Module 4.1: Valuation U.S. Treasuries
R Commentary

See module *Ch 4.1 Valuation US Treasuries*. The UST inputs are contained in *UST Inputs.R* (as well as embedded in various files) and various UST-related functions are contained in *UST Functions.R*. The sample test program that reproduces standard UST bond calculations reported below is in *Valuation UST Test.R*. Other sample data files have naming conventions like *2018-01-17 2.25 2027-11-15 Valuation UST Test.R*.

The Constant Maturity Treasury yields (CMT) inputs are contained in *CMT Inputs.R* and various CMT-related functions are contained in *CMT Functions.R*. The sample test program is in *UST Book Spread Over CMT Test.R*.

Valuation UST Test.R (Selected Excerpts and Output)

The key inputs assuming a $1,000,000 par UST bond settling on January 17, 2018 is given below. Note that the `inputBondPrice` is set to –99 indicating it will be an output.

```
#
# Test Inputs for U.S. Treasury bonds
#
MarketQuotedBondPrice = 975312.50 # Dollars: Quoted price without accrued interest
inputFrequency = 2L         # Coupon frequency per year, 1, 2, 4, or 12
inputCouponRate = 2.25      # Percent
inputPar = 1000000.0        # Currency
inputYieldToMaturity = 2.535235  # Percent
inputYtMType = 'BEY' # BEY-traditional method, CC-continuously compounded
# Dollars: Quoted bond price without accrued interest (stubbed to -99)
inputBondPrice = -99
SettlementDateMonth = 1     # Integer: 1-12
SettlementDateDay = 17      # Integer: 1-31
SettlementDateYear = 2018   # Integer: 1-very high number
MaturityDateMonth = 11      # Integer: 1-12
MaturityDateDay = 15        # Integer: 1-31
MaturityDateYear = 2027     # Integer: 1-very high number
#
# UST functions (semi-annual only)
#
source("UST Functions.R")
BONDInputData <- list(inputFrequency, inputCouponRate, inputPar,
  inputYieldToMaturity, inputYtMType, inputBondPrice,
  SettlementDateMonth, SettlementDateDay, SettlementDateYear,
  MaturityDateMonth, MaturityDateDay, MaturityDateYear)
names(BONDInputData) <- c("Frequency", "CouponRate", "Par",
  "YieldToMaturity", "YtMType", "BondPrice",
  "SettlementDateMonth", "SettlementDateDay", "SettlementDateYear",
  "MaturityDateMonth", "MaturityDateDay", "MaturityDateYear")
# Data frame easier to manage later
BONDInputData <- as.data.frame(BONDInputData)
```

Whenever building applications, it is useful to build a variety of functions that take care of mundane tasks such as day counting and time to maturity.

```
#
# 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)
```


© 2023 Robert Brooks. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
NAD; NTD; f; N; Mat
```
Based on the bond valuation function and the accrued interest function, we compute various results.
```
#
# Bond value given yield to maturity
#
ModelValueOfBond = BondValue(BONDInputData)
AccruedInterestAmount = AccruedInterest(BONDInputData)
ModelValueOfBond; AccruedInterestAmount
ModelQuotedBondPrice = ModelValueOfBond - AccruedInterestAmount
ModelQuotedBondPrice; MarketQuotedBondPrice
PriceError <- ModelQuotedBondPrice - MarketQuotedBondPrice
PriceError
```
To audit our results, we take our derived market-based quoted bond price and compute the implied yield to maturity based on the traditional semi-annual bond equivalent yield.
```
#
# Yield to maturity given bond value
#
inputBondPrice = MarketQuotedBondPrice #Dollars:Quoted price w/o accrued interest
BONDInputData$BondPrice <- inputBondPrice
ModelYieldToMaturity <- YieldToMaturitySolver(BONDInputData)
ModelYieldToMaturity; inputYieldToMaturity
YtMError <- ModelYieldToMaturity - inputYieldToMaturity
YtMError
```
The bond valuation function is given next including both the semi-annual bond equivalent method as well as the continuously compounded method. Note that this function relies on other functions, such as `CouponsRemaining(B)` and `FractionElapsed(B)`.
```
#
# BondValue: Dollar value of bond including accrued interest (Traditional method)
#
BondValue = function(B){
  with(B,{
    PV = 0.0
    RemainingCoupons = CouponsRemaining(B)
    ElapsedTime = FractionElapsed(B)
    if(YtMType == 'BEY'){
      for(i in 1:RemainingCoupons){
        PV = PV + ( (CouponRate/(Frequency*100.0))*Par ) /
          ((1.0 + (YieldToMaturity/(Frequency*100.0)))^(i - ElapsedTime))
      }
      PV = PV+Par/((1.0 +
        (YieldToMaturity/(Frequency*100.0)))^(RemainingCoupons - ElapsedTime))
    } else if(YtMType == 'CC'){
      for(i in 1:RemainingCoupons){
        PV = PV + ( (CouponRate/(Frequency*100.0))*Par ) *
          exp(-(YieldToMaturity/100)*((i - ElapsedTime)/Frequency))
      }
      PV = PV + Par *
        exp(-(YieldToMaturity/100)*((RemainingCoupons - ElapsedTime)/Frequency))
    } else PV = -99
    return( PV )
  })
}
```
Recall the yield to maturity function is based on rearranging the bond valuation equation so as to equal zero (see Equation **Error! Reference source not found.**). Thus, the price difference function below is used in the yield to maturity function also given below.
```
#
# Function that finds the difference between market price and model value
#  (used in ytm below)
#
PriceDifference <- function(YTM, B){
  tempActualPrice = B$BondPrice + AccruedInterest(B)
  originalYTM = B$YieldToMaturity
```


© 2023 Robert Brooks. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
  B$YieldToMaturity = YTM
  tempBondValue = BondValue(B)
  PD = abs(tempActualPrice - BondValue(B))
  B$YieldToMaturity = originalYTM
  return( PD )
}
#
# Yield To Maturity
#
YieldToMaturitySolver = function(B){
# Minimize the objective function (PriceDifference)
#  by changing YieldToMaturity
# Note using ActualPrice and not BondValue
# optimize will solve for the first parameter in the function
#    (tempYieldToMatuirity in FRMPriceDifference here)
  solution = optimize(PriceDifference, B, interval = c(0.0001, 5000),
    tol = .Machine$double.eps^0.25)
  # Print YieldToMaturity that equates actual and model bond prices
  YTM = solution$minimum
  return( YTM )
}
```

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

Several initial parameters need to be established. The coupon payment frequency per year (`inputFrequency`), the assume bond par value (`inputPar`), a maturity threshold for selecting one bond for detailed analysis (`RoughMaturity`), the number of LSC factors (`NFactors`), the number of years to include in the fitted base curve (`NBaseCurve`), and then whether to fix the range on the plot illustrating the results (`FixRange`, `FRMax`, and `FRMin`) as illustrate here:

```
inputFrequency <- 2
inputPar <- 1000000.0
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
```

When valuing bonds, it is important to distinguish between the trade date and the settlement date. Currently, USTs settle on the trade date plus two business days ($T + 2$). As 9/17/2019 is a Tuesday, we can simply use 17 + 2 or 9/19/2019. Often it is much more convenient to manage programs by placing details in separate files such as a file that manages the inputs and the bond functions.

```
SettlementDateMonth = 9     # Based on file name
SettlementDateDay = 17 + 2  # Current practice is + 2 days settlement
SettlementDateYear = 2019
source("UST Book Inputs.R") # Access UST book
source("UST Functions.R") # UST functions (semi-annual only)
```

Once the particulars of an individual bond have been appropriately acquired, we can test our various functions. The goal should be to produce your own new functions to solve yet unknown analytical challenges. The particular bond we examined is:

```
> UST[SelectedBond,] # Console: Selected bond parameters
    COUPON  ASKED ASKED.YIELD JMaturityDate MaturityDate  APrice
260   6.25 143.04       1.787         25702    5/15/2030 143.125
```

This bond is the first bond trading with more than 10 years to maturity. It has a high coupon of 6.25% and is trading at 143 4/32nds of par or 143.125% of par. The bond matures on 5/15/2030.

```
# Calendar manipulations
N = CouponsRemaining(BONDInputData)
# ElapsedOutput contains fraction, JLastDate, JNextDate, and JCurrentDate
ElapsedOutput = Elapsed(BONDInputData)
# Number of Total Days
NTD <- ElapsedOutput$NextDate - ElapsedOutput$LastDate
```


© 2023 Robert Brooks. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
# 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
```

In this particular case, we find NAD = 127 days have accrued since the last coupon, NTD = 184 days between the last coupon payment date and the next coupon payment date, f = 69.02% of the coupon period has elapsed, N = 22 coupons remaining, and Mat = 10.6549 years to maturity.

Bond valuation calculations are considered next with the following lines of code.

```
# Bond value given yield to maturity
MarketQuotedBondPrice <- inputBondPrice
MarketValueOfBond <- BondValue(BONDInputData)
AccruedInterestAmount <- AccruedInterest(BONDInputData)
ModelQuotedBondPrice <- MarketValueOfBond - AccruedInterestAmount
```

The results are as follows:

```
> MarketValueOfBond; AccruedInterestAmount;
[1] 1452791
[1] 21569.29
> ModelQuotedBondPrice; MarketQuotedBondPrice
[1] 1431222
[1] 1431250
```

Note that our model is only $28 off per $1 million par, well within rounding error on the yield to maturity. If we add $1/10^{th}$ of a basis point to the yield to maturity or 1.786 (1.787 – 0.001), the model quoted bond price is 1,431,342 or $120 more placing the model value well above the market price.

While every function will not be extensively reviewed, we highlight a few important insights with the bond value function.

```
BondValue = function(B){
  with(B,{
    PV = 0.0
    RemainingCoupons = CouponsRemaining(B)
    ElapsedTime = FractionElapsed(B)
    for(i in 1:RemainingCoupons){
      PV = PV + ( (CouponRate/(Frequency*100.0))*Par ) /
        ((1.0 + (YieldToMaturity/(Frequency*100.0)))^(i - ElapsedTime))
    }
    PV = PV+Par/((1.0 +
      (YieldToMaturity/(Frequency*100.0)))^(RemainingCoupons - ElapsedTime))
    return( PV )
  })
}
```

The second line above takes a copy of `BONDInputData` received as `B` and makes the variables accessible without direct reference. For example, `CouponRate` can be used without `B$` placed in front. Also, note the ability to call other functions from within the `BondValue` function, such as `CouponsRemaining(B)`.

The analysis of yield to maturity results in a similar conclusion that our approach is accurate.

```
# Yield to maturity given bond value
inputBondPrice = MarketQuotedBondPrice #Dollars:Quoted price w/o accrued interest
BONDInputData$BondPrice <- inputBondPrice
EstYieldToMaturity = YieldToMaturitySolver(BONDInputData)
```

The results for this bond with initial bond price of $1,431,250 results in an estimated yield to maturity very close to the reported yield to maturity.

```
> EstYieldToMaturity; inputYieldToMaturity
[1] 1.78677
[1] 1.787
```

The R package deployed here is `optimx`, a flexible as well as very thorough package. Specifically, we use the `optimize()` function observed in the *UST Functions.R* file.

```
YieldToMaturitySolver = function(B){
# Minimize the objective function (PriceDifference)
#  by changing YieldToMaturity
```


© 2023 Robert Brooks. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
# Note using ActualPrice and not BondValue
# optimize will solve for the first parameter in the function
#    (tempYieldToMatuirity in FRMPriceDifference here)
  solution = optimize(PriceDifference, B, interval = c(0.0001, 5000),
    tol = .Machine$double.eps^0.25)
  # Print YieldToMaturity that equates actual and model bond prices
  YTM = solution$minimum
  return( YTM )
}
```

Thus far, all we have done is engineered well-known solutions related to bond math. We now seek to value USTs in a way that facilitates comparison across bonds. Based on CMT yields, we derive the CMT curve as well as the discount curve with continuous compounding. With CMT yields imported and initial guesses for the LSC model established, we run the following lines of code:

```
# Given coefficients for discount curve based on LSC,
#  estimate sum squared difference
Answer <- DiffCMTRates(x, NFactors, Sc, NBaseCurve, MarketCMTRates)
Answer
# 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))
```

The function `DiffCMTRates` at its core computes the sum of squared errors between the estimated CMT rate (`SR` denoting eventually swap rates) and the market CMT rates (`MSR`).

```
    SR[j] <- ((1.0 - DF[NFix])/(0.5*sum(DF)))*100.0
  }
  Diff <- sum((MSR - SR)^2, na.rm = TRUE)
```

This function is used in the nonlinear optimization package `optimx`. If the `'nlminb'` method fails, the program tries a few more methods. For almost all applications, the `'nlminb'` method works well. In the rare case where it fails, we have found that other methods are successful.

The four factor LSC model output in this case is:

```
> y
[1]  2.9357806 -0.9041041 -0.5768654 -1.5219563
```


© 2023 Robert Brooks. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.