Computational Problem Set #2 Solutions

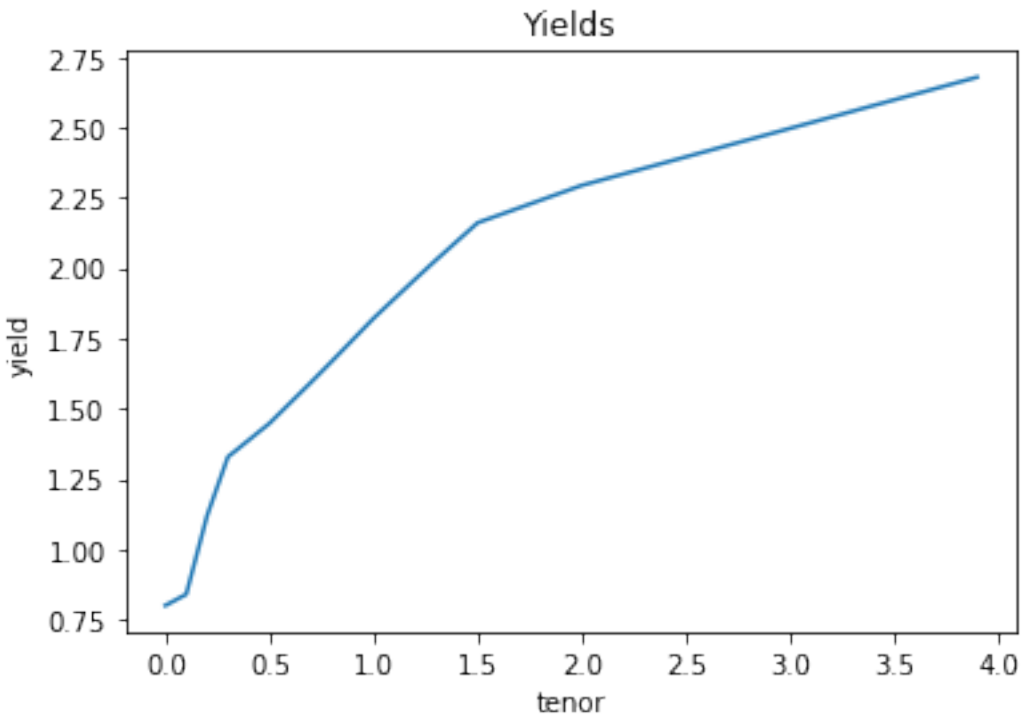
Problem 1
(a)

```
In [78]: import fixedincome as fi
         import numpy as np
         import matplotlib.pyplot as plt
```

We start by rebuilding the curve we constructed in class, using the tools in the fixedincome module. Here is the data from lecture we used:

Type	Tenor	Rate/Price
Deposits	1M	0.8
	2M	1.0
	3M	1.3
Futures	6M	98.4
	9M	98.0
	12M	97.6
	15M	97.3
	18M	97.0
Swaps	2Y	2.3
	4Y	2.7

```
In [79]: dates = [1/12, 2/12, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 2.0, 4.0]
         deposits = [0.8, 1.0, 1.3]
         futures = [98.4, 98.0, 97.6, 97.3, 97.0]
         swaprates = [2.3, 2.7]

In [80]: yc = fi.curve_factory(dates=dates, libor=deposits, futures=futures, swaps=swaprates)

In [81]: yc.plot_yields()
```



Thus we have confirmed the result from lecture by comparing yield curve plots.
We reproduce the data from problem set 5 here:

Type	Tenor	Rate/Price
Deposits	1W	2.0
	1M	2.2
	2M	2.27
	3M	2.36
Futures	6M	97.4
	9M	97.0
Swaps	1Y	3.0
	2Y	3.6
	3Y	3.95
	4Y	4.2

```
In [82]: dates = [1/52, 1/12, 2/12, 0.25, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0]
         deposits = [2.0, 2.2, 2.27, 2.36]
         futures = [97.4, 97.0]
         swaprates = [3.0, 3.6, 3.95, 4.2]

In [83]: yc2 = fi.curve_factory(dates=dates, libor=deposits, futures=futures, swaps=swaprates)

In [84]: yc2.get_tenors()

Out[84]: [0.019230769230769232,
          0.08333333333333333,
```

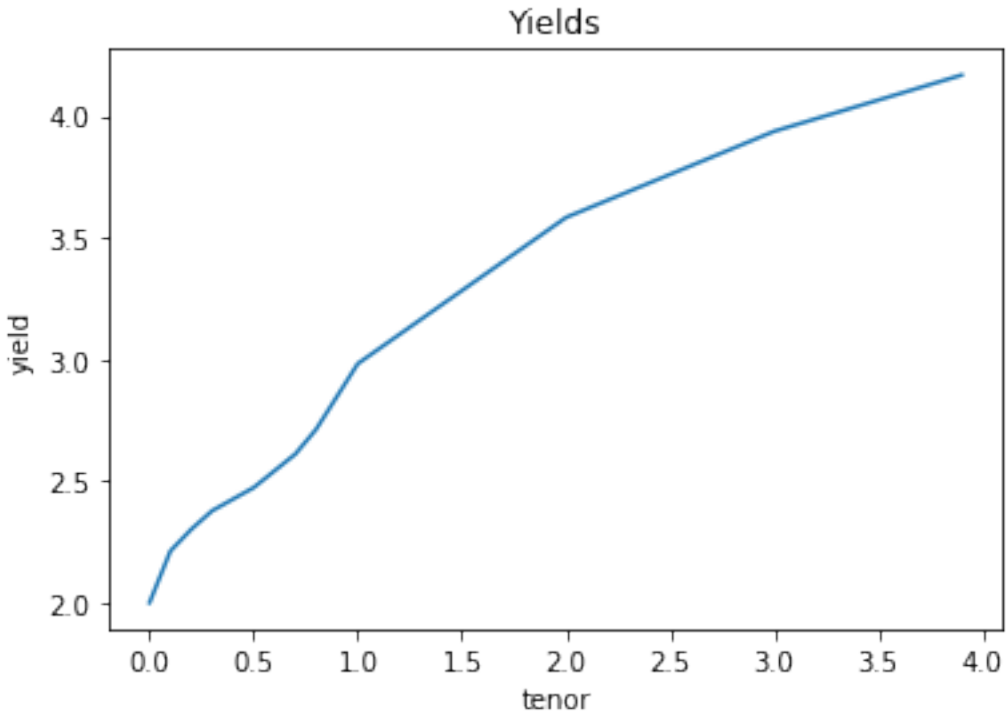
```
           0.16666666666666666,
           0.25,
           0.5,
           0.75,
           1.0,
           2.0,
           3.0,
           4.0]
```

```
In [85]: yc2.get_rates()
```

```
Out[85]: [1.9996154832063695,
          2.197985794764072,
          2.265716716659804,
          2.3530652632622195,
          2.4723258511729345,
          2.644485879275427,
          2.9815170310766788,
          3.5823684733174996,
          3.936610028120395,
          4.193130458902937]
```

These rates agree with the results from constructing the curve by hand (compare with the solutions for problem set 5). And here we plot the results:

```
In [86]: yc2.plot_yields()
```



(b)

To reprice the calibration instruments, recall that simply compounded LIBOR rates $L(0,T)$ are related to discount factors (bond prices) by

$$L(0,T) = \frac{1 - P(0,T)}{TP(0,T)}$$

The discount factors $P(0,T)$ can be extracted directly from the yield curve object:

```
In [87]: df1M = yc.discount_factor(1/12)
         l1M = (1.0 - df1M)/((1/12)*df1M)
         print(100*l1M)

0.7999999999999131
```

```
In [88]: df2M = yc.discount_factor(2/12)
         l2M = (1.0 - df2M)/((2/12)*df2M)
         print(100*l2M)

0.9999999999999991
```

```
In [89]: df3M = yc.discount_factor(0.25)
         l3M = (1.0 - df3M)/(0.25*df3M)
         print(100*l3M)

1.2999999999999805
```

This reproduces the LIBOR rates used to calibrate the short end of the curve.

To check the futures prices, recall that the futures price is

$$P = 100 - F(0,T_1,T_2)$$

where $F(0,T_1,T_2)$ is the Eurodollar futures rate. Also, recall we use the forward rates $L(0,T_1,T_2)$ as proxies for the futures rates. And recall that the futures rates are related to discount factors by

$$L(0,T_1,T_2) = \frac{P(0,T_1) - P(0,T_2)}{(T_1 - T_2)P(0,T_2)}$$

This shows us how to derive futures prices from the discount factors produced by the yield curve:

```
In [90]: df6M = yc.discount_factor(0.5)
         l3M6M = (df3M - df6M)/(0.25*df6M)
         p6M = 100 - 100 * l3M6M
         print(p6M)
```

```
98.39999999999999
```

```
In [91]: df9M = yc.discount_factor(0.75)
         l6M9M = (df6M - df9M)/(0.25*df9M)
         p9M = 100 - 100 * l6M9M
         print(p9M)
```

```
98.00000000000004
```

```
In [92]: df12M = yc.discount_factor(1.0)
         l9M12M = (df9M - df12M)/(0.25*df12M)
         p12M = 100 - 100 * l9M12M
         print(p12M)
```

```
97.6
```

```
In [93]: df15M = yc.discount_factor(1.25)
         l12M15M = (df12M - df15M)/(0.25*df15M)
         p15M = 100 - 100 * l12M15M
         print(p15M)
```

```
97.29999999999998
```

```
In [94]: df18M = yc.discount_factor(1.5)
         l15M18M = (df15M - df18M)/(0.25*df18M)
         p18M = 100 - 100 * l15M18M
         print(p18M)
```

```
96.99999999999996
```

Thus we reproduce the prices of all the futures used to calibrate the middle section of the curve.

Finally, reproduce the calibrating swap rates, recall the expression for the fair swap rate in terms of the discount factors $P(0,T)$:

$$S(T_J) = \frac{2(1 - P(0,T_J))}{\sum_{j=1}^{J} P(0,T_j)}$$

This assumes this is a swap based on 6 month LIBOR for the floating rate and making semi-annual payments. The sum in the deonominator is the sum over all the payment dates of the swap. Most importantly, the formula shows we can compute implied swap rates using nothing but discount factors, so we proceed doing so using discount factors extracted from the yield curve.

```
In [95]: df2Y = yc.discount_factor(2.0)
         swap2Y = 2 * (1.0 - df2Y)/(df6M + df12M + df18M + df2Y)
         print(100 * swap2Y)
```

```
2.3000000383392742
```

```
In [96]: df30M = yc.discount_factor(2.5)
         df3Y = yc.discount_factor(3.0)
         df42M = yc.discount_factor(3.5)
         df4Y =yc.discount_factor(4.0)
         swap4Y = 2*(1.0 - df4Y)/(df6M + df12M + df18M + df2Y + df30M + df3Y + df42M + df4Y)
         print(100 * swap4Y)
```

```
2.7000002171062056
```

We have thus replicated the prices of all the calibration instruments, from which we may justifiably derive some comfort that our curve building technology is working. Now we'll work through the same procedure for the yield curve from problem set 5 (yc2 in the work we've done here).

```
In [162]: df1W = yc2.discount_factor(1/52)
          l1W = (1.0 - df1W)/((1/52) * df1W)
          print(100*l1W)
```

```
2.0000000000000946
```

```
In [173]: df1M = yc2.discount_factor(1/12)
          l1M = (1.0 - df1M)/((1/12) * df1M)
          print(100*l1M)
```

```
2.2000000000000814
```

```
In [164]: df2M = yc2.discount_factor(2/12)
          l2M = (1.0 - df2M)/((2/12) * df2M)
          print(100*l2M)
```

```
2.2699999999999254
```

```
In [166]: df3M = yc2.discount_factor(0.25)
          l3M = (1.0 - df3M)/(0.25 * df3M)
          print(100*l3M)
```

```
2.3600000000000216
```

```
In [167]: df6M = yc2.discount_factor(0.5)
          l3M6M = (df3M - df6M)/(0.25*df6M)
          p6M = 100 - 100 * l3M6M
          print(p6M)
```

```
97.4
```

```
In [168]: df9M = yc2.discount_factor(0.75)
          l6M9M = (df6M - df9M)/(0.25*df9M)
          p9M = 100 - 100 * l6M9M
          print(p9M)
```

```
96.99999999999999
```

```
In [169]: df1Y = yc2.discount_factor(1.0)
          swap1Y = 2 * (1.0 - df1Y)/(df6M + df1Y)
          print(100*swap1Y)
```

```
2.9999991872156397
```

```
In [170]: df1p5Y = yc2.discount_factor(1.5)
          df2Y = yc2.discount_factor(2.0)
          swap3Y = 2 * (1.0 - df2Y)/(df6M + df1Y + df1p5Y + df2Y)
          print(100*swap3Y)
```

```
3.6000003568460226
```

```
In [171]: df2p5Y = yc2.discount_factor(2.5)
          df3Y = yc2.discount_factor(3.0)
          swap3Y = 2 * (1.0 - df3Y)/(df6M + df1Y + df1p5Y + df2Y + df2p5Y + df3Y)
          print(100*swap3Y)
```

```
3.950001006938194
```

```
In [172]: df3p5Y = yc2.discount_factor(3.5)
          df4Y = yc2.discount_factor(4.0)
          swap4Y = 2 * (1.0 - df4Y)/(df6M + df1Y + df1p5Y + df2Y + df2p5Y + df3Y + df3p5Y + df4Y
          print(100*swap4Y)
```

```
4.200001692856424
```

Upon comparing with the table of originating data we see that we have correctly priced all the calibration instruments.

(c)

Recall that the value of an FRA from the lender's perspective is

$$V(0) = N(P(0,T_2)(T_2 - T_1)K + P(0,T_2) - P(0,T_1))$$

where N is the principal, K is the contract rate and the loan term is from T_1 to T_2. We are given N=\$100,000, K=2%=0.02, and the dates T_1=3 and T_2=3.5. We can get the discount factors from the yield curve and evaluate the formula:

```
In [110]: df1 = yc2.discount_factor(3.0)
          df2 = yc2.discount_factor(3.5)
          V = 100000*(df2 * 0.5 * 0.02 + df2 - df1)
          print(V)

-1254.8198358152129
```

The fair contract rate is the forward rate:

$$K = \frac{P(0,T_1) - P(0,T_2))}{(T_2 - T_1)P(0,T_2)}$$

```
In [111]: K = (df1 - df2)/(0.5*df2)
          print(100*K)

4.893334502928242
```

(d)

For swap pricing, we use the same formula we used in part (b), but for a swap making quarterly payments, we will need discount factors every 3 months.

```
In [174]: df3M = yc2.discount_factor(0.25)
          df6M = yc2.discount_factor(0.5)
          df9M = yc2.discount_factor(0.75)
          df12M = yc2.discount_factor(1.0)
          df15M = yc2.discount_factor(1.25)
          df18M = yc2.discount_factor(1.5)
          df21M = yc2.discount_factor(1.75)
          df24M = yc2.discount_factor(2.0)

In [175]: swap4Y = 4 * (1 - df24M)/(df3M+df6M+df9M+df12M+df15M+df18M+df21M+df24M)
          print(100*swap4Y)

3.5829073672588927
```

To value the described swap recall the formula from lecture, that the value of a swap, from the payer's perspective, is

$$V = N(1 - P(0,T_J)) - \frac{SN}{m}\sum_{j=1}^{J} P(0,T_j)$$

where N is the notional amount, S is the contract swap rate, and m is the payment frequency (=4 in our case). Since no notional value was given, we calculate the value per dollar notional, so, effectively set N=1. And we are given \$S=\$3%=0.03:

```
In [176]: V = 1-df24M - 0.03/4*(df3M+df6M+df9M+df12M+df15M+df18M+df21M+df24M)
          print(V)
```

```
0.011248615516038618
```

Problem 2

Note that equation (1) is the same arbitrage relationship that implies the formula for forward interest rates in terms of bond prices. It simply expresses that if I invest from now until time T_2 I should get the same return as investing up to time T_1 at prevailing interest rates, and then rolling that investment over into an investment from time T_1 to T_2 at a rate I lock in today.

We start by building a yield curve object based on the suggested rates:

```
In [115]: tenors = [0.5, 1.0, 2.0, 3.0, 5.0, 7.0, 10.0]
          rates = [1.65, 2.0, 2.7, 3.1, 3.85, 4.2, 4.3]
          yc = fi.curve_factory(dates=tenors, rates=rates)
```

Starting from the case of annual compounding ($m = 1$), we evaluate both sides of the formula (1). Rather than using the suggested dates $T_1 = 2$ and $T_2 = 5$.

Note that we must divide by 100 because these methods return rates in percentage form, but formula (1) assumes the rates are expressed as decimals.

```
In [116]: l2Y = yc.spot_rate(2.0, compounding=1)/100
          l2Y5Y = yc.forward_rate(2.0, 5.0, compounding=1)/100
          l5Y = yc.spot_rate(5.0, compounding=1)/100

In [117]: lhs = (1.0 + l2Y)**2 * (1.0 + l2Y5Y)**3
          print(lhs)

1.2122765037074443


In [118]: rhs = (1.0 + l5Y)**5
          print(rhs)

1.2122765037074434
```

So the numbers check out and the test is passed. One small issue with the dates just used is that they correspond to benchmark dates used to calibrate the curve. Thus this does not provide an entirely rigorous test of the curve building function of this library. So, for the rest of the tests, I will choose dates $T_1 = 4$ and $T_2 = 6$ neither of which are benchmark dates for this curve.

```
In [119]: l4Y = yc.spot_rate(4.0, compounding=1)/100
          l4Y6Y = yc.forward_rate(4.0, 6.0, compounding=1)/100
          l6Y = yc.spot_rate(6.0, compounding=1)/100

In [120]: lhs = (1.0 + l4Y)**4 * (1.0 + l4Y6Y)**2
          print(lhs)

1.2731574549175275
```

```
In [121]: rhs = (1.0 + l6Y)**6
          print(rhs)
```

```
1.273157454917527
```

Now try semiannual compounding:

```
In [122]: l4Y = yc.spot_rate(4.0, compounding=2)/100
          l4Y6Y = yc.forward_rate(4.0, 6.0, compounding=2)/100
          l6Y = yc.spot_rate(6.0, compounding=2)/100
```

```
In [123]: lhs = (1.0 + l4Y / 2)**8 * (1.0 + l4Y6Y/2)**4
          print(lhs)
```

```
1.2731574549175277
```

```
In [124]: rhs = (1.0 + l6Y/2)**12
          print(rhs)
```

```
1.2731574549175275
```

And, monthly compounding:

```
In [125]: l4Y = yc.spot_rate(4.0, compounding=12)/100
          l4Y6Y = yc.forward_rate(4.0, 6.0, compounding=12)/100
          l6Y = yc.spot_rate(6.0, compounding=12)/100
```

```
In [126]: lhs = (1.0 + l4Y/12)**48 * (1.0 + l4Y6Y/12)**24
          print(lhs)
```

```
1.2731574549175217
```

```
In [127]: rhs = (1.0 + l6Y/12)**72
          print(rhs)
```

```
1.273157454917534
```

The version of equation (1) that holds for simple compounding is

$$(1 + T_1 L(0,T))(1 + (T_2 - T_1)L(0,T_1,T_2)) = 1 + T_2 L(0,T_2)$$

```
In [128]: l4Y = yc.spot_rate(4.0, compounding=0)/100
          l4Y6Y = yc.forward_rate(4.0, 6.0, compounding=0)/100
          l6Y = yc.spot_rate(6.0, compounding=0)/100
```

```
In [129]: lhs = (1 + 4 * l4Y) * (1 + 2 * l4Y6Y)
          print(lhs)
```

```
1.2731574549175269
```

```
In [130]: rhs = 1 + 6 *l6Y
          print(rhs)
```

```
1.2731574549175269
```

Finally, we write down the continuously compounded version of (1). Denoting the continuously compounded spot rate for maturity T by $r(T)$ and the continously compounded forward rate for a loan term from T_1 to T_2 by $r(T_1, T_2)$ then the continuously compounded version of (1) is

$$e^{T_1 r(T_1)} e^{(T_2 - T_1) r(T_1, T_2)} = e^{T_2 r(T_2)}$$

By using the law of exponents and taking logarithms this equation reduces to

$$T_1 r(T_1) + (T_2 - T_1) r(T_1, T_2) = T_2 r(T_2)$$

which we then solve for the forward rate:

$$r(T_1, T_2) = \frac{T_2 r(T_2) - T_1 r(T_1)}{T_2 - T_1}$$

and this is the equation we will verify.

```
In [131]: r1 = yc.get_yield(4.0)
          r12 = yc.forward_rate(4.0, 6.0, compounding="continuous")
          r2 = yc.get_yield(6.0)
```

```
In [132]: rhs = (6.0 * r2 - 4.0 * r1)/(6-4)
          print(rhs)
```

```
5.125
```

```
In [133]: print(r12)
```

```
5.124999999999999
```

Problem 3

Forward interest rates can be thought of as "break-even rates". This means that T-forward rates must be the realized rates at time T for a bond investment to be competitive with investing cash, or investing at the rates for horizon T. This exercise explores this phenomenon.

We begin by computing the 1-year forward rates from the proposed spot curve. To begin, we instantiate the curve object from the rates we have been given.

```
In [134]: import fixedincome as fi
          tenors = [1.0, 2.0, 3.0, 5.0, 7.0, 10.0]
          rates = [1.5, 2.1, 2.5, 2.8, 3.1, 3.3]
          yc_current = fi.curve_factory(dates=tenors, rates=rates)
```

From this curve we wish to extract the 1 year forward rates corresponding to the payment dates of the proposed bond. The bond is a 5 year bond making annual payments. In 1 year, the first coupon will have been paid, so we are concerned with rates with tenors corresponding to the remaining payments at dates 2, 3, 4, and 5 years ahead. Although it is immaterial, as long as we are consistent, we use annually compounded rates since the bond is an annually paying bond.

```
In [135]: yc_current.forward_rate(1.0, 2.0, compounding=1)

Out[135]: 2.736780276348938

In [136]: yc_current.forward_rate(1.0, 3.0, compounding=1)

Out[136]: 3.045453395351694

In [137]: yc_current.forward_rate(1.0, 4.0, compounding=1)

Out[137]: 3.0798076052003465

In [138]: yc_current.forward_rate(1.0, 5.0, compounding=1)

Out[138]: 3.174340749910276
```

We will now compare these rates with what the realized spot rates would be if, in 1 year, the yield curve turns out to be the one proposed in the problem. So we insantiate a curve object for this curve:

```
In [139]: tenors1 = [1.0, 2.0, 4.0, 6.0, 9.0]
          rates1 = [1.7, 2.1, 2.4, 2.7, 3.0]
          yc_1year = fi.curve_factory(dates=tenors1, rates=rates1)
```

We now compute the spot rates corresponding to the remaining payment dates of the bond, from the point of view of 1 year hence.

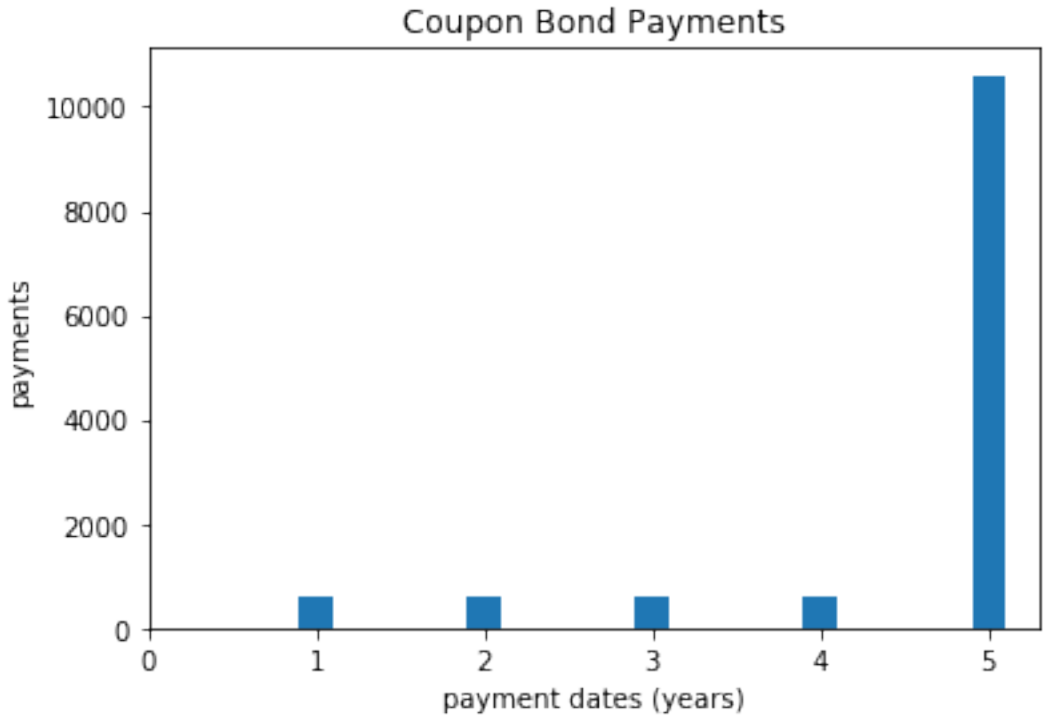
```
In [140]: yc_1year.spot_rate(1.0, compounding=1)

Out[140]: 1.7145322325240686

In [141]: yc_1year.spot_rate(2.0, compounding=1)

Out[141]: 2.122205163752877

In [142]: yc_1year.spot_rate(3.0, compounding=1)

Out[142]: 2.275503416444602

In [143]: yc_1year.spot_rate(4.0, compounding=1)
```

```
Out[143]: 2.4290317890621527
```

So in this scenario the realized spot rates corresponding to the payment dates of the remaining coupons turn out to be less than the forward rates implied by the yield curve. Remembering that there is an inverse relationship between bond prices and interest rates, we expect that holding the bond, strategy 1, to be more profitable than liquidating and investing the cash. To check this, we will value both positions in 1 year.

We start by valuing the bond at the start date. For this, we must instantiate the correct bond object. No face value was given for the problem, but it will not affect the analysis, so we may choose any convenient number. I'll go with $10,000.

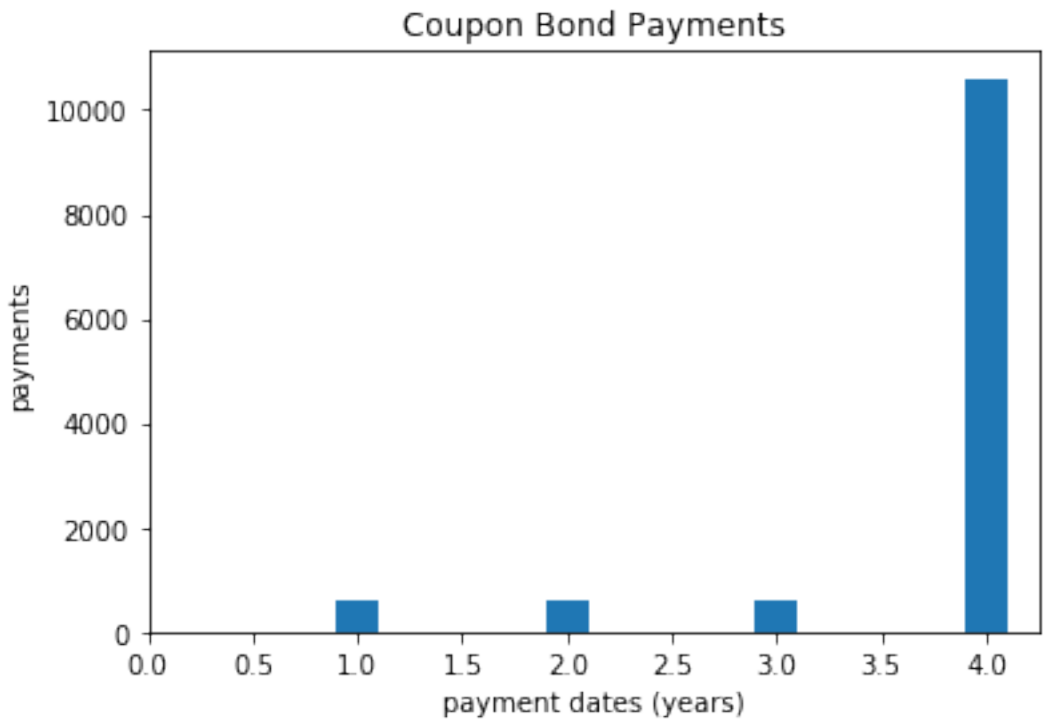
```
In [144]: bond0 = fi.create_coupon_bond(maturity=5.0, face=10000, rate=6, frequency=1)
          bond0.plot_payments()
```



We now value the 2 bond strategies.

Bond Strategy 1: To value the bond position 1 year ahead, we need a bond object reflecting what the bond's payoffs will be as of 1 year forward. For this, we simply reduce the maturity by 1 year:

```
In [145]: bond1 = fi.create_coupon_bond(maturity=4.0, face=10000, rate=6, frequency=1)
          bond1.plot_payments()
```



Now to value this strategy, we must price the bond position, but we must add to that the $600 coupon that will just have been paid. Of course, to price the bond position we must use the hypothesized curve 1 year hence:

```
In [146]: price = bond1.price(yc_1year)
          strategy1_value = price + 600
          print(strategy1_value)
```

```
11955.763281822337
```

Bond Strategy 2: In bond strategy 2, rather than holding the bond, we liquidate it, and invest the cash for 1 year at prevailing interest rates. The value of the bond position at the start is

```
In [147]: price0 = bond0.price(yc_current)
```

To carry out this strategy, we will sell the bond for this cash, and invest it for 1 year. The value of the strategy in 1 year is then simply the 1 year future value of this cash. We may calculate this by dividing by the 1 year discount factor. Thus the value of bond strategy 2 in 1 year is

```
In [148]: strategy2_value = price0/yc_current.discount_factor(1.0)
          print(strategy2_value)
```

```
11651.353259001422
```

And we see that, as predicted, bond strategy 1, holding the bond position, is the more profitable strategy.

Now we carry out the same analysis, using the proposed alternative scenario. We insantiate a new curve object for the alternative interest rates given in the problem:

```
In [149]: tenors1B = [1.0, 2.0, 4.0, 6.0, 9.0]
          rates1B = [3.0, 3.4, 3.7, 4.0, 4.2]
          yc_1yearB = fi.curve_factory(dates=tenors1B, rates=rates1B)
```

We compute the spot rates implied by this curve, corresponding to the bond payment dates, as before:

```
In [150]: yc_1yearB.spot_rate(1.0, compounding=1)

Out[150]: 3.045453395351694

In [151]: yc_1yearB.spot_rate(2.0, compounding=1)

Out[151]: 3.4584606728117917

In [152]: yc_1yearB.spot_rate(3.0, compounding=1)

Out[152]: 3.613764812806486

In [153]: yc_1yearB.spot_rate(4.0, compounding=1)

Out[153]: 3.7693020838157176
```

These spot rates exceed the forward rates computed above, so now we expect strategy 1, holding the bond position, be less profitabe than strategy 2 of liquidating and investing the money at prevailing rates. We check this prediction by, once again, comparing the value of the two positions in 1 year. The value of strategy 2, dependent entirely on the current yield curve, is unchanged, so we only need to calculate the value of strategy 1 using this alternative yield curve:

```
In [154]: priceB = bond1.price(yc_1yearB)
          strategy1_valueB = priceB + 600
          print(strategy1_valueB)

11423.978451557852
```

And, as predicted, under this scenario, strategy 1 is less profitable than strategy 2. If the realized rates at some future date are more than the forward rates derived from the prevailing yield curve, then we are better off liquidating a bond position and investing cash.