CFRM 520 Homework 3

Frank Fineis

July 17, 2015

Overview

My approach to Homework 3 was to run all of the Monte Carlo simulations at once. I value the cap, floor, swap, fuel-switching, and collar options together in one large (364 x 8) matrix, `valueStor`, 200 times, aggregate each option's payoffs into monthly time-buckets, and then average the bucketed payoffs on a per-month basis. In this way, we only wait for a single Monte Carlo simulation and avoid having to re-simulate power, gas, and/or heating oil prices for every problem.

The main changes from the Demo 9 code are that the two `for` loops are expanded: `valueStor` grows from 5 to 8 columns to accommodate the swap, fuel-switching, and collar options, and `bucketStor` has four times as many rows; its dimension is now $(4N \times 12)$. Besides for the matrix expansions, all of the variables and indexing arrays from Demo 9 the same.

Expanding the storage matrices:

```
#Expanding...
valueStor = matrix(0, nbDays, 8)
colnames(valueStor) = c("Power", "Gas", "HO", "Cap", "Floor", "Swap",
                        "Fuel_Switching", "Collar")
bucketStor = matrix(0,4*N,length(indexStartDates))
```

From here, we can run one single Monte Carlo loop and value each of the 5 options. I'll get into the specifics of valuing each type of option in the following sections.

Once a single Monte Carlo simulation is finished (there are 200 of them), we need to store the cumulative monthly payoffs into the "time buckets." Considering that `bucketStor` now has four times as many rows, we just index by intervals of 200 to correctly store the location of each bucketed cashflow simulation:

```
#Bucket cash flows: (1) Swap, (2) Fuel_Switching, (3) Collar individually,
#and then (4) All assets combined
for(k in 1:length(indexStartDates)){
    bucketStor[j,k] = sum(valueStor[indexStartDates[k]:indexEndDates[k],6])
    bucketStor[j+N, k] = sum(valueStor[indexStartDates[k]:indexEndDates[k],7])
    bucketStor[j+(2*N), k] = sum(valueStor[indexStartDates[k]:indexEndDates[k],8])
    bucketStor[j+(3*N), k] = sum(valueStor[indexStartDates[k]:indexEndDates[k],4:8])
}
```

For each asset, we then take the average of the monthly cashflows in order estimate their discounted payoffs over time, and we save the results. Since the monthly payoff simulations are stored column-wise, we use the `colMeans` function over the appropriate rows of `bucketStore`, like this:

```
#Store averaged monthly cashflows
Prob1 = t(as.data.frame(colMeans(bucketStor[1:N,])))
Prob2 = t(as.data.frame(colMeans(bucketStor[(N+1):(2*N),])))
```

```r
Prob3 = t(as.data.frame(colMeans(bucketStor[((2*N)+1):(3*N),])))
Prob4 = t(as.data.frame(colMeans(bucketStor[((3*N)+1):(4*N),])))
results = list(Prob1 = Prob1, Prob2 = Prob2, Prob3 = Prob3, Prob4 = Prob4)
for (k in 1:4){rownames(results[[k]]) <- "Cashflows"
    colnames(results[[k]]) <- unique(months(storDates))}
```

Problem 1: Swap options

A swap is basically a forward, in that a counterparty agrees to buy a predetermined fixed tariff, `K`, at certain dates in the future. The payoff 'caplet' of a single day's swap on power price is

$$(P - K)$$

If the power price is above the fixed tariff, then the counterparty will be in the money, but the opposite is true when $K > P$.

Great, now the first three columns of `valueStore` will contain the simulated power, gas, and heating oil prices, in that order, and their evolution will follow a joint stochastic, mean-reverting process. The fourth column of `valueStore` contains the daily payoffs of a swap option, which is valued according to $\max(0, P_t - \$80)$, where P_t is the spot price of energy. Note that we present-value each option's payoff to January 1, 2009-dollars with the discount factor of $e^{(-r \cdot t_{elapsed})}$.

We can value each swap in R like this:

```r
#futures payoff: make money when spot > strike, lose o/w
valueStor[i+1,"Swap"] = (valueStor[i+1,"Power"] -KSwap)
    *exp(-r*(as.numeric(storDates[i+1]-valDate))/365)
```

The averaged discounted monthly payoffs for the swap options:

```r
print(results[[1]])
```

```
##            January February     March     April      May    June     July
## Cashflows 134.4768 13.25975 -11.73398 -62.87715 49.90622 223.481 430.0344
##             August September  October November  December
## Cashflows 313.5268  75.77872 -11.1274 -42.5917 -20.14288
```

Problem 2: Fuel-switching options

To value the discounted payoffs of a natural gas/heating oil fuel-switching option, recall from the thermal assets session that the payoff on a dual fuel generator is

$$Payoff = Q \times \max(P - \min(HR_1 \cdot Gas, HR_2 \cdot Oil) - (VOM + Emissions), 0)$$

Great, we've already modeled the heating oil prices. Next, we need to change those prices from dollars per barrel (bbl) to dollars per MMBTU, because natural gas prices are in dollars per MMBTU. To do this, I've just multiplied the heating oil prices by `scaleFactor`. The plant operator will choose whichever is cheaper on a per-MMBTU basis, natural gas or heating oil. In order to compare dollars per MMBTU to dollars per MWh (power prices are in $/MMBTU), we have to multiply the minimum of the natural gas/heating oil comparison by the `heatRate` in order to be able to convert from fuel used to power generated. Thus, the discounted payoff of this option, per day, is given by the following R code:

```
#Choose cheaper fuel, on per MMBTU basis...
valueStor[i+1,"Fuel_Switching"] = max(0, valueStor[i+1,"Power"] -
    (heatRate*min(valueStor[1,"Gas"], scaleFactor*valueStor[i+1,"HO"]))-K)*
    exp(-r*(as.numeric(storDates[i+1]-valDate))/365)
```

where `K = VOM + Emissions` is equal to \$11.50.

The averaged discounted monthly payoffs for the fuel-switching options:

```
print(results[[2]])
```

```
##           January February    March    April      May     June     July
## Cashflows 44.9167 288.7577 739.1212 640.6542 523.9499 427.2126 443.2564
##             August September  October November December
## Cashflows 366.0485   188.489 82.22458 32.68718 45.33137
```

Problem 3: Collar options

From session 1, recall that a collar is 'a portfolio of a long position in a call (cap) and a short position in a put (floor).' There is a subtle difference between valuing the collar and valuing the caps and floors that we've already done in Demo 9; the cap is insurance only in the case where power price is higher than some fixed cap-strike level, and the floor is insurance only in the case where power prices dip below some fixed floor-strike level. That is, a cap's payoff only comes when $P_t > K_{cap}$, and similarly a floor's payoff only comes when $P_t < K_{floor}$.

The collar is a dual cap and put, effective at the same time: if $P_t > K_{cap}$ *or* $P_t < K_{floor}$, the collar pays off. This is easily implemented in R:

```
#collar is long a cap (max(0, S-K)) and short a floor (-max(0, K-S))
if(valueStor[i+1,"Power"] < KFloor){
    valueStor[i+1, "Collar"] = -max(0,KFloor- valueStor[i+1,"Power"])*
      exp(-r*(as.numeric(storDates[i+1]-valDate))/365)}
if(valueStor[i+1, "Power"] > KCap){
    valueStor[i+1, "Collar"] = max(0,valueStor[i+1,"Power"]-KCap)*
      exp(-r*(as.numeric(storDates[i+1]-valDate))/365)}
```

The averaged discounted monthly payoffs for the collar options:

```
print(results[[3]])
```

```
##            January February    March     April      May     June     July
## Cashflows 158.4447 51.65683 55.17083 -6.877456 111.5663 302.5436 447.0853
##             August September  October  November  December
## Cashflows 346.8006  125.9165 37.93812 -2.356564 -13.14847
```

Problem 4: Portfolio CFaR

The main idea here is that we'll add up all 5 of the asset's payoffs during the `bucketing` aggregation process instead of just estimating the monthly cashflows of an individual asset. This is easy enough, see the code above involving `bucketStor` - we just sum together the 5 payoffs each month.

Again, we average each of the portfolio's 200 monthly cashflows with the `colMeans` function, and we can also get an estimate of the standard error for each month's cashflow:

```r
# Monthly portfolio payoffs:
print(results[[4]])
```

```
##             January February    March    April      May     June     July
## Cashflows 421.1295 388.1797 808.3526 558.3807 755.8032 1137.363 1630.717
##              August September October  November December
## Cashflows 1275.517  465.9775 128.968 -13.11923 31.49503
```

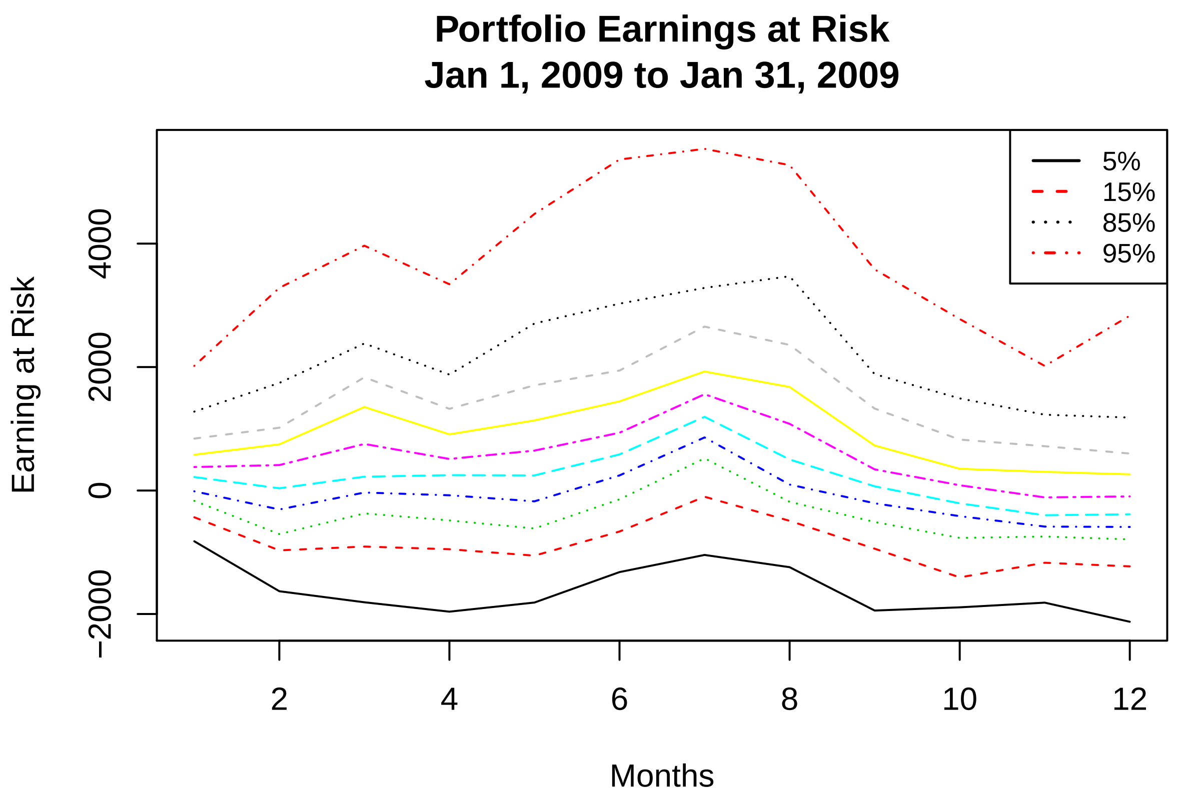
```r
# Standard error per time bucket:
se = apply(bucketStor[((3*N)+1):(4*N),], MARGIN = 2, FUN = sd)/sqrt(N)
cat("Std. Errors:", se)
```

```
## Std. Errors: 69.39328 106.9351 136.8186 121.5552 135.5826 140.4593 133.5185 149.3471 112.5348 103.25
```

Finally, we can get the empirical quantiles for each of the portfolio's monthly cashflows just as we did in Demo 9, but now we are only asked for the quantiles in 10% intervals. Plotting the monthly quantiles, we can see the portolio's estimated monthly cashflow at risk:

```r
percentile = matrix(0,10,length(indexEndDates)) #storage matrix
for(t in 1:length(indexEndDates)){
  percentile[,t] = quantile(bucketStor[((3*N)+1):(4*N),t],c(seq(0.05,0.95,0.10)))
}

#plot 5% empirical quantile:
plot(c(percentile[1,]),type="l", ylab="Earning at Risk", xlab="Months",
     ylim=c(min(percentile),max(percentile)),
     main = "Portfolio Earnings at Risk\nJan 1, 2009 to Jan 31, 2009")
#plot all other empirical quantiles:
for(v in 2:10){lines(percentile[v,],col=v,lwd=1, lty=v)}
legend("topright",legend=c("5%","15%","85%","95%"),
       col = c(1,2,9,10), lwd=c(1.5,1.5,1.5,1.5), lty=c(1,2,9,10), cex=0.8)
```


Portfolio Earnings at Risk
Jan 1, 2009 to Jan 31, 2009
Earning at Risk
Months
5%
15%
85%
95%
−2000
0
2000
4000
2
4
6
8
10
12