for_fut_backward

May 11, 2020

[1]:
```
import numpy as np
from scipy.optimize import minimize
```

[6]:
```
import pprint
pp= pprint.PrettyPrinter(depth=10)
```

[27]:
```
#question 5 setup
T=11
times= np.arange(T)
def ho_lee_rn(n, heads):
    #definition of Ho-Lee model
    an= 0.015-0.00125*n
    bn= 0.0025
    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))
```

[28]:
```
def compute_b1011(all_rn):
    prices= {T:[1], T-1: 1/(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:
            #no discounting or intermediate payments
            tprices[k]= (.5*(prices[t+1][k+1] + prices[t+1][k]))
        prices[t]= tprices
    print("time 10 price of ZCB maturity 11 : ")
    pp.pprint(prices)
    return prices[0][0]
```

```python
def compute_b0T(all_rn, T):
    #typical backward induction
    prices= {T:[1], T-1: 1/(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]))
        prices[t]= tprices
    print("time 0 price of ZCB maturity {} :".format(T))
    pp.pprint(prices)
    return prices[0][0]

def get_stats(all_rn):
    #answer to question 5 on hw
    r10= all_rn[10]
    b010= compute_b0T(all_rn, 10)
    b011= compute_b0T(all_rn, 11)
    ans_a= b010/b011-1
    ans_b= np.mean(r10)
    ans_d= np.mean(100*(1-4*r10))
    ans_c= np.mean(100*compute_b1011(all_rn))
    return ans_a, ans_b, ans_c, ans_d
```

[29]:
```python
get_stats(all_rn)
```

```
time 0 price of ZCB maturity 10 :
{0: array([0.8618829]),
 1: array([0.88450733, 0.86511496]),
 2: array([0.90551437, 0.88782423, 0.87052193]),
 3: array([0.92475655, 0.90891005, 0.89337313, 0.878139  ]),
 4: array([0.94209574, 0.92822439, 0.91459074, 0.90119014, 0.88801806]),
 5: array([0.9574048 , 0.9456286 , 0.93402579, 0.92259341, 0.91132853,
       0.90022831]),
 6: array([0.97056911, 0.96099506, 0.95153878, 0.94219852, 0.93297259,
       0.9238593 , 0.91485702]),
 7: array([0.98148801, 0.97420875, 0.96700128, 0.95986474, 0.95279825,
       0.94580096, 0.93887201, 0.93201059]),
 8: array([0.99007603, 0.98516859, 0.98029755, 0.97546254, 0.97066322,
       0.96589923, 0.96117022, 0.95647586, 0.95181581]),
 9: array([0.99626401, 0.99378882, 0.9913259 , 0.98887515, 0.9864365 ,
       0.98400984, 0.98159509, 0.97919217, 0.97680098, 0.97442144]),
 10: [1]}
time 0 price of ZCB maturity 11 :
```

```
{0: array([0.84921656]),
 1: array([0.87257001, 0.8513396 ]),
 2: array([0.89438426, 0.87475144, 0.85559629]),
 3: array([0.91450791, 0.89662022, 0.87912519, 0.86201327]),
 4: array([0.93279806, 0.91679418, 0.90110332, 0.88571863, 0.8706334 ]),
 5: array([0.94912203, 0.93513006, 0.92137815, 0.9078616 , 0.89457582,
       0.88151632]),
 6: array([0.96335886, 0.95149484, 0.93980571, 0.92828848, 0.91694021,
       0.90575802, 0.89473906]),
 7: array([0.97540085, 0.96576726, 0.95625231, 0.94685425, 0.93757137,
       0.92840197, 0.91934439, 0.910397  ]),
 8: array([0.98515486, 0.97783935, 0.97059609, 0.9634242 , 0.9563228 ,
       0.94929101, 0.942328  , 0.93543291, 0.92860494]),
 9: array([0.99254352, 0.98761774, 0.98272855, 0.97787557, 0.97305844,
       0.96827683, 0.96353038, 0.95881874, 0.95414157, 0.94949855]),
 10: array([0.99750623, 0.99502488, 0.99255583, 0.99009901, 0.98765432,
       0.98522167, 0.98280098, 0.98039216, 0.97799511, 0.97560976,
       0.97323601]),
 11: [1]}
time 10 price of ZCB maturity 11 :
{0: array([0.98523662]),
 1: array([0.98645   , 0.98402324]),
 2: array([0.98766636, 0.98523363, 0.98281285]),
 3: array([0.98888573, 0.986447  , 0.98402026, 0.98160544]),
 4: array([0.99010811, 0.98766335, 0.98523064, 0.98280988, 0.98040099]),
 5: array([0.99133351, 0.98888271, 0.986444  , 0.98401728, 0.98160248,
       0.9791995 ]),
 6: array([0.99256194, 0.99010508, 0.98766034, 0.98522765, 0.98280692,
       0.98039805, 0.97800096]),
 7: array([0.99379342, 0.99133047, 0.98887969, 0.986441  , 0.98401431,
       0.98159953, 0.97919657, 0.97680535]),
 8: array([0.99502795, 0.99255889, 0.99010204, 0.98765733, 0.98522466,
       0.98280395, 0.9803951 , 0.97799803, 0.97561266]),
 9: array([0.99626556, 0.99379035, 0.99132742, 0.98887667, 0.986438  ,
       0.98401133, 0.98159657, 0.97919363, 0.97680243, 0.97442288]),
 10: array([0.99750623, 0.99502488, 0.99255583, 0.99009901, 0.98765432,
       0.98522167, 0.98280098, 0.98039216, 0.97799511, 0.97560976,
       0.97323601]),
 11: [1]}
```

[29]:
```
(0.014915329044920433, 0.014999999999999998, 98.52366179644693, 94.0)
```

[30]:
```python
#Question 6: interest rate tree setup
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
all_rn= get_all_rn(np.arange(T))
```

[31]:
```
epsilon_V= np.arange(1, 10)
epsilon_W= np.array([6,7,8,9])
def callable_backward(q, call_dates):
    F= 1000
    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:
                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[0]-F)**2

def find_parq():
    res_v= minimize(callable_backward, x0=0.06, args=(epsilon_V,))
    res_w= minimize(callable_backward, x0=0.06, args=(epsilon_W,))
    return res_v.x[0], res_w.x[0]
```

[32]:
```
find_parq()
```

[32]:
```
(0.06640530519823765, 0.0618809080269416)
```

[23]:
```
#Question 7
def verify_formula(r0=0.1, m=10, alpha=1.02, beta=.98):
    product= (1/beta + 1/alpha)*.5
    return 100/(1+r0)*(product)**m
```

[24]:
```
verify_formula()
```

[24]:
```
91.27352855439308
```

[ ]:
```

```