HW: Libor Market Model Implementation

금융공학 20173855
박완배

다음과 같이 주어진 정보로 Libor Market Model 을 이용, ratchet cap 과 sticky cap 의 각 caplet 가격을 몬테카를로 시뮬레이션으로 구하였다. 난수 생성에는 antithetic variable technique 를 이용하였다.

Spot rate: 5% flat

Volatility data

Year, k	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
σ_k(%)	15.50	18.25	17.91	17.74	17.27	16.79	16.30	16.01	15.76	15.54

시뮬레이션을 이용하여 구한 결과는 아래와 같다.

```
==========Ratchet Cap==========
1       0.197
2       0.208
3       0.204
4       0.195
5       0.189
6       0.184
7       0.174
8       0.168
9       0.162
10      0.152
==========Sticky Cap==========
1       0.197
2       0.337
3       0.415
4       0.46
5       0.487
6       0.503
7       0.505
8       0.502
9       0.496
10      0.484
==============================
Computation Time: 0.837 sec
```

```
<Python Code>
import numpy as np
import time

class MarketVariable:
   def __init__(self, spot, sigma):
      self.spot = spot
      self.sigma = sigma

class Cap:
   def __init__(self, notional, strike):
      self.notional = notional
      self.strike = strike

   def getPayoff(self, spot):
      return np.where(spot > self.strike,
                spot - self.strike, 0) * self.notional

class RatchetCap(Cap):
   def __init__(self, notional, spread, maturity, dt):
      self.notional = notional
      self.spread = spread
      self.maturity = maturity
      self.dt = dt

   def getStrike(self, spot):
      self.strike = spot + self.spread

   def  getCapletPrice(self,  spotProcess):    #Under  Rolling-Forward
Measure
      df = 0
      result = []
      numLoop = int(self.maturity / self.dt)   #Number of loop
      for i in range(numLoop):
         df = df + spotProcess[i+1]
         self.getStrike(spotProcess[i])
         payoff = self.getPayoff(spotProcess[i+1])
         dpoff = payoff * np.exp(-df)
         price = dpoff.mean()
         result.append(price)
      return result

class StickyCap(Cap):
   def __init__(self, notional, spread, maturity, dt):
      self.notional = notional
      self.spread = spread
      self.maturity = maturity
      self.dt = dt
      self.strike = np.array([False])

   def getStrike(self, spot):
      if self.strike.all() == False:
         self.strike = spot + self.spread
      else:
         self.strike = np.where(spot > self.strike,
                      self.strike, spot) + self.spread
```

```
    def getCapletPrice(self, spotProcess):
        df = 0
        self.getStrike(spotProcess[0])
        numLoop = int(self.maturity / self.dt)
        result = []
        for i in range(numLoop):
            df = df + spotProcess[i+1]
            payoff = self.getPayoff(spot[i+1])
            dpoff = payoff * np.exp(-df)
            price = dpoff.mean()
            result.append(price)
            self.getStrike(spot[i+1])
        return result

def getForwardRate(spot, t1, t2):
    return (spot[t2-1] * t2 - spot[t1-1] * t1) / (t2 - t1)

def forwardVol(sigma, t, dt, k):
    if k == 0:
        return sigma[k]
    else:
        psum = 0
        for i in range(1, k+1):
            psum += forwardVol(sigma, t, dt, k - i) ** 2 * dt
        return np.sqrt((sigma[k]**2 * t[k] - psum) / dt)


def getDrift(forward, lamda, j, dt):        # drift 계산
    lam = lamda[:len(lamda) - j]
    #dt*Fi*lambda[i-j-1] / (1+dt*Fi)
    drift1 = forward * lam * dt / (1 + dt * forward)
    drift = np.zeros([len(drift1), len(drift1[0])])  # Drift matrix
    for i in range(len(drift[0])):
        drift[:,i] = (drift1[:,:i+1]*lamda[i]).sum(axis = 1) \
            - (0.5 * lamda[i] ** 2)
    return drift

def getLambda(sigma, t, dt, maturity):      #Volatility 로부터 Lambda 계산
함수
    lamda = np.zeros(maturity)
    for k in range(maturity):
        lamda[k] = forwardVol(sigma, t, dt, k)
    return lamda

def spotToBond(spot, dt):
    bond = []
    for i in range(len(spot)):
        bond.append(np.exp(-spot[i] * dt))
    return bond

#Libor Market Model Simulation Function
def LMMSimulation(spot, forward, simnum):
    #F[k][t_j] generate
    frate = []
    for i in range(10):
```

```
        ar = np.zeros([simnum, i+2])
        ar[:,0] = forward[:,i]
        frate.append(ar)

    #Simulation
    spot = [np.zeros(simnum) + spot[0]]
    for i in range(10):

        #Get Drift
        drift = getDrift(forward, lamda, i, dt)

        #Generate Random Number
        eps = np.random.randn(int(0.5 * simnum))
        eps = np.r_[eps, -eps]     # Antithetic Variable Technique

        for j in range(10-i):
            diff = lamda[j] * eps * np.sqrt(dt)
            frate[j][:,i+1] = frate[j][:,i] * np.exp(drift[:,j] * dt +
diff)

        spot.append(frate.pop(0)[:,-1])
        forward = np.zeros([simnum, 9-i])
        for k in range(len(forward[0])):
            forward[:,k] = frate[k][:,i+1]
    return spot

if __name__ == '__main__':
    #Variable Initialization
    start = time.time()
    spot = np.array([0.05] * 11)
    sigma = np.array([0.1550, 0.1825, 0.1791, 0.1774, 0.1727, 0.1679,
0.1630,
                      0.1601, 0.1576, 0.1554])
    t = np.arange(1, 11)
    dt = 1
    maturity = 10
    simnum = 100000

    #Calculate Forward Rate from Spot Rate
    forward = np.zeros([simnum, 10]) \
        + np.array([getForwardRate(spot, i, i+1) for i in range(1, 11)])

    #Calculate Lambda
    lamda = getLambda(sigma, t, dt, maturity)

    #Generate Future Spot Rate
    spot = LMMSimulation(spot, forward, simnum)

    #Ratchet Cap
    rCap = RatchetCap(100, 0.0025, 10, 1)
    rprice = rCap.getCapletPrice(spot)

    #Sticky Cap
    sCap = StickyCap(100, 0.0025, 10, 1)
    sprice = sCap.getCapletPrice(spot)
```

```
#Print Result
print("="*10 + "Ratchet Cap" + "="*10)
for i in range(len(rprice)):
   print(i+1, "\t", round(rprice[i], 3))

print("="*10 + "Sticky Cap" + "="*10)
for i in range(len(sprice)):
   print(i+1, "\t", round(sprice[i], 3))
print("="*30)
print("Computation Time: %.3f sec" %(time.time() - start))
```