backward_induction

March 30, 2020

[1]:
```
import numpy as np
```

[2]:
```
#HW5 Q5
T=10
times= np.arange(T)
def ho_lee_rn(n, heads):
    #definition of Ho-Lee model
    an= 0.06-0.005*n
    bn= 0.01
    return an + bn*heads

def get_all_rn(times):
    #compute interest rate tree here
    res= {}
    for i in times:
        res[i]= ho_lee_rn(i, np.arange(i+1))
    return res

all_rn= get_all_rn(np.arange(T))

def backward(terminal_val, q, F= 100):
    #backward induction for vanilla bond
    #returns dictionary of prices where key= time, value= array of prices␣
 ↪ordered by ascending num heads
    prices= {T:[terminal_val], T-1: terminal_val/(1+all_rn[T-1])}
    remain_time= np.arange(T-2, -1, -1)
    for t in remain_time:
        #tprices= number of uniq prices at time t
        tprices= np.empty(t+1)
        #num_heads= 0 to t heads at time t
        num_heads= np.arange(t+1)
        for k in num_heads:
            #multiple payments: add fixed coupon
            tprices[k]= 1/(1+all_rn[t][k])*(.5*(prices[t+1][k+1] +␣
 ↪prices[t+1][k])+F*q)
        prices[t]= tprices
    return prices
```

[3]:
```
backward(105, 0.05)
```

[3]:
```
{10: [105],
 9: array([103.44827586, 102.43902439, 101.44927536, 100.4784689 ,
        99.52606635,  98.5915493 ,  97.6744186 ,  96.77419355,
        95.89041096,  95.02262443]),
 8: array([105.82710797, 103.82927172, 101.88833859, 100.00215964,
        98.16868663,  96.38596631,  94.65213526,  92.96541491,
        91.324107  ]),
 7: array([107.14945351, 104.21140595, 101.38301351,  98.65916885,
        96.03504833,  93.50609375,  91.06799547,  88.71667667]),
 6: array([107.45672789, 103.6511632 , 100.02008684,  96.55387603,
        93.24352434,  90.08059686,  87.05718905]),
 5: array([106.81540632, 102.23504786,  97.90235207,  93.80159641,
        89.91819591,  86.23861102]),
 4: array([105.31271836, 100.06542854,  95.14337192,  90.52326744,
        86.18370691]),
 3: array([103.05174493,  97.25535567,  91.862272  ,  86.84045319]),
 2: array([100.14623838,  93.92340928,  88.17884355]),
 1: array([96.71547282, 90.18885109]),
 0: array([92.87939807])}
```

[4]:
```
backward(106, 0.06)
```

[4]:
```
{10: [106],
 9: array([104.43349754, 103.41463415, 102.41545894, 101.4354067 ,
       100.47393365,  99.53051643,  98.60465116,  97.69585253,
        96.80365297,  95.92760181]),
 8: array([107.768692  , 105.74276363, 103.77445463, 101.86159064,
       100.00209909,  98.19400355,  96.43541838,  94.72454381,
        93.05966126]),
 7: array([110.00558811, 107.01314892, 104.13207908, 101.35719893,
        98.68361626,  96.10670787,  93.62210239,  91.22566442]),
 6: array([111.17414419, 107.28135961, 103.56632286, 100.01925245,
        96.63099259,  93.39296771,  90.29714074]),
 5: array([111.33116126, 106.62568539, 102.17325844,  97.95786152,
        93.9646327 ,  90.17977348]),
 4: array([110.55617628, 105.14235421, 100.06184904,  95.29088515,
        90.80759545]),
 3: array([108.9466653 , 102.94038068,  97.3487015 ,  92.13882819]),
 2: array([106.61287904, 100.13635952,  94.15305125]),
 1: array([103.67262491,  96.84948863]),
 0: array([100.24627997])}
```

[5]:
```
backward(107, 0.07)
```

[5]:
```
{10: [107],
 9: array([105.41871921, 104.3902439 , 103.38164251, 102.3923445 ,
        101.42180095, 100.46948357,  99.53488372,  98.61751152,
         97.71689498,  96.83257919]),
 8: array([109.71027604, 107.65625554, 105.66057068, 103.72102164,
        101.83551156, 100.00204079,  98.2187015 ,  96.48367271,
         94.79521553]),
 7: array([112.86172272, 109.81489189, 106.88114465, 104.05522901,
        101.3321842 ,  98.707322  ,  96.17620931,  93.73465216]),
 6: array([114.89156049, 110.91155603, 107.11255888, 103.48462887,
        100.01846084,  96.70533857,  93.53709242]),
 5: array([115.84691619, 111.01632293, 106.44416481, 102.11412663,
         98.01106949,  94.12093594]),
 4: array([115.79963419, 110.21927988, 104.98032615, 100.05850286,
         95.431484  ]),
 3: array([114.84158568, 108.6254057 , 102.83513099,  97.43720319]),
 2: array([113.0795197 , 106.34930976, 100.12725896]),
 1: array([110.629777  , 103.51012616]),
 0: array([107.61316187])}
```

[6]:
```
def Qn(all_rn, curr_t, k):
    #definition of the coupon function
    prev_r= all_rn[curr_t-1][k]
    if (prev_r > 0.085):
        return 0.035
    elif (0.035 <= prev_r <= 0.085):
        return 0.12 - prev_r
    else:
        return 0.085

def all_qn(all_rn, times):
    res= {}
    for t in times:
        q_t= np.empty(t)
        num_heads= np.arange(t)
        for k in num_heads:
            q_t[k]= Qn(all_rn, t, k)
        res[t]= q_t
    return res

#pay coupons at time 1 to 10
all_q= all_qn(all_rn, np.arange(1, 11))

def varq_backward(F= 1000):
    #inverse floater backward induction (variable coupon)
    #returns dictionary of prices where key= time, value= array of prices␣
 ↪ordered by ascending num heads
```

```
    terminal_val= F*(1+all_q[T])
    prices= {T-1: terminal_val/(1+all_rn[T-1])}
    remain_time= np.arange(T-2, -1, -1)
    for t in remain_time:
        tprices= np.empty(t+1)
        num_heads= np.arange(t+1)
        for k in num_heads:
            #coupon rate depends on k at time t but received at t+1
            tprices[k]= 1/(1+all_rn[t][k])*(.5*(prices[t+1][k+1] +␣
↪prices[t+1][k])+F*all_q[t+1][k])
        prices[t]= tprices
    return prices
```

[7]:
```
varq_backward()
```

[7]:
```
{9: array([1068.96551724, 1058.53658537, 1048.30917874, 1028.70813397,
        1009.47867299,  990.61032864,  972.09302326,  953.91705069,
         945.20547945,  936.65158371]),
 8: array([1126.22652089, 1105.26493403, 1075.48909265, 1037.23181284,
        1000.0419819 ,  963.88007098,  928.70836757,  903.26721566,
         887.20775598]),
 7: array([1171.4592463 , 1135.62996458, 1082.64158157, 1027.14397855,
         973.67232529,  922.13415746,  876.48644388,  849.53195052]),
 6: array([1202.47049072, 1143.39978181, 1071.3264572 , 1000.38504898,
         932.61985176,  869.73175988,  823.86164881]),
 5: array([1215.39626692, 1131.44796125, 1043.46516881,  959.15723039,
         880.1635403 ,  812.71585654]),
 4: array([1205.21357123, 1102.33958574, 1001.23698075,  906.2246592 ,
         820.77749854]),
 3: array([1175.86275453, 1058.56709313,  947.16508918,  845.11728267]),
 2: array([1130.68087984, 1002.70385958,  884.244099  ]),
 1: array([1072.6941893 ,  937.53425285]),
 0: array([1004.82473686])}
```

[8]:
```
#HW5 Q8
#call dates
epsilon_V= np.arange(1, 10)
epsilon_W= np.array([6,7,8,9])

def callable_backward(call_dates, F= 1000, q=0.06, call_price= 1000):
    #returns dictionary of prices where key= time, value= array of prices␣
↪ordered by ascending num heads
    prices= {T: [1000]*(T+1)}
    remain_time= np.arange(T-1, -1, -1)
    for t in remain_time:
        tprices= np.empty(t+1)
        num_heads= np.arange(t+1)
```

```
        if t in call_dates:
            for k in num_heads:
                wait= 1/(1+all_rn[t][k])*(.5*(prices[t+1][k+1] +␣
↪prices[t+1][k])+F*q)
                tprices[k]= min(call_price, wait)
        else:
            for k in num_heads:
                tprices[k]= 1/(1+all_rn[t][k])*(.5*(prices[t+1][k+1] +␣
↪prices[t+1][k])+F*q)
        prices[t]= tprices
    return prices
```

[9]:
```
callable_backward(epsilon_V)
```

[9]:
```
{10: [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000],
 9: array([1000.        , 1000.        , 1000.        , 1000.        ,
        1000.        ,  995.30516432,  986.04651163,  976.95852535,
         968.03652968,  959.2760181 ]),
 8: array([1000.        , 1000.        , 1000.        , 1000.        ,
         997.78545487,  981.94003549,  964.35418378,  947.24543809,
         930.59661263]),
 7: array([1000.        , 1000.        , 1000.        , 1000.        ,
         985.78661519,  961.06707873,  936.2210239 ,  912.25664416]),
 6: array([1000.        , 1000.        , 1000.        ,  993.2955732 ,
         965.81948314,  933.92967715,  902.97140737]),
 5: array([1000.        , 1000.        , 1000.        ,  976.11035509,
         939.41821409,  901.7977348 ]),
 4: array([1000.        , 1000.        ,  988.73129957,  951.18157438,
         907.97034671]),
 3: array([1000.        ,  999.39872018,  967.09524599,  920.53577725]),
 2: array([1000.        ,  984.19526706,  938.14533796]),
 1: array([997.24894173, 958.84535447]),
 0: array([979.28976236])}
```

[10]:
```
callable_backward(epsilon_W)
```

[10]:
```
{10: [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000],
 9: array([1000.        , 1000.        , 1000.        , 1000.        ,
        1000.        ,  995.30516432,  986.04651163,  976.95852535,
         968.03652968,  959.2760181 ]),
 8: array([1000.        , 1000.        , 1000.        , 1000.        ,
         997.78545487,  981.94003549,  964.35418378,  947.24543809,
         930.59661263]),
 7: array([1000.        , 1000.        , 1000.        , 1000.        ,
         985.78661519,  961.06707873,  936.2210239 ,  912.25664416]),
 6: array([1000.        , 1000.        , 1000.        ,  993.2955732 ,
         965.81948314,  933.92967715,  902.97140737]),
```

```
5: array([1024.15458937, 1014.35406699, 1001.56188303,  976.11035509,
       939.41821409,  901.7977348 ]),
4: array([1037.74454633, 1017.10283334,  989.46803685,  951.18157438,
       907.97034671]),
3: array([1040.59683238, 1007.85349298,  967.44113203,  920.53577725]),
2: array([1032.59539303,  988.34652124,  938.30696696]),
1: array([1014.66441434,  960.87018225]),
0: array([988.45971537])}
```

[ ]: