vasicek&swaps

May 11, 2020

[1]:
```python
import numpy as np
import pprint
from scipy.optimize import minimize
pp= pprint.PrettyPrinter(indent=4)
T=10
times= np.arange(T)
```

[2]:
```python
def ho_lee_rn(n, heads, r0, lam, sigma):
    #definition of Ho-Lee model
    sig_coeff= (heads-(n-heads))
    return r0 + lam*n + sig_coeff*sigma

def get_all_rn(times, r0, lam, sigma):
    #compute interest rate tree here
    res= {}
    for i in times:
        res[i]= ho_lee_rn(i, np.arange(i+1), r0, lam, sigma)
    return res
```

[8]:
```python
def payment(all_rn, curr_t, k, K, F=1000):
    #definition of the payment function
    prev_r= all_rn[curr_t-1][k]
    return F*(K-prev_r)

def all_pay(all_rn, times, K):
    #get all coupon payments
    res= {}
    for t in times:
        q_t= np.empty(t)
        num_heads= np.arange(t)
        for k in num_heads:
            q_t[k]= payment(all_rn, t, k, K)
        res[t]= q_t
    return res

def backward_swap(K, F=1000):
    all_payments= all_pay(all_rn, np.arange(1, 11), K)
```

```
    terminal_val= all_payments[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:
            tprices[k]= 1/(1+all_rn[t][k])*(.5*(prices[t+1][k+1] +␣
↪prices[t+1][k])+all_payments[t+1][k])
        prices[t]= tprices
    return prices
```

[9]:
```
all_rn= get_all_rn(np.arange(T), 0.06, 0, 0.005)

#q3
exercise_dates= np.array([4,5,6])
def backward_bermudan_swaption(exercise_dates, K=0.062, F=1000):
    swap_prices= backward_swap(K)
    terminal_val= swap_prices[exercise_dates[-1]]
    terminal_val[terminal_val<0]=0
    prices={exercise_dates[-1]:terminal_val}
    remain_time= np.arange(exercise_dates[-1]-1, -1, -1)
    for t in remain_time:
        tprices= np.empty(t+1)
        num_heads= np.arange(t+1)
        if t in exercise_dates:
            curr_val= swap_prices[t]
            curr_val[curr_val<0]=0
            for k in num_heads:
                wait= 1/(1+all_rn[t][k])*(.5*(prices[t+1][k+1] +␣
↪prices[t+1][k]))
                tprices[k]= max(curr_val[k], 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]))
        prices[t]= tprices
    return prices
```

[10]:
```
pp.pprint(backward_bermudan_swaption(exercise_dates))
```

```
{   0: array([23.2960203]),
    1: array([37.49541904, 11.89214399]),
    2: array([57.9465345 , 21.16879967,  4.16146703]),
    3: array([84.9885512 , 36.69917125,  8.17868405,  0.72685539]),
    4: array([116.04867859,  61.57739341,  15.85785793,   1.56273909,
          0.        ]),
```

```
5: array([122.34312246,  75.03812901,  30.27439715,   3.34426166,
     0.        ,   0.        ]),
6: array([119.17627452,  80.07398898,  42.75570065,   7.12327734,
     0.        ,   0.        ,   0.        ])}
```

[13]:
```
#q4
def backward_payer_swaption(K, exercise_date=5, F=1000):
    swap_prices= backward_swap(K)
    terminal_val= swap_prices[exercise_date]*-1
    terminal_val[terminal_val<0]=0
    prices={exercise_date:terminal_val}
    remain_time= np.arange(exercise_date-1, -1, -1)
    for t in remain_time:
        tprices= np.empty(t+1)
        num_heads= np.arange(t+1)
        for k in num_heads:
            tprices[k]= 1/(1+all_rn[t][k])*(.5*(prices[t+1][k+1] +
 ↪prices[t+1][k]))
        prices[t]= tprices
    return prices[0]

def backward_putable_swap(K):
    swap_prices=backward_swap(K)
    payer_swaption_prices= backward_payer_swaption(K)
    return swap_prices[0] + payer_swaption_prices[0]
```

[14]:
```
pp.pprint(backward_putable_swap(.062))
```

```
array([27.94391666])
```

[15]:
```
def putable_swap_t0(K):
    swap_prices=backward_swap(K)
    payer_swaption_prices=backward_payer_swaption(K)
    return (swap_prices[0]+ payer_swaption_prices[0])**2

def find_putableK():
    res= minimize(putable_swap_t0, x0=0.06)
    return res.x
```

[16]:
```
pp.pprint(find_putableK())
```

```
array([0.0572298])
```

[19]:
```
import itertools
def count_duplicates(time):
    res= {}
    bin_seq= ["".join(seq) for seq in itertools.product("01", repeat=time)]
    for seq in bin_seq:
```

```
        num_heads= seq.count("1")
        res[num_heads]= res.get(num_heads, 0) + 1
    return res

all_rn= get_all_rn(np.arange(T), 0.03, 0, 0.0025)
```

[28]:
```
#q5
def backward_bond(all_rn, terminal_val=106, q=0.06, F=100, T=T):
    #backward induction for vanilla bond
    prices= {T:[F]*10, 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

def backward_futures(all_rn, terminal_val, q, F, curr_t=5):
    #backward induction for futures
    prices= backward_bond(all_rn, terminal_val, q, F)
    dup= count_duplicates(curr_t)
    dup_val= np.array(list(dup.values()))
    total= sum(dup_val)
    return np.sum(prices[curr_t]*dup_val)/total

def compute_futures_dv01(all_rn, delta, terminal_val=106000, q=0.06, F=100000):
    futures_price= backward_futures(all_rn, terminal_val, q, F)
    all_rn2= get_all_rn(np.arange(T), 0.03+delta, 0, 0.0025)
    new_price= backward_futures(all_rn2, terminal_val, q, F)
    return -1*(new_price- futures_price)/(10000*delta)
```

[29]:
```
pp.pprint(compute_futures_dv01(all_rn, .001))
pp.pprint(compute_futures_dv01(all_rn, .0001))
```

```
49.617228732755756
49.74265290585754
```

[30]:
```
def compute_bond_dv01(all_rn, delta):
    bond_price= backward_bond(all_rn, terminal_val=106000, F=100000, T=5)
    all_rn2= get_all_rn(np.arange(T), 0.03+delta, 0, 0.0025)
    new_price= backward_bond(all_rn2, terminal_val=106000, F=100000, T=5)
```

```
    return -1*(new_price[0]- bond_price[0])/(10000*delta)
```

[31]:
```
pp.pprint(compute_bond_dv01(all_rn, .001))
pp.pprint(compute_bond_dv01(all_rn, .0001))
```

```
array([49.58823259])
array([49.71355648])
```

[34]:
```
#q6
all_rn= get_all_rn(np.arange(T), 0.06, 0, 0.005)
def g(x):
    #risk neutral measure for recombining vasicek
    compare_term= 1 + .1*(0.06-x)/0.005
    if compare_term < 0:
        return 0
    elif 0 <= compare_term <=2:
        return 1/2 + .1*(0.06-x)/(2*0.005)
    elif compare_term > 2:
        return 1

def recomb_vasicek_bond(all_rn, terminal_val=106, q=0.06, F=100, T=10):
    prices= {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
            curr_interest= all_rn[t][k]
            p_tilda= g(curr_interest)
            q_tilda= 1- p_tilda
            tprices[k]= 1/(1+curr_interest)*(p_tilda*(prices[t+1][k+1]) +␣
↪q_tilda*(prices[t+1][k]) + F*q)
        prices[t]= tprices
    return prices
```

[35]:
```
pp.pprint(recomb_vasicek_bond(all_rn))
```

```
{   0: array([100.14086385]),
    1: array([102.56205552,  97.73657584]),
    2: array([104.78711631, 100.08866588,  95.6459712 ]),
    3: array([106.72524035, 102.22729331,  97.96067835,  93.91195543]),
    4: array([108.26440223, 104.05436213, 100.04605725,  96.22842434,
        92.59107506]),
    5: array([109.2692366 , 105.44886763, 101.79588864,  98.30175272,
```

```
         94.95840602,  91.75825617]),
    6: array([109.58007261, 106.26485563, 103.07902647, 100.01654655,
         97.07170233,  94.23908559,  91.51357502]),
    7: array([109.0140011 , 106.33084321, 103.73742414, 101.23001356,
         98.80506513,  96.45920605,  94.18922732,  91.99207454]),
    8: array([107.36913773, 105.45174173, 103.58598305, 101.77002178,
        100.00209909,  98.28053301,  96.60371442,  94.97010332,
         93.37822532]),
    9: array([104.43349754, 103.41463415, 102.41545894, 101.4354067 ,
        100.47393365,  99.53051643,  98.60465116,  97.69585253,
         96.80365297,  95.92760181])}
```

[36]:

```
def recomb_vasicek_callable(call_dates, F= 1000, q=0.06, call_price= 1000):
    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:
                curr_interest= all_rn[t][k]
                p_tilda= g(curr_interest)
                q_tilda= 1- p_tilda
                wait= 1/(1+all_rn[t][k])*(p_tilda*(prices[t+1][k+1]) +␣
 ↪q_tilda*(prices[t+1][k])+F*q)
                tprices[k]= min(call_price, wait)
        else:
            for k in num_heads:
                curr_interest= all_rn[t][k]
                p_tilda= g(curr_interest)
                q_tilda= 1- p_tilda
                tprices[k]= 1/(1+all_rn[t][k])*(p_tilda*(prices[t+1][k+1]) +␣
 ↪q_tilda*(prices[t+1][k])+F*q)
        prices[t]= tprices
    return prices
```

[37]:

```
pp.pprint(recomb_vasicek_callable(np.arange(1, 10)))
```

```
{   0: array([984.35517679]),
    1: array([998.13758833, 968.69538647]),
    2: array([1000.        ,  987.33664671,  952.50095756]),
    3: array([1000.        ,  999.68994876,  973.46374226,  937.74444809]),
    4: array([1000.        , 1000.        ,  990.31435625,  960.14664349,
         925.65703836]),
    5: array([1000.        , 1000.        , 1000.        ,  979.46643525,
         949.19261846,  917.58256169]),
    6: array([1000.        , 1000.        , 1000.        ,  993.81893823,
         970.06963891,  942.39085587,  915.13575016]),
```

```
7: array([1000.        , 1000.        , 1000.        , 1000.        ,
        986.89614905,  964.59206052,  941.89227324,  919.92074541]),
8: array([1000.        , 1000.        , 1000.        , 1000.        ,
        997.78545487,  982.80533013,  966.03714421,  949.70103322,
        933.7822532 ]),
9: array([1000.        , 1000.        , 1000.        , 1000.        ,
       1000.        ,  995.30516432,  986.04651163,  976.95852535,
        968.03652968,  959.2760181 ]),
10: [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000]}
```

[ ]: