Module 7.2: Static Risk Management U. S. Treasuries
R Commentary

See module *Ch 8.2 SRM US Treasuries*.

We comment on selections from three test programs along with associated other files.
SRM Traditional UST Test.R (Selected Excerpts and Output)
This program illustrates three bonds with identical characteristics except for coupon amount. The base bond is an actual bond with roughly 10 years to maturity and a stated coupon of 2.25%. We compare its sensitivity to a 1.125% coupon bond and a 3.375% coupon bond. This program follows closely with Module 4.1 with the addition of selected traditional duration and convexity measures. The core source code is the following for loop where the methods for various calculations are called.

```
for(i in 1:NumberOfObservations){
  Y[i] <- LowerBound + (i-1)*StepSize
  BONDInputData$YieldToMaturity = Y[i]
  BONDInputData$CouponRate = originalCR
  OBV[i] = BondValue(BONDInputData) #- AccruedInterest(BONDInputData)
  ODuration[i] = Duration(BONDInputData)
  OConvexity[i] = Convexity(BONDInputData)
  # High Coupon
  BONDInputData$CouponRate = HCoupon
  HBV[i] = BondValue(BONDInputData) #- AccruedInterest(BONDInputData)
  HDuration[i] = Duration(BONDInputData)
  HConvexity[i] = Convexity(BONDInputData)
  # Low Coupon
  BONDInputData$CouponRate = LCoupon
  LBV[i] = BondValue(BONDInputData) #- AccruedInterest(BONDInputData)
  LDuration[i] = Duration(BONDInputData)
  LConvexity[i] = Convexity(BONDInputData)
  BONDInputData$CouponRate = originalCR
}
```

Note that the bond value could easily be adjusted to hold only the quoted bond price by extracting the accrued interest (currently commented out). Standard duration and convexity values are estimated based on their analytic expressions given in this module.

```
#
# Duration: Macaulay duration
#
Duration = function(B){
  with(B,{
    DV = 0.0
    RemainingCoupons = CouponsRemaining(B)
    ElapsedTime = FractionElapsed(B)
    for(i in 1:RemainingCoupons){
      DV = DV + (i - ElapsedTime)*( (CouponRate/(Frequency*100.0))*Par ) /
        ((1.0 + (YieldToMaturity/(Frequency*100.0)))^(i + 1 - ElapsedTime))
    }
    DV = DV + ((RemainingCoupons - ElapsedTime) * Par) /
      ((1.0 + (YieldToMaturity /
      (Frequency*100.0)))^(RemainingCoupons + 1 - ElapsedTime))
    DV = DV / (Frequency*BondValue(B))
    return( DV )
  })
}
#
# Convexity: Standard convexity
#
Convexity = function(B){
  with(B,{
    Convexity = 0.0
    RemainingCoupons = CouponsRemaining(B)
    ElapsedTime = FractionElapsed(B)
```

```
    for(i in 1:RemainingCoupons){
      Convexity = Convexity + ( (i + 1 - ElapsedTime)*(i - ElapsedTime) *
        ( (CouponRate/(Frequency*100.0))*Par) ) /
        ((1.0 + (YieldToMaturity/(Frequency*100.0)))^(i + 2 - ElapsedTime))
    }
    Convexity = Convexity + ((RemainingCoupons + 1 - ElapsedTime) *
      (RemainingCoupons - ElapsedTime) * Par) /
      ((1.0 + (YieldToMaturity /
      (Frequency*100.0)))^(RemainingCoupons + 2 - ElapsedTime))
    Convexity = Convexity/( (Frequency^2) * BondValue(B))
    return( Convexity )
  })
}
```

We now turn to more advanced applications by tying UST bond values to the CMT curve fitted with the LSC model. We first explore a dataset comprising all traded UST notes and bonds.

SRM UST Book Spreads Over CMT Test.R (Selected Excerpts and Output)

In this test program, we explore the entire set of UST notes and bonds as provided by the *Wall Street Journal*. The data was taken manually from the Treasury markets data page and pasted into a spreadsheet. Prior to analyzing the effective duration and effective convexity of every bond, we select one to repeat the valuation analysis as review. The snippet of code presented here is just setting up the analysis.

```
#
# Fixed Parameters
#
inputFrequency <- 2
inputPar <- 1000000.0
inputChangeInYTM <- 0.01 # Effective duration and convexity
RoughMaturity <- 10 # Years
NFactors <- 4 # Number of factors including Level, 8 or less
NBaseCurve <- 30 # Potential observation for every year for 30 years
# Plot range information
FixRange <- FALSE # For plots
FRMax <- 3.1 # Plot bounds if fixed
FRMin <- 2.4
# Input files for U. S. Treasury bond and CMT rates
USTFileName <- 'UST20200619.xlsx'
CMTFileName <- 'CMT20200619.xlsx' # Should have same date as UST
mTitle = "UST: June 19, 2020" # Date in graph title
# Downloaded UST data stored with date appended: use for settlement
SettlementDateMonth = 6     # Based on file name
SettlementDateDay = 19 + 2  # Current practice is + 2 days settlement
SettlementDateYear = 2020
source("UST Book Inputs.R") # Access UST book
source("SRM UST Functions.R") # UST functions (semi-annual only)
```

Next we pick a bond with slightly more than 10 years to maturity and replicate selected valuation calculations. We provide a few lines next from the console window.

```
> UST[SelectedBond,] # Console: Selected bond parameters
    COUPON   ASKED ASKED.YIELD JMaturityDate MaturityDate   APrice
260  5.375 148.106       0.665         25978    2/15/2031 148.3281
…
> # Calendar manipulations
> N = CouponsRemaining(BONDInputData)
> # ElapsedOutput contains fraction, JLastDate, JNextDate, and JCurrentDate
> ElapsedOutput = Elapsed(BONDInputData)
> # Number of Total Days
> NTD <- ElapsedOutput$NextDate - ElapsedOutput$LastDate
> # Number of Accrued Days since last semi-annual coupon
> NAD <- ElapsedOutput$Fraction * NTD
> # Fraction of coupon period that has elapsed already
> f <- ElapsedOutput$Fraction
> # Bond maturity, in years
> Mat <- TimeToMaturity(BONDInputData)
```

```
> NAD; NTD; f; N; Mat
[1] 127
[1] 182
[1] 0.6978022
[1] 22
[1] 10.6511
> # Bond value given yield to maturity
> MarketQuotedBondPrice <- inputBondPrice
> MarketValueOfBond <- BondValue(BONDInputData)
> AccruedInterestAmount <- AccruedInterest(BONDInputData)
> ModelQuotedBondPrice <- MarketValueOfBond - AccruedInterestAmount
> MarketValueOfBond; AccruedInterestAmount;
[1] 1502281
[1] 18753.43
> ModelQuotedBondPrice; MarketQuotedBondPrice
[1] 1483528
[1] 1483281
> # Yield to maturity given bond value
> inputBondPrice = MarketQuotedBondPrice #Dollars:Quoted price w/o accrued interest
> BONDInputData$BondPrice <- inputBondPrice
> EstYieldToMaturity = YieldToMaturitySolver(BONDInputData)
> EstYieldToMaturity; inputYieldToMaturity
[1] 0.6669188
[1] 0.665
```

Based on the selected bond, we confirm that the bond quoted value is roughly equivalent to the quoted market price and the estimated yield to maturity is very close to the reported yield to maturity. Remember, in this calculation, there is always some estimation error due to the reported numbers being rounded.

We now seek to fit the LSC model to CMT rates. We deploy a four factor model where the scalar coefficients are the selected bond maturity for the first one (s_1 = 10.65 years) and arbitrarily three years for the second one (s_2 = 3 years). Generally, you want the scalars to be significantly different and focused on the portion of the maturities where maximal fit is important for your purposes. The scalars are set in the CMT Inputs.R file. The following are selected excerpts from the console related to fitting the CMT curve.

```
> # Just quickly check input parameters for DiffCMTRates
> x
[1]  1.47 -1.29  0.00  0.00
> NFactors
[1] 4
> Sc
[1] 10.6511  3.0000
> NBaseCurve
[1] 30
> MarketCMTRates
 [1] 0.18 0.19 0.22   NA 0.33   NA 0.53   NA   NA 0.70   NA
[12]   NA   NA   NA   NA   NA   NA   NA   NA 1.23   NA   NA
[23]   NA   NA   NA   NA   NA   NA   NA 1.47
> # Given coefficients for discount curve based on LSC,
> #  estimate sum squared difference
> Answer <- DiffCMTRates(x, NFactors, Sc, NBaseCurve, MarketCMTRates)
> Answer
[1] 0.381357
```

Clearly, the function `DiffCMTRates` does not return zero; hence, the initial LSC parameters, `x`, are not the optimal ones.

```
> # optimx R package provides minimization routine to select LSC coefficients
> # to minimize squared differences #, all.methods=TRUE (uses all methods)
> OptOutput <- optimx(par=x, fn=DiffCMTRates, NFac = NFactors, S = Sc,
+   NCMTs = NBaseCurve, MSR = MarketCMTRates,
+   method=c('nlminb'), control=list(save.failures=FALSE, maxit=2500))
> # If 'nlminb' failed, then try a few more optimization routines,
> #  quit when first one produces answer
…
```

The vector y contains the proposed optimal LSC parameters of the four factor model.

```
> y
```

```
[1]  2.253396 -2.021602  0.315353 -1.164027
> # Check to see if sum of squared errors is close to zero
> Answer2 <- DiffCMTRates(y, NFactors, Sc, NBaseCurve, MarketCMTRates)
> Answer2
[1] 0.001759754
```

Here we find the sum of squared errors to be very close to zero. With the fitted LSC model, we are ready to turn to estimating the spot rates and discount rates.

```
SREstimates <- CMTRates(y, NFactors, Sc, NBaseCurve)
SREstimates
# Based on LSC parameters, y, and other inputs, NFactors, Sc, NBaseCurve,
#  provide estimates of fitted discount rates
DREstimates <- DiscountRates(y, NFactors, Sc, NBaseCurve)
DREstimates
```

Based on numerical approximation, effective duration is very easy to code.

```
#
# Effective Duration
#
EffectiveDuration = function(B){
  OriginalYTM = B$YieldToMaturity
  OriginalBV = BondValue(B)
  B$YieldToMaturity = OriginalYTM + B$ChangeInYTM
  UpBV = BondValue(B)
  B$YieldToMaturity = OriginalYTM - B$ChangeInYTM
  DownBV = BondValue(B)
  B$YieldToMaturity = OriginalYTM
  EffDur = (DownBV - UpBV)/(2.0*OriginalBV*(B$ChangeInYTM/100.0))
  return( EffDur )
}
```

Based on numerical approximation, effective convexity is also very easy to code. Note the log transform is used because for some very small changes in yield, machine error can be significant.

```
#
# Effective Convexity
#
EffectiveConvexity = function(B){
  OriginalYTM = B$YieldToMaturity
  OriginalBV = BondValue(B)
  B$YieldToMaturity = OriginalYTM + B$ChangeInYTM
  UpBV = BondValue(B)
  B$YieldToMaturity = OriginalYTM - B$ChangeInYTM
  DownBV = BondValue(B)
  B$YieldToMaturity = OriginalYTM
  Num = log((DownBV - OriginalBV) - (OriginalBV - UpBV))
  Den = -log(OriginalBV) - 2.0*log(B$ChangeInYTM/100.0)
  EffConv = exp(Num + Den)
  return( EffConv )
}
```

The final test program focuses on holding period return decomposition.

SRM UST Book Spreads Over CMT LSC Test.R (Selected Excerpts and Output)

We select various parameters related to the CMT curve and check the fit. As before, estimating static risk measures numerically is straightforward.

```
BaseCurveMDLevel = function(B, LSC){
  OriginalLevel = LSC$Intercept
  OriginalBV = BondValueDF(B, LSC)
  LSC$Intercept = OriginalLevel + B$ChangeInYTM
  UpBV = BondValueDF(B, LSC)
  LSC$Intercept = OriginalLevel - B$ChangeInYTM
  DownBV = BondValueDF(B, LSC)
  LSC$Intercept = OriginalLevel
  BCMDLevel = (DownBV - UpBV)/(2.0*OriginalBV*(B$ChangeInYTM/100.0))
  return( BCMDLevel )
}
```

Convexity requires some care. The log transform seems to reduce machine error.

```
#
# LSC Convexity -- Level
#
BaseCurveCYLevel = function(B, LSC){
  OriginalLevel = LSC$Intercept
  OriginalBV = BondValueDF(B, LSC)
  LSC$Intercept = OriginalLevel + B$ChangeInYTM
  UpBV = BondValueDF(B, LSC)
  LSC$Intercept = OriginalLevel - B$ChangeInYTM
  DownBV = BondValueDF(B, LSC)
  LSC$Intercept = OriginalLevel
  TChange = (DownBV - OriginalBV) - (OriginalBV - UpBV)
  if(TChange > 0.00001){
    Num = log((DownBV - OriginalBV) - (OriginalBV - UpBV))
    Den = -log(OriginalBV) - 2.0*log(B$ChangeInYTM/100.0)
    BCCYLevel = exp(Num + Den)
  } else {
    BCCYLevel = 0.0
  }
  return( BCCYLevel )
}
```

With cross convexity we exploit the relationship with modified duration.

```
#
# LSC Cross Convexity -- Level and Slope
#
BaseCurveCCLevelSlope = function(B, LSC){
  OriginalSlope = LSC$Slope
  LSC$Slope = OriginalSlope + B$ChangeInYTM
  UpV = BaseCurveMDLevel(B, LSC)
  LSC$Slope = OriginalSlope - B$ChangeInYTM
  DownV = BaseCurveMDLevel(B, LSC)
  LSC$Slope = OriginalSlope
  BCCCLevelSlope = (DownV - UpV)/(2.0*(B$ChangeInYTM/100.0))
  return( BCCCLevelSlope )
}
```