Fixed_Income_stringModel_YiTaoHu

YiTao Hu

26/02/2020

```r
#first we import data and some useful libraries
library(derivmkts)
library(LSMonteCarlo)
library(readxl)
library(readr)
D_0_T=read_excel("Homework 7 pfilea.xlsx", col_names = FALSE)
D_0_T=D_0_T$...1
sigmas=read_excel("Homework 7 sigma.xlsx",      col_names = FALSE)
sigmas=sigmas$...1
Corr_mat=read_csv("Homework 7 corrin.csv",
    col_names = FALSE)
U=read_csv("Homework 7 corchol.csv",
     col_names = FALSE)
U=as.matrix(U)
```

Q1

Solve for r* and compute recusive scheme for dD(t,T)

Recall dynamics of the bond price under string model

$$dD(t,T) = r_t^* D(t,T)dt + \sigma(T-t)D(t,T)dW(t,T)$$

where $W(t,T)$ are correlated Brownian Motion at time t. in other words, $d\vec{W}_t$ is a multivariate normal variavle $MVN(0,\Sigma)$ where Σ is the var-cov matrix of MVN

First,we define a function to compute $d\vec{W}_t$.

$$d\vec{W}_t = \vec{W_{t+dt}} - \vec{W}_t = \sqrt{dt}U^T\vec{Z}$$

where U^T is the Cholesky root of Correlation matrix and $\vec{Z}$ is a standard MVN

```r
compute_dWt=function(cho_corr,dt){
  dWt=sqrt(dt)*cho_corr%*%rnorm(nrow(as.matrix(cho_corr)))
  return(dWt)
}
dWt=compute_dWt(cho_corr = U[-20,-20],dt = 0.5)
```

Then we need to define a function to compute r*

```r
compt_rf=function(D_0.5){
  rf=2*(1/D_0.5-1)
  return(rf)
}
rf=compt_rf(D_0.5 = D_0_T[1])
```

From the formula above, we can get our extrapolation scheme:

$$D(t+dt,T) = D(t,T) + dD(t,T) = D(t,T) + r_t^* D(t,T)dt + \sigma(T-t)D(t,T)dW(t,T)$$

```r
#define a function to compute D(t+dt,T)
comput_Dtforward=function(DT,D_0.5,sigmas,dt,cho_corr){
  rf=compt_rf(D_0.5 = D_0.5)
  dWt=compute_dWt(cho_corr = cho_corr,dt = dt )
  D_tplus=DT+rf*DT*dt+sigmas*DT*dWt
  return(D_tplus)
}
```

Compute one D(t,T) trajectory

Now we can compute one particular trajectories of the whole DT function series using the recusive scheme defined above.

```r
#define a function to compute one particular senerio
cho_corr=U
senoro_sim=function(D0T,sigmas,cho_corr){
  #initialize the scenario matrix
  scenario_df=data.frame(matrix(0,nrow = 20,ncol = 20))
  colnames(scenario_df)=seq(0,9.5,0.5)
  scenario_df[,1]=D0T
  #simulate the trajectories
  for (t in 2:20){
    scenario_df[t:20,t]=comput_Dtforward(DT = scenario_df[t:20,(t-1)],D_0.5 = scenario_df[(t-1),(t-1)],
  }
  return(as.matrix(scenario_df))
}
```

Then, we can perform 10,000 scenarios simulation

```r
batch_sim=function(D0T,sigmas,scenario_num=10000)
  {
  #initialize the simulation
  sim_array=array(data = 0,dim = c(20,20,scenario_num))
  #because paralell computing exceeds the CPU limit, here we use for loop to do simulation
  for(i in 1:scenario_num){
    sim_array[,,i]=senoro_sim(D0T = D0T,sigmas = sigmas,cho_corr = cho_corr)
  }
  return(sim_array)
}
sim_out=batch_sim(D0T=D_0_T[1:20],sigmas=sigmas[1:20],scenario_num=10000)
#Then,we can look at the structure's average time decay across different scenerios
Ave_sim_decay=apply(sim_out, MARGIN =c(1,2), FUN = mean)
Ave_sim_decay
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
##  [1,] 0.9724765 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [2,] 0.9445693 0.9712539 0.0000000 0.0000000 0.0000000 0.0000000
##  [3,] 0.9163238 0.9419094 0.9683550 0.0000000 0.0000000 0.0000000
##  [4,] 0.8879337 0.9129268 0.9380022 0.9682574 0.0000000 0.0000000
##  [5,] 0.8597406 0.8841044 0.9079895 0.9368499 0.9678044 0.0000000
##  [6,] 0.8320438 0.8555911 0.8781353 0.9056011 0.9351236 0.9649078
##  [7,] 0.8050770 0.8280785 0.8497309 0.8761707 0.9042466 0.9323576
##  [8,] 0.7789041 0.8014280 0.8225616 0.8483234 0.8756504 0.9023615
##  [9,] 0.7534803 0.7755353 0.7964018 0.8213936 0.8482873 0.8739277
```

```
## [10,] 0.7287341 0.7501257 0.7706793 0.7952077 0.8210304 0.8460108
## [11,] 0.7045800 0.7253565 0.7455341 0.7698610 0.7949843 0.8193463
## [12,] 0.6810105 0.7009283 0.7204630 0.7441906 0.7684779 0.7928365
## [13,] 0.6580423 0.6773426 0.6965413 0.7195130 0.7430448 0.7669286
## [14,] 0.6356925 0.6544195 0.6733357 0.6958156 0.7185876 0.7418989
## [15,] 0.6139770 0.6317398 0.6502574 0.6722225 0.6946111 0.7175845
## [16,] 0.5929047 0.6100173 0.6277786 0.6491745 0.6708213 0.6932375
## [17,] 0.5724819 0.5892066 0.6065220 0.6272320 0.6482353 0.6698426
## [18,] 0.5527129 0.5688354 0.5857380 0.6057865 0.6263058 0.6472387
## [19,] 0.5336010 0.5491449 0.5655853 0.5849698 0.6049369 0.6255486
## [20,] 0.5151482 0.5301801 0.5460852 0.5647864 0.5839895 0.6039391
##            [,7]      [,8]      [,9]     [,10]     [,11]     [,12]
##  [1,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [2,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [3,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [4,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [5,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [6,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [7,] 0.9639989 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [8,] 0.9319177 0.9651867 0.0000000 0.0000000 0.0000000 0.0000000
##  [9,] 0.9021569 0.9334680 0.9657629 0.0000000 0.0000000 0.0000000
## [10,] 0.8728890 0.9025762 0.9329530 0.9653307 0.0000000 0.0000000
## [11,] 0.8455162 0.8743743 0.9032823 0.9341089 0.9669526 0.0000000
## [12,] 0.8184723 0.8466357 0.8745201 0.9037902 0.9351363 0.9656619
## [13,] 0.7920704 0.8195286 0.8463470 0.8743926 0.9042283 0.9333104
## [14,] 0.7665058 0.7935410 0.8198453 0.8472511 0.8754854 0.9031534
## [15,] 0.7415176 0.7679419 0.7935202 0.8207384 0.8483112 0.8749827
## [16,] 0.7168348 0.7425823 0.7678228 0.7944043 0.8214006 0.8474263
## [17,] 0.6929343 0.7181819 0.7429845 0.7689031 0.7953372 0.8205800
## [18,] 0.6697136 0.6943357 0.7186940 0.7438324 0.7700582 0.7944728
## [19,] 0.6478488 0.6719607 0.6958492 0.7204672 0.7460658 0.7701043
## [20,] 0.6259209 0.6494724 0.6729088 0.6971771 0.7225264 0.7459760
##           [,13]     [,14]     [,15]     [,16]     [,17]     [,18]
##  [1,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [2,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [3,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [4,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [5,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [6,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [7,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [8,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [9,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [10,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [11,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [12,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [13,] 0.9649561 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [14,] 0.9330958 0.9650936 0.0000000 0.0000000 0.0000000 0.0000000
## [15,] 0.9034414 0.9337924 0.9658776 0.0000000 0.0000000 0.0000000
## [16,] 0.8749312 0.9039198 0.9344260 0.9664321 0.0000000 0.0000000
## [17,] 0.8472408 0.8750003 0.9037568 0.9339040 0.9647448 0.0000000
## [18,] 0.8208452 0.8477965 0.8756669 0.9039230 0.9330126 0.9656223
## [19,] 0.7957961 0.8223196 0.8493682 0.8765276 0.9040901 0.9353364
## [20,] 0.7711382 0.7970150 0.8229523 0.8495633 0.8764451 0.9064663
##           [,19]     [,20]
```

```
##  [1,] 0.0000000 0.00000
##  [2,] 0.0000000 0.00000
##  [3,] 0.0000000 0.00000
##  [4,] 0.0000000 0.00000
##  [5,] 0.0000000 0.00000
##  [6,] 0.0000000 0.00000
##  [7,] 0.0000000 0.00000
##  [8,] 0.0000000 0.00000
##  [9,] 0.0000000 0.00000
## [10,] 0.0000000 0.00000
## [11,] 0.0000000 0.00000
## [12,] 0.0000000 0.00000
## [13,] 0.0000000 0.00000
## [14,] 0.0000000 0.00000
## [15,] 0.0000000 0.00000
## [16,] 0.0000000 0.00000
## [17,] 0.0000000 0.00000
## [18,] 0.0000000 0.00000
## [19,] 0.9679634 0.00000
## [20,] 0.9376790 0.96744
```

2

Here, we need to compute the forward par rates five years ahead from the initial term structure.

$$C_{forward} = 2[\frac{100D(N) - 100D(N+M)}{\sum_{i=1}^{2M} D(N+i/2)}]$$

```
Forward_DTs=D_0_T[11:21]
Forward_par=2*(100*Forward_DTs[1]-100*Forward_DTs[2:11])/cumsum(Forward_DTs[2:11])
Forward_par
```

```
##  [1] 6.921920 6.950839 6.976849 6.999824 7.020012 7.037521 7.052398
##  [8] 7.064631 7.074196 7.081173
```

3

We first need to define a function to map discount factor to par rates.

```
#define a function to get the Discount factor decay for T=0.5:5
get_DT_decay=function(sim_mat)
{
  DTdecay=matrix(data = 0,nrow = 11,ncol = 10)
  Trim_mat=sim_mat
  for (T in 1:10){
    DTdecay[,T]=diag(Trim_mat)[1:11]
    Trim_mat=Trim_mat[-1,-ncol(Trim_mat)]
  }
  DTdecay=t(DTdecay)
  rownames(DTdecay)=seq(0.5,5,0.5)
  colnames(DTdecay)=seq(0,5,0.5)
  return(DTdecay)
}
```

Compute the par rates at each time step

```
compt_par=function(DT){
  ParRates=2*(100-100*DT)/cumsum(DT)/100
  return(ParRates)
}
#assible the two function above into one
get_par_rates=function(sim_mat){
  DTs=get_DT_decay(sim_mat)
  Par_decay=apply(DTs, 2, compt_par)
  return(Par_decay)
}
#compute the terminal payoff at each scenarios
get_batch_par=function(sim_asset){
  #initialize the par simulation arrary
  par_array=array(data = 0,dim = c(10,11,dim(sim_asset)[3]))
  #because paralell computing exceeds the CPU limit, here we use for loop to do simulation
  for(i in 1:dim(sim_asset)[3]){
    par_array[,,i]=get_par_rates(sim_asset[,,i])
  }
  return(par_array)
}
#compute all par rates decay based on all simulation
par_rates_batch=get_batch_par(sim_asset = sim_out)
```

Recall the future price of the contract is the risk-neutral expected value of the underlying asset:

$$K = E^Q(B_T)$$

where B_T in this case is the price of the bond to deliver at terminal time T.

First,we need to mapping par rates dynamics to bond price at time T.

```
#compute bond price at terminal time T=5 for each scenerio and each maturity
bond_value=data.frame(matrix(0,nrow = 10,000,ncol = 10))
colnames(bond_value)=seq(0.5,5,0.5)
for (i in 1:dim(par_rates_batch)[3]){
    for (mar in 1:10){
      bond_value[i,mar]=bondpv(coupon = Forward_par[mar],mat = (mar/2),yield =  par_rates_batch[mar,11,i
    }
}
#select out the cheapest bond for each scenerio
B_T_min=apply(bond_value[,seq(2,10,2)], 1, min)
#compute the risk-neutral expected value
K=mean(apply(bond_value[,seq(2,10,2)], 2, mean))
K
```

```
## [1] 100.679
```

Recall the price of any rate-related derivative should be the risk-neutral expected value of discounted final cash flow.

$$Option = E^Q(e^{-\int_0^T Rsd_s}(K - B_{Tcheap}))$$

where both the bond to deliver $B_{Tcheap}$and the short rates Rs are stochastic.

```
#grab the paths of short rates for all timestep
Discount_sim=apply(sim_out, 3, diag)[1:10,]
#multiple the short rate to get total discount factor for each scenoro
Disount_T=apply(Discount_sim, 2, cumprod)[10,]
```

```r
#compute the option price
Opt_val=mean(Disount_T*(K-B_T_min))
Opt_val
```

```
## [1] 6.688859
```