * 14/2) + \
           1.375/2 * math.e ** (-rate_10yr/100 * 15/2) + \
           1.375/2 * math.e ** (-rate_10yr/100 * 16/2) + \
           1.375/2 * math.e ** (-rate_10yr/100 * 17/2) + \
           1.375/2 * math.e ** (-rate_10yr/100 * 18/2) + \
           1.375/2 * math.e ** (-rate_10yr/100 * 19/2) + \
           (1.375/2 + 100) * math.e ** (-rate_10yr/100 * 20/2) - 99.021857




solution = root_scalar(equation, bracket=[0, 100])
rate_10yr = solution.root

rate_10yr
```

[47]: 1.3767881801017547

[48]:
```
# 20 year
# assume (10, 20] zero rate stays constant
20 * 365
```

[48]: 7300

[49]:
```
clean.loc[clean['security_term'] == 20 * 365]
```

[49]:
```
Empty DataFrame
Columns: [record_date, security_type, security_term, auction_date, price_per100,
issue_date, maturity_date, int_payment_frequency, int_rate]
Index: []
```

[50]:
```
clean.loc[clean['security_term'] > 10 * 365]
```

[50]:
```
   record_date security_type  …  int_payment_frequency int_rate
80  2023-01-01          Bond  …            Semi-Annual    3.875
59  2023-01-01          Bond  …            Semi-Annual    3.875
89  2023-01-01          Bond  …            Semi-Annual    4.125

[3 rows x 9 columns]
```

[51]:
```python
def equation(rate_20yr):
    return 3.875/2 * math.e ** (-rate_6m/100 * 1/2) + \
           3.875/2 * math.e ** (-rate_1yr/100 * 2/2) + \
           3.875/2 * math.e ** (-rate_2yr/100 * 3/2) + \
           3.875/2 * math.e ** (-rate_2yr/100 * 4/2) + \
           3.875/2 * math.e ** (-rate_3yr/100 * 5/2) + \
           3.875/2 * math.e ** (-rate_3yr/100 * 6/2) + \
           3.875/2 * math.e ** (-rate_5yr/100 * 7/2) + \
           3.875/2 * math.e ** (-rate_5yr/100 * 8/2) + \
           3.875/2 * math.e ** (-rate_5yr/100 * 9/2) + \
           3.875/2 * math.e ** (-rate_5yr/100 * 10/2) + \
           3.875/2 * math.e ** (-rate_7yr/100 * 11/2) + \
           3.875/2 * math.e ** (-rate_7yr/100 * 12/2) + \
           3.875/2 * math.e ** (-rate_7yr/100 * 13/2) + \
           3.875/2 * math.e ** (-rate_7yr/100 * 14/2) + \
           3.875/2 * math.e ** (-rate_10yr/100 * 15/2) + \
           3.875/2 * math.e ** (-rate_10yr/100 * 16/2) + \
           3.875/2 * math.e ** (-rate_10yr/100 * 17/2) + \
           3.875/2 * math.e ** (-rate_10yr/100 * 18/2) + \
           3.875/2 * math.e ** (-rate_10yr/100 * 19/2) + \
           3.875/2 * math.e ** (-rate_10yr/100 * 20/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 21/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 22/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 23/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 24/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 25/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 26/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 27/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 28/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 29/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 30/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 31/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 32/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 33/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 34/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 35/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 36/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 37/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 38/2) + \
           3.875/2 * math.e ** (-rate_20yr/100 * 39/2) + \
```

```
            (3.875/2 + 100) * math.e ** (-rate_20yr/100 * 40/2) - 98.155462



solution = root_scalar(equation, bracket=[0, 100])
rate_20yr = solution.root

rate_20yr
```

[51]: 4.121883985034735

```
[52]: # 30 Yr
      # assume (20, 30] zero rate stays constant
      30 * 365
```

[52]: 10950

```
[53]: clean.loc[clean['security_term'] > 20 * 365]
```

```
[53]:    record_date security_type  …  int_payment_frequency int_rate
      89  2023-01-01          Bond  …            Semi-Annual    4.125

      [1 rows x 9 columns]
```

```
[54]: def equation(rate_30yr):
          return 4.125/2 * math.e ** (-rate_6m/100 * 1/2) + \
                 4.125/2 * math.e ** (-rate_1yr/100 * 2/2) + \
                 4.125/2 * math.e ** (-rate_2yr/100 * 3/2) + \
                 4.125/2 * math.e ** (-rate_2yr/100 * 4/2) + \
                 4.125/2 * math.e ** (-rate_3yr/100 * 5/2) + \
                 4.125/2 * math.e ** (-rate_3yr/100 * 6/2) + \
                 4.125/2 * math.e ** (-rate_5yr/100 * 7/2) + \
                 4.125/2 * math.e ** (-rate_5yr/100 * 8/2) + \
                 4.125/2 * math.e ** (-rate_5yr/100 * 9/2) + \
                 4.125/2 * math.e ** (-rate_5yr/100 * 10/2) + \
                 4.125/2 * math.e ** (-rate_7yr/100 * 11/2) + \
                 4.125/2 * math.e ** (-rate_7yr/100 * 12/2) + \
                 4.125/2 * math.e ** (-rate_7yr/100 * 13/2) + \
                 4.125/2 * math.e ** (-rate_7yr/100 * 14/2) + \
                 4.125/2 * math.e ** (-rate_10yr/100 * 15/2) + \
                 4.125/2 * math.e ** (-rate_10yr/100 * 16/2) + \
                 4.125/2 * math.e ** (-rate_10yr/100 * 17/2) + \
                 4.125/2 * math.e ** (-rate_10yr/100 * 18/2) + \
                 4.125/2 * math.e ** (-rate_10yr/100 * 19/2) + \
                 4.125/2 * math.e ** (-rate_10yr/100 * 20/2) + \
                 4.125/2 * math.e ** (-rate_20yr/100 * 21/2) + \
                 4.125/2 * math.e ** (-rate_20yr/100 * 22/2) + \
```

```
            4.125/2 * math.e ** (-rate_20yr/100 * 23/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 24/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 25/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 26/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 27/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 28/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 29/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 30/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 31/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 32/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 33/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 34/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 35/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 36/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 37/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 38/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 39/2) + \
            4.125/2 * math.e ** (-rate_20yr/100 * 40/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 41/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 42/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 43/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 44/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 45/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 46/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 47/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 48/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 49/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 50/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 51/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 52/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 53/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 54/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 55/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 56/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 57/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 58/2) + \
            4.125/2 * math.e ** (-rate_30yr/100 * 59/2) + \
            (4.125/2 + 100) * math.e ** (-rate_30yr/100 * 60/2) - 96.329971



solution = root_scalar(equation, bracket=[0, 100])
rate_30yr = solution.root

rate_30yr
```

[54]: 4.568788969555178

[55]:
```
zero_rate_dict = {
    'Term': [30, 30 * 2, 30 * 3, 30 * 4, 30 * 6, 30 * 12, 30 * 12 * 2, 30 * 12␣
↪* 3, 30 * 12 * 5, 30 * 12 * 7, 30 * 12 * 10, 30 * 12 * 20, 30 * 12 * 30],
    'zero_rate': [rate_1m,␣
↪rate_2m,rate_3m,rate_4m,rate_6m,rate_1yr,rate_2yr,rate_3yr,rate_5yr,rate_7yr,rate_10yr,rate
}
```

[56]:
```
zeros = pd.DataFrame(zero_rate_dict)
zeros['Term'] = zeros['Term']/360
zeros
```

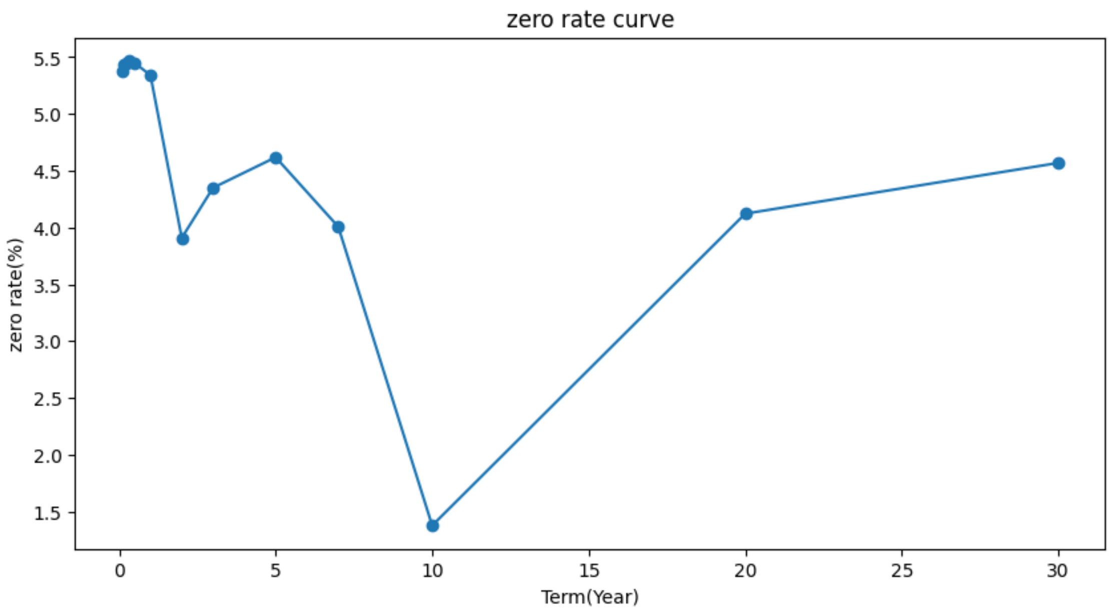
[56]:

	Term	zero_rate
0	0.083333	5.374596
1	0.166667	5.429353
2	0.250000	5.451039
3	0.333333	5.467351
4	0.500000	5.447573
5	1.000000	5.340998
6	2.000000	3.908962
7	3.000000	4.349990
8	5.000000	4.618543
9	7.000000	4.006853
10	10.000000	1.376788
11	20.000000	4.121884
12	30.000000	4.568789

[57]:
```
import matplotlib.pyplot as plt
```

[58]:
```
plt.figure(figsize=(10, 5))
plt.plot(zeros['Term'],zeros['zero_rate'], marker = 'o')
plt.xlabel('Term(Year)')
plt.ylabel('zero rate(%)')
plt.title('zero rate curve')
```

[58]: Text(0.5, 1.0, 'zero rate curve')



2 yield curve calculation

from zero rate curve we could get par yield curve

[59]:
```
zeros['discount_factor'] = math.e **(-zeros['Term'] * zeros['zero_rate']/100)
zeros
```

[59]:

	Term	zero_rate	discount_factor
0	0.083333	5.374596	0.995531
1	0.166667	5.429353	0.990992
2	0.250000	5.451039	0.986465
3	0.333333	5.467351	0.981941
4	0.500000	5.447573	0.973130
5	1.000000	5.340998	0.947991
6	2.000000	3.908962	0.924799
7	3.000000	4.349990	0.877657
8	5.000000	4.618543	0.793797
9	7.000000	4.006853	0.755421
10	10.000000	1.376788	0.871379
11	20.000000	4.121884	0.438508
12	30.000000	4.568789	0.253945

for T bills with term <= 0.5 year, to get par rate, I assume, there is only one coupon payment on maturity date. Otherwise there is no way to make its price equal to face value

[60]:
```python
par_yield_ls = []
for i in range(5):
  df = zeros.loc[i, 'discount_factor']
  def equation(c):
    return (100 + c/2) * df - 100

  solution = root_scalar(equation, bracket=[0, 100])
  par_yield_ls.append(solution.root)

par_yield_ls
```

[60]:
```
[0.8977749732723691,
 1.8179972351106546,
 2.744175096621386,
 3.678316360651421,
 5.52244163511177]
```

[61]:
```python
data = zeros.loc[zeros['Term']>= 0.5].copy().reset_index().iloc[:, 1:]
data
```

[61]:

	Term	zero_rate	discount_factor
0	0.5	5.447573	0.973130
1	1.0	5.340998	0.947991
2	2.0	3.908962	0.924799
3	3.0	4.349990	0.877657
4	5.0	4.618543	0.793797
5	7.0	4.006853	0.755421
6	10.0	1.376788	0.871379
7	20.0	4.121884	0.438508
8	30.0	4.568789	0.253945

[62]:
```python
# 1 year par rate
def equation(c):
    return c/2 * 0.973130 + (c/2 + 100) * 0.947991 - 100


solution = root_scalar(equation, bracket=[0, 100])
print(solution.root)
par_yield_ls.append(solution.root)
```

```
5.414442921606698
```

[63]:
```python
# 2 year par rate
def equation(c):
    return c/2 * 0.973130 + c/2 * 0.947991 + c/2 * 0.924799 + (c/2 + 100) * 0.
 ↪924799- 100
```

```
solution = root_scalar(equation, bracket=[0, 100])
print(solution.root)
par_yield_ls.append(solution.root)
```

3.988682264576066

[64]:
```
# 3 year par rate
def equation(c):
    return c/2 * 0.973130 + c/2 * 0.947991 + c/2 * 0.924799+ c/2 * 0.924799 + c/
 ↪2 * 0.877657+ (c/2 + 100) * 0.877657- 100


# Use root_scalar to find the root
solution = root_scalar(equation, bracket=[0, 100])
print(solution.root)
par_yield_ls.append(solution.root)
```

4.427878009414712

[65]:
```
# 5 year par rate
def equation(c):
    return c/2 * 0.973130 + c/2 * 0.947991 + c/2 * 0.924799+ c/2 * 0.924799 + c/
 ↪2 * 0.877657+ c/2 * 0.877657 + c/2 * 0.793797+ c/2 * 0.793797+ c/2 * 0.
 ↪793797+ (c/2 + 100) * 0.793797- 100


solution = root_scalar(equation, bracket=[0, 100])
print(solution.root)
par_yield_ls.append(solution.root)
```

4.739633667504826

[66]:
```
# 7 year par rate
def equation(c):
    return c/2 * 0.973130 + c/2 * 0.947991 + c/2 * 0.924799+ c/2 * 0.924799 + c/
 ↪2 * 0.877657+ c/2 * 0.877657 + c/2 * 0.793797+ c/2 * 0.793797+ c/2 * 0.
 ↪793797+ c/2 * 0.793797- 100 +\
    c/2 * 0.755421 + c/2 * 0.755421 + c/2 * 0.755421 + (c/2 + 100) * 0.755421


solution = root_scalar(equation, bracket=[0, 100])
print(solution.root)
par_yield_ls.append(solution.root)
```

4.172668805215089

[67]:
```
# 10 year par rate
def equation(c):
    return c/2 * 0.973130 + c/2 * 0.947991 + c/2 * 0.924799+ c/2 * 0.924799 + c/
↪2 * 0.877657+ c/2 * 0.877657 + c/2 * 0.793797+ c/2 * 0.793797+ c/2 * 0.
↪793797+ c/2 * 0.793797- 100 +\
    c/2 * 0.755421 + c/2 * 0.755421 + c/2 * 0.755421 + c/2  * 0.755421 +\
    c/2 * 0.871379 + c/2 * 0.871379 + c/2 * 0.871379 +  c/2 * 0.871379 +  c/2 *␣
↪0.871379 +  (c/2 + 100) * 0.871379



solution = root_scalar(equation, bracket=[0, 100])
print(solution.root)
par_yield_ls.append(solution.root)
```

```
1.5175463606395756
```

[68]:
```
# 20 year par rate
def equation(c):
    return c/2 * 0.973130 + c/2 * 0.947991 + c/2 * 0.924799+ c/2 * 0.924799 + c/
↪2 * 0.877657+ c/2 * 0.877657 + c/2 * 0.793797+ c/2 * 0.793797+ c/2 * 0.
↪793797+ c/2 * 0.793797- 100 +\
    c/2 * 0.755421 + c/2 * 0.755421 + c/2 * 0.755421 + c/2  * 0.755421 +\
    c/2 * 0.871379 + c/2 * 0.871379 + c/2 * 0.871379 +  c/2 * 0.871379 +  c/2 *␣
↪0.871379 +  c/2 * 0.871379 +\
    c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.
↪438508 + c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.438508 +␣
↪(c/2 + 100) * 0.438508



solution = root_scalar(equation, bracket=[0, 100])
print(solution.root)

par_yield_ls.append(solution.root)
```

```
5.263265692453398
```

[69]:
```
# 30 year par rate
def equation(c):
    return c/2 * 0.973130 + c/2 * 0.947991 + c/2 * 0.924799+ c/2 * 0.924799 + c/
↪2 * 0.877657+ c/2 * 0.877657 + c/2 * 0.793797+ c/2 * 0.793797+ c/2 * 0.
↪793797+ c/2 * 0.793797- 100 +\
    c/2 * 0.755421 + c/2 * 0.755421 + c/2 * 0.755421 + c/2  * 0.755421 +\
    c/2 * 0.871379 + c/2 * 0.871379 + c/2 * 0.871379 +  c/2 * 0.871379 +  c/2 *␣
↪0.871379 +  c/2 * 0.871379 +\
    c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.
↪438508 + c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.438508 + c/2 * 0.438508 +␣
↪c/2 * 0.438508 +\
```

```
    c/2 * 0.253945 + c/2 * 0.253945 + c/2 * 0.253945 + c/2 * 0.253945 + c/2 * 0.
↪253945 + c/2 * 0.253945 + c/2 * 0.253945 + c/2 * 0.253945 + c/2 * 0.253945 +␣
↪(c/2 + 100) * 0.253945


solution = root_scalar(equation, bracket=[0, 100])
print(solution.root)

par_yield_ls.append(solution.root)
```

```
6.24948980572682
```

[70]:
```
par_rate_dict = {
    'Term': zeros['Term'],
    'par_rate': par_yield_ls,
    'official_rate': [5.55,        5.60,        5.55,        5.61,        5.
↪53,        5.46,        5.03,        4.80,        4.60,        4.
↪61,        4.59,        4.92,        4.73]
}
par_rates= pd.DataFrame(par_rate_dict)
par_rates
```

[70]:

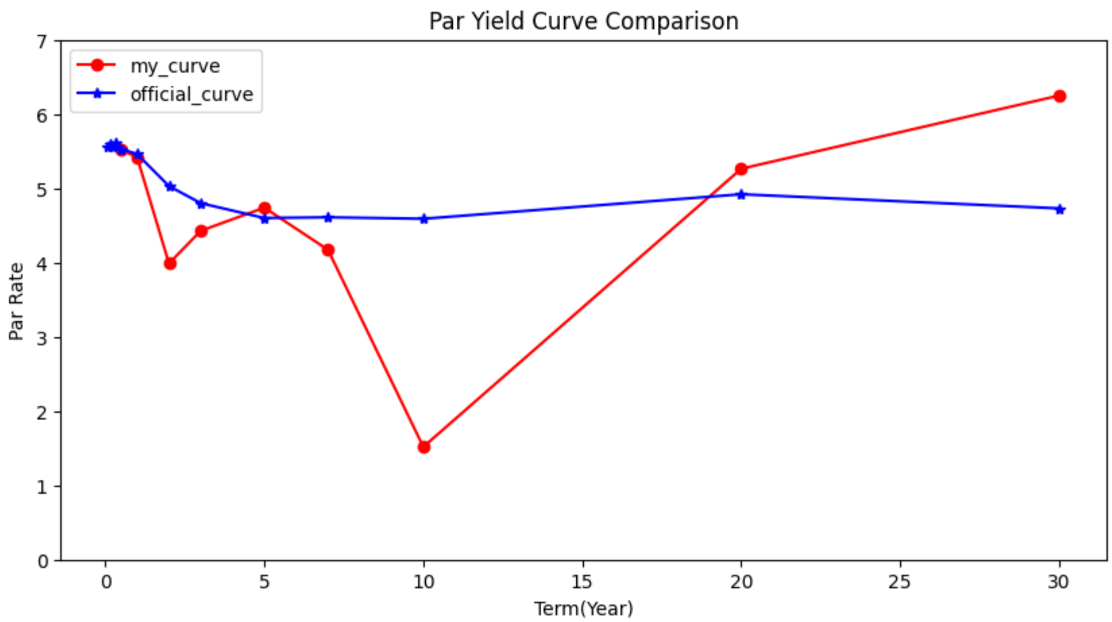
	Term	par_rate	official_rate
0	0.083333	0.897775	5.55
1	0.166667	1.817997	5.60
2	0.250000	2.744175	5.55
3	0.333333	3.678316	5.61
4	0.500000	5.522442	5.53
5	1.000000	5.414443	5.46
6	2.000000	3.988682	5.03
7	3.000000	4.427878	4.80
8	5.000000	4.739634	4.60
9	7.000000	4.172669	4.61
10	10.000000	1.517546	4.59
11	20.000000	5.263266	4.92
12	30.000000	6.249490	4.73

[73]:
```
plt.figure(figsize=(10, 5))
plt.plot(par_rates['Term'].iloc[4:, ], par_rates['par_rate'].iloc[4:, ],␣
↪label='my_curve', linestyle='-', marker='o', color='red')
plt.plot(par_rates['Term'], par_rates['official_rate'], label='official_curve',␣
↪linestyle='-', marker='*', color='blue')

plt.xlabel('Term(Year)')
plt.ylabel('Par Rate')
plt.ylim(0, 7)
plt.title('Par Yield Curve Comparison')
```

```
plt.legend()
```

[73]: <matplotlib.legend.Legend at 0x7f9293118070>



[ ]: