BondValuation: An R Package for Fixed Coupon Bond Analysis
by Wadim Djatschenko

Abstract The purpose of this paper is to introduce the R package BondValuation for the analysis of large datasets of fixed coupon bonds. The conceptual heterogeneity of fixed coupon bonds traded in the global markets imposes a high degree of complexity on their comparative analysis. Contrary to baseline fixed income theory, in practice, most bonds feature coupon period irregularities. In addition, there are a multitude of day count methods that determine the interest accrual, the cash flows and the discount factors used in bond valuation. Several R packages, e.g., fBonds, RQuantLib, and YieldCurve, provide tools for fixed income analysis. Nevertheless, none of them is capable of evaluating bonds featuring irregular first and/or final coupon periods, and neither provides adequate coverage of day count conventions currently used in the global bond markets. The R package BondValuation closes this gap using the generalized valuation methodology presented in 1.

Introduction

Although bond valuation using the traditional present value approach is fundamental in financial theory and practice, the R community lacks applications that comprehensively handle the peculiarities of real-world fixed coupon bonds. A possible reason for the slow development of adequate computation tools concerns the matter’s theoretical intricacy, characterized by a complex interaction of day count conventions (DCC) and irregularities in the temporal structure of the fixed income instruments.

A day count convention is an instrument-specific set of rules that prescribes the way in which calendar dates are converted to numerical values. Thus, given a schedule of a bond’s anniversary dates (i.e., issue date, coupon payment dates, maturity date), a day count convention is used, e.g., to determine the fraction of regular coupon periods between two calendar dates within the bond’s life. Irregular first and final coupon periods occur irrespective of the stipulated day count convention. The lengths of the first and final coupon periods are measured in fractions of regular coupon periods and calculated according to the rules of the specified day count convention. A first or final coupon period is irregular, if its length differs from 1, which is the length of a regular coupon period.1

The R package RQuantLib (Eddelbuettel et al., 2018) provides access to parts of QuantLib (QuantLib Team, 2018), which is the leading open-source software library for quantitative finance. Currently, QuantLib incorporates methods for the analysis and valuation of a wide variety of financial instruments, such as options, swaps, various financial derivatives, and several types of bonds, including fixed rate bonds. Nevertheless, QuantLib does not implement methods for handling irregular coupon periods, and the coverage of DCCs is not exhaustive with nine different conventions.

A closer examination of bond market data reveals the importance of this problem. According to the Thomson Reuters EIKON database, 99.66% of the plain vanilla fixed coupon bonds that were issued worldwide in 2017 are spread over 15 different DCCs, and 67% of them feature irregular first and/or final coupon periods. Given the enormous size of the global bond market, neglecting irregular coupon periods potentially leads to cash flow miscalculations in the tens of billions of US dollars, as 7 points out.

Essentially, DCCs influence bond valuation in three places. First, the amounts of interest payable at the end of any irregular coupon period are computed according to the respective convention. Second, the powers of the discount factors used in present value calculations depend on the stipulated DCC. Finally, in contrast to stocks, the full prices of bonds are usually not directly observable but need to be calculated as the sum of the quoted clean price and accrued interest, which is paid by the buyer to the seller if the transaction is conducted between two coupon payment dates. Accrued interest is computed conformal to the stipulated bond- and market-specific DCCs.

? addresses these three aspects and proposes a generalized valuation methodology for fixed coupon bonds that allows for irregular first and final coupon periods and is compatible with any conceivable DCC. In summary, the methodology can be described as follows. In a first step, ? introduces a standardized bond-specific temporal structure, which is determined by the stipulated DCC. Based on this time structure, a valuation formula is derived that allows for first and final coupon periods of any lengths. The novelty of this proposed evaluation formula lies in the isolation of each DCC-dependent parameter, resulting in a modular structure that can easily integrate any conceivable DCC. In addition,

---

1? provides a comprehensive overview of most day count conventions currently used in the global bond markets and demonstrates their interactions with irregular coupon periods.
presents closed-form solutions for the valuation formula’s first and second derivatives, which are useful in the Newton-Raphson based determination of the bond’s yield as well as in calculation of duration and convexity. The approach outlined in relies exclusively on information that is typically provided by financial data vendors and is seamlessly implemented in the R package BondValuation.

The remainder of this paper consists of the two main sections, “The BondValuation package” and “Application of the package BondValuation”. The section entitled “The BondValuation package” provides an overview of the functions implemented in the R package BondValuation and briefly illustrates the underlying theoretical concepts. The subsection entitled “Day count conventions” introduces the DCCs covered by BondValuation and demonstrates their impact on interest accrual using the function AccrInt(). Subsequently, the bond-specific temporal structure and its implementation within the function AnnivDates() are illustrated. In the following subsection, the calculation of Cash flows, accrued interest, and dirty price is demonstrated using the functions AnnivDates() and DP(). Next, the functions BondVal.Yield() and BondVal.Price() are used to compute Yield to maturity, duration, and convexity. The section entitled “Application of the package BondValuation” demonstrates how the R package BondValuation can be used for the analysis of large data frames of fixed coupon bonds. The paper ends with a short “Conclusion”.

The BondValuation package


The workhorse function of the package, AnnivDates(), performs a variety of sanity checks on the input data and, if possible, automatically corrects corrupted entries. It determines the bond-specific temporal structure and cash flows. The output of AnnivDates() is used in the downstream processes of the functions BondVal.Price(), BondVal.Yield(), and DP(). While AnnivDates(), BondVal.Price(), BondVal.Yield(), and DP() require bond data as input, the function AccrInt() simply computes the amount of interest accruing from some start date to some end date.

The data frames PanelSomeBonds2016 and SomeBonds2016 provide simulated data of 100 plain vanilla fixed coupon corporate bonds issued in 2016. List.DCC provides an overview of the DCCs implemented in the R package BondValuation. NonBusDays.Brazil is used with the BusDay/252 (Brazilian) convention and contains all non-business days in Brazil from 1946-01-01 to 2299-12-31 based on the Brazilian national holiday calendar.

Day count conventions

All DCCs that are identified by Thomson Reuters EIKON for plain vanilla fixed coupon bonds in 2017, and, additionally, the 30E/360 (ISDA) method, are covered by the R package BondValuation:

> # example 1
> library(BondValuation)
> print(List.DCC, row.names = FALSE)

<table>
<thead>
<tr>
<th>DCC</th>
<th>DCC. Name</th>
<th>DCC. Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACT/ACT (ISDA)</td>
<td>ISDA (1998); ISDA (2006) section 4.16 (b)</td>
</tr>
<tr>
<td>2</td>
<td>ACT/ACT (ICMA)</td>
<td>ICMA Rule 251; ISDA (2006) section 4.16 (c)</td>
</tr>
<tr>
<td>3</td>
<td>ACT/ACT (AFB)</td>
<td>ISDA (1998); EBF (2004); SWX (2003)</td>
</tr>
<tr>
<td>4</td>
<td>ACT/365L</td>
<td>ICMA Rule 251; SWX (2003)</td>
</tr>
<tr>
<td>5</td>
<td>30/360</td>
<td>ISDA (2006) section 4.16 (f); MSRB (2017) Rule G-33</td>
</tr>
<tr>
<td>6</td>
<td>30E/360</td>
<td>ICMA Rule 251; ISDA (2006) section 4.16 (g); SWX (2003)</td>
</tr>
<tr>
<td>7</td>
<td>30/360 (German)</td>
<td>ISDA (2006) section 4.16 (h)</td>
</tr>
<tr>
<td>8</td>
<td>30/360 US</td>
<td>EBF (2004); SWX (2003)</td>
</tr>
<tr>
<td>9</td>
<td>ACT/365 (Fixed)</td>
<td>Mayle (1993); SWX (2003)</td>
</tr>
<tr>
<td>10</td>
<td>ACT/365 (Fixed)</td>
<td>ISDA (2006) section 4.16 (d); SWX (2003)</td>
</tr>
<tr>
<td>11</td>
<td>ACT(NL)/365</td>
<td>Krgin (2002); Thomson Reuters EIKON</td>
</tr>
<tr>
<td>12</td>
<td>ACT/360</td>
<td>ISDA (2006) section 4.16 (e); SWX (2003)</td>
</tr>
<tr>
<td>13</td>
<td>30/365</td>
<td>Krgin (2002); Thomson Reuters EIKON</td>
</tr>
<tr>
<td>14</td>
<td>ACT/365 (Canadian Bond)</td>
<td>IIAC (2018); Thomson Reuters EIKON</td>
</tr>
<tr>
<td>15</td>
<td>ACT/364</td>
<td>Thomson Reuters EIKON</td>
</tr>
<tr>
<td>16</td>
<td>BusDay/252 (Brazilian)</td>
<td>Caputo Silva et al. (2010), ?</td>
</tr>
</tbody>
</table>

? provides a comprehensive overview of these DCCs.
The function `AccrInt()` can be used to compare the differences in interest accrual between the day count methods. As an example, the code below returns the number of days and the amount of interest (in percent of the bond’s par value) accrued from Start = 2011-08-31 to End = 2012-02-29 with different DCCs, ceteris paribus. In this example, we assume that CpY = 2 coupons are paid per year, the nominal interest rate p.a. is Coup = 5.25%, the bond is redeemed at RV = 100% of its par value, and payments follow the End-of-Month rule\(^3\), i.e., EOM = 1. In addition, some of the DCCs require specification of the next coupon payment’s year figure, YearNCP = 2012, and the maturity date, Mat = 2021-08-31.

```r
> # example 2
> library(BondValuation)
> DCC_Comparison<-data.frame(Start = rep(as.Date("2011-08-31"), 16),
+ End = rep(as.Date("2012-02-29"), 16),
+ Coup = rep(5.25, 16),
+ DCC = seq(1, 16),
+ DCC.Name = List.DCC[, 2],
+ RV = rep(100, 16),
+ CpY = rep(2, 16),
+ Mat = rep(as.Date("2021-08-31"), 16),
+ YearNCP = rep(2012, 16),
+ EOM = rep(1, 16))
> AccrIntOutput <- suppressWarnings(
+ apply(
+ DCC_Comparison[, c('Start', 'End', 'Coup', 'DCC', 'RV', 'CpY', 'Mat',
+ 'YearNCP', 'EOM')], 1,
+ function(y) AccrInt(y[1], y[2], y[3], y[4], y[5], y[6], y[7], y[8], y[9])
+ )
+ )
> Accrued_Interest <- do.call(rbind, lapply(AccrIntOutput, function(x) x[[1]]))
> Days_Accrued <- do.call(rbind, lapply(AccrIntOutput, function(x) x[[2]]))
> DCC_Comparison <- cbind(DCC_Comparison, Accrued_Interest, Days_Accrued)
> print(DCC_Comparison[, c('DCC.Name', 'Start', 'End', 'Days_Accrued',
+ 'Accrued_Interest')], row.names = FALSE)
```

<table>
<thead>
<tr>
<th>DCC_Name</th>
<th>Start</th>
<th>End</th>
<th>Days_Accrued</th>
<th>Accrued_Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT/ACT (ISDA) 2011-08-31</td>
<td>2012-02-29</td>
<td>182</td>
<td>2.615490</td>
<td></td>
</tr>
<tr>
<td>ACT/ACT (ICMA) 2011-08-31</td>
<td>2012-02-29</td>
<td>182</td>
<td>2.625000</td>
<td></td>
</tr>
<tr>
<td>ACT/ACT (AFB) 2011-08-31</td>
<td>2012-02-29</td>
<td>182</td>
<td>2.617808</td>
<td></td>
</tr>
<tr>
<td>ACT/365L 2011-08-31</td>
<td>2012-02-29</td>
<td>179</td>
<td>2.610656</td>
<td></td>
</tr>
<tr>
<td>30/360 2011-08-31</td>
<td>2012-02-29</td>
<td>179</td>
<td>2.610417</td>
<td></td>
</tr>
<tr>
<td>30E/360 2011-08-31</td>
<td>2012-02-29</td>
<td>179</td>
<td>2.610417</td>
<td></td>
</tr>
<tr>
<td>30E/360 (ISDA) 2011-08-31</td>
<td>2012-02-29</td>
<td>180</td>
<td>2.625000</td>
<td></td>
</tr>
<tr>
<td>30/360 (German) 2011-08-31</td>
<td>2012-02-29</td>
<td>180</td>
<td>2.625000</td>
<td></td>
</tr>
<tr>
<td>30/360 US 2011-08-31</td>
<td>2012-02-29</td>
<td>179</td>
<td>2.610417</td>
<td></td>
</tr>
<tr>
<td>ACT/365 (Fixed) 2011-08-31</td>
<td>2012-02-29</td>
<td>182</td>
<td>2.617808</td>
<td></td>
</tr>
<tr>
<td>ACT(NL)/365 2011-08-31</td>
<td>2012-02-29</td>
<td>182</td>
<td>2.617808</td>
<td></td>
</tr>
<tr>
<td>ACT/360 2011-08-31</td>
<td>2012-02-29</td>
<td>179</td>
<td>2.654167</td>
<td></td>
</tr>
<tr>
<td>30/365 2011-08-31</td>
<td>2012-02-29</td>
<td>179</td>
<td>2.574658</td>
<td></td>
</tr>
<tr>
<td>ACT/365 (Canadian Bond) 2011-08-31</td>
<td>2012-02-29</td>
<td>182</td>
<td>2.625000</td>
<td></td>
</tr>
<tr>
<td>ACT/364 2011-08-31</td>
<td>2012-02-29</td>
<td>182</td>
<td>2.625000</td>
<td></td>
</tr>
<tr>
<td>BusDay/252 (Brazilian) 2011-08-31</td>
<td>2012-02-29</td>
<td>124</td>
<td>2.549769</td>
<td></td>
</tr>
</tbody>
</table>

**Bond-specific temporal structure**

The function `AnnivDates()` evaluates bond-specific information and returns the bond’s time-invariant characteristics in the data frame `Traits`, the bond’s temporal structure in the data frame `DateVectors` and, if the nominal interest rate is passed, the bond’s cash flows in the data frame `PaySched`\(^4\). The classes and formats of input data are checked and adjusted, if possible. Moreover, `AnnivDates()` performs several plausibility tests, e.g., whether the provided calendar dates are in a correct chronological order and whether there are inconsistencies among the provided parameters. The results of these sanity checks are reported in the data frame `Warnings`.

\(^3\)See manual to the R package `BondValuation` for details on implementation and Krgin (2002) for the theoretical background of the End-of-Month rule.

\(^4\)See ? for theoretical background.
The minimum accepted input for `AnnivDates()` are two calendar dates, of which the first is interpreted as the bond’s issue date and the second as its maturity date. If, as illustrated below, only two dates are passed to `AnnivDates()`, several parameters take on default values, which are reported in warning messages.

```r
> # example 3
> library(BondValuation)
> AnnivDates(as.Date("2019-05-31"), "2021-07-31")

$ "Warnings"
Em_FIAD_differ EmMatMissing CpYOVERRIDE RV_set100percent NegLifeFlag
0 1 1 0 0
ZeroFlag Em_Mat_SameMY ChronErrorFlag FIPD_LIPD_equal IPD_CpY_Corrupt
0 0 0 0 0
EOM_Deviation EOMOverride DCCOverride NoCoups
0 1 1 0

$Traits
DateOrigin CpY FIAD Em Em_Orig FIPD
FIPD_Orig est_FIPD LIPD LIPD_Orig est_LIPD Mat
<NA> 2019-07-31 <NA> <NA> 2021-01-31 2021-07-31
Refer FCPType FCPLength LCPTYPE LCPLength Par
2021-07-31 short 0.3370166 regular 1 100
CouponInPercent.p.a DayCountConvention EOM_Orig est_EOM EOM_used
NA 2 NA 1 1

$DateVectors
RealDates RD_indexes CoupDates CD_indexes AnnivDates AD_indexes
2019-05-31 0.6629834 2019-07-31 1 2019-01-31 0
2019-07-31 1.0000000 2020-01-31 2 2019-07-31 1
2020-01-31 2.0000000 2020-07-31 3 2020-01-31 2
2020-07-31 3.0000000 2021-01-31 4 2020-07-31 3
2021-01-31 4.0000000 2021-07-31 5 2021-01-31 4
2021-07-31 5.0000000 <NA> NA 2021-07-31 5

Warning messages:
1: In InputFormatCheck(Em = Em, Mat = Mat, CpY = CpY, FIPD = FIPD, :
   The maturity date (Mat) is supplied as a string of class "character" in the
   format "yyyy-mm-dd". It is converted to class "Date" using the command
   as.Date(Mat, "Y-M-d") and processed as Mat = 2021-07-31 .
2: In AnnivDates(as.Date("2019-05-31"), "2021-07-31") :
   Number of interest payments per year (CpY) is missing or NA. CpY is set 2!
3: In AnnivDates(as.Date("2019-05-31"), "2021-07-31") :
   Redemption value (RV) is missing or NA. RV is set 100!
4: In AnnivDates(as.Date("2019-05-31"), "2021-07-31") :
   EOM was not provided or NA! EOM is set 1 .
   Note: The available calendar dates suggest that EOM = 1 .
5: In AnnivDates(as.Date("2019-05-31"), "2021-07-31") :
   The day count identifier (DCC) is missing or NA. DCC is set 2 (Act/Act (ICMA))!

Since neither the first nor the penultimate coupon payment date is passed to `AnnivDates()`, the calendar dates in the data frame `DateVectors` are constructed backwards starting from the maturity date 2021-07-31. This results in a bond with a short first coupon period having a length of $Traits$FCPLength = 0.3370166 regular coupon periods.

The data frame `DateVectors` contains three vectors of calendar dates, RealDates, CoupDates, and AnnivDates, and their corresponding indexes, RD_indexes, CD_indexes, and AD_indexes. The vector RealDates comprises the bond’s issue date, maturity date, and all coupon payment dates in between, while CoupDates contains only the coupon payment dates. The vector AnnivDates consists of the bond’s so-called anniversary dates, i.e., scheduled coupon dates and notional coupon dates located before the first and after the penultimate coupon payment dates. The lengths of the first ($Traits$FCPLength) and final coupon periods ($Traits$LCPLength) are calculated as differences between the corresponding values of the vectors AD_indexes and RD_indexes. RD_indexes are used in the functions BondVal.Price(), BondVal.Yield(), and DP() to determine the powers of discount factors in pricing formulas.

As warning message 4 in the example above reports, the function `AnnivDates()` analyzes the
provided calendar dates to ascertain whether the bond follows the End-of-Month rule (EOM). An automated replacement of the provided value of EOM by the value determined by the function AnnivDates() can be activated by setting option FindEOM = TRUE, which defaults to FALSE.

The following example illustrates the effect of EOM on DateVectors. In addition to issue date (Em) and maturity date (Mat), the first coupon payment date (FIPD), the penultimate coupon payment date (LIPD), the number of coupon payments p.a. (CpY), and EOM are passed to the function AnnivDates():

```r
> # example 4
> library(BondValuation)
> # example 4a: computing DateVectors for EOM = 1
> EOM.input <- 1
> AnnivDates(Em = as.Date("2019-05-31"),
+ Mat = as.Date("2021-07-31"),
+ CpY = 2,
+ FIPD = as.Date("2020-02-29"),
+ LIPD = as.Date("2021-02-28"),
+ EOM = EOM.input)
> AnnivDates(Em = as.Date("2019-05-31"),
+ Mat = as.Date("2021-07-31"),
+ CpY = 2,
+ FIPD = as.Date("2020-02-29"),
+ LIPD = as.Date("2021-02-28"),
+ EOM = EOM.input)$DateVectors
```

```
RealDates RD_indexes CoupDates CD_indexes AnnivDates AD_indexes
1 2019-05-31 -0.500000 2020-02-29 1.000000 2019-02-28 -1
2 2020-02-29 1.000000 2020-08-31 2.000000 2019-08-31 0
3 2020-08-31 2.000000 2021-02-28 3.000000 2020-02-29 1
4 2021-02-28 3.000000 2021-07-31 3.831522 2020-02-29 1
5 2021-07-31 3.831522 <NA> NA 2021-02-28 3
6 <NA> NA <NA> NA 2021-08-31 4
```

WARNING messages:
1: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"), :
Redemption value (RV) is missing or NA. RV is set 100!
2: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"), :
The day count identifier (DCC) is missing or NA. DCC is set 2 (Act/Act (ICMA))!

```
> # example 4b: computing DateVectors for EOM = 0
> EOM.input <- 0
> AnnivDates(Em = as.Date("2019-05-31"),
+ Mat = as.Date("2021-07-31"),
+ CpY = 2,
+ FIPD = as.Date("2020-02-29"),
+ LIPD = as.Date("2021-02-28"),
+ EOM = EOM.input)
> AnnivDates(Em = as.Date("2019-05-31"),
+ Mat = as.Date("2021-07-31"),
+ CpY = 2,
+ FIPD = as.Date("2020-02-29"),
+ LIPD = as.Date("2021-02-28"),
+ EOM = EOM.input)$DateVectors
```

```
RealDates RD_indexes CoupDates CD_indexes AnnivDates AD_indexes
1 2019-05-31 -0.4945055 2020-02-29 1.000000 2019-02-28 -1
2 2020-02-29 1.0000000 2020-08-29 2.000000 2019-08-29 0
3 2020-08-29 2.0000000 2021-02-28 3.000000 2020-02-29 1
4 2021-02-28 3.0000000 2021-07-31 3.840659 2020-02-29 2
5 2021-07-31 3.8406593 <NA> NA 2021-02-28 3
6 <NA> NA <NA> NA 2021-08-29 4
```

WARNING messages:
1: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"), :
Redemption value (RV) is missing or NA. RV is set 100!
2: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"), :
The available calendar dates suggest that EOM = 1. Option FindEOM = FALSE is active. Provided EOM is not overridden and remains EOM = 0.
3: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"), :
The day count identifier (DCC) is missing or NA. DCC is set 2 (Act/Act (ICMA))!
```
```
In contrast to example 3, the bond in example 4 features a long first and a short final coupon period. The function AnnivDates() has checked whether the provided dates FIPD and LIPD are on each other’s anniversaries and constructed the calendar dates in DateVectors backwards and forwards from LIPD. The values of RD_indexes and AD_indexes are illustrated in Figure 1, where E corresponds to the first element of RD_indexes and M is the final element of RD_indexes.

As shown in example 4, all else equal, the value of EOM affects the DCC-conformal temporal locations of the issue date E and the maturity date M and, hence, the lengths of the first and final coupon periods, which, in turn, determine the amounts of interest paid on the first and final coupon payment dates. So far, no value of DCC was passed to the function AnnivDates(). As reported in the warning
messages in example 4, the parameter DCC defaults to the Act/Act (ICMA) convention. The following, example 5, illustrates how RD_indexes, FCPLength and LCPLength vary across DCCs for EOM = 0.

![Timeline illustration of the bonds in examples 4 and 5.](image)

> # example 5
> library(BondValuation)
> TempStruct.by.DCC <- data.frame(Em = rep(as.Date("2019-05-31"), 16),
+ Mat = rep(as.Date("2021-07-31"), 16),
+ CpY = rep(2, 16),
+ FIPD = rep(as.Date("2020-02-29"), 16),
+ LIPD = rep(as.Date("2021-02-28"), 16),
+ FIAD = rep(as.Date("2019-05-31"), 16),
+ DCC = seq(1, 16),
+ EOM = rep(0, 16),
+ DCC.Name = List.DCC[, 2])
> # Applying AnnivDates() to the data frame TempStruct.by.DCC for EOM = 0
> suppressWarnings(
+ FullAnalysis.EOM0 <- apply(
+ TempStruct.by.DCC[, c("Em", "Mat", "CpY", "FIPD", "LIPD", "FIAD", "DCC", "EOM")],
+ 1, function(y) AnnivDates(
+ y[1], y[2], y[3], y[4], y[5], y[6], , , y[7], y[8])
+ )
+ )
> FCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, 
+ max, na.omit)
+ )
+ )
+ )
+ )
+ )
+ )
> TempStruct.EOM0 <- lapply(FullAnalysis.EOM0, 
+ max, lengths(TempStruct.EOM0)))
> TempStruct.EOM0 <- as.data.frame(do.call(rbind, lapply(TempStruct.EOM0, round, 4)))
> names(TempStruct.EOM0)[c(18:16)] <- c("E", "01", "02", "03", "M",
+ "FCPLength", "LCPLength")
> print(TempStruct.EOM0[, c(9:ncol(TempStruct.EOM0))],
+ row.names = FALSE)

<table>
<thead>
<tr>
<th>DCC_Name</th>
<th>DCC</th>
<th>E</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>M</th>
<th>FCPLength</th>
<th>LCPLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT/ACT (ISDA)</td>
<td>-0.4945</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8407</td>
<td>1.4945</td>
<td>0.8407</td>
<td></td>
</tr>
<tr>
<td>ACT/ACT (ICMA)</td>
<td>-0.4945</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8407</td>
<td>1.4945</td>
<td>0.8407</td>
<td></td>
</tr>
<tr>
<td>ACT/ACT (AFB)</td>
<td>-0.4945</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8407</td>
<td>1.4945</td>
<td>0.8407</td>
<td></td>
</tr>
<tr>
<td>30/360</td>
<td>-0.4862</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8453</td>
<td>1.4862</td>
<td>0.8453</td>
<td></td>
</tr>
<tr>
<td>30E/360</td>
<td>-0.4917</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8398</td>
<td>1.4917</td>
<td>0.8398</td>
<td></td>
</tr>
<tr>
<td>30/360 (ISDA)</td>
<td>-0.4972</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8380</td>
<td>1.4972</td>
<td>0.8380</td>
<td></td>
</tr>
<tr>
<td>30/360 (German)</td>
<td>-0.4972</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8380</td>
<td>1.4972</td>
<td>0.8380</td>
<td></td>
</tr>
<tr>
<td>30/360 US</td>
<td>-0.4862</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8453</td>
<td>1.4862</td>
<td>0.8453</td>
<td></td>
</tr>
<tr>
<td>ACT/365 (Fixed)</td>
<td>-0.4945</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8407</td>
<td>1.4945</td>
<td>0.8407</td>
<td></td>
</tr>
<tr>
<td>ACT/365 (Canadian Bond)</td>
<td>-0.4945</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8407</td>
<td>1.4945</td>
<td>0.8407</td>
<td></td>
</tr>
<tr>
<td>ACT/364</td>
<td>-0.4945</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8407</td>
<td>1.4945</td>
<td>0.8407</td>
<td></td>
</tr>
<tr>
<td>BusDay/252 (Brazilian)</td>
<td>-0.5040</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3.8425</td>
<td>1.5040</td>
<td>0.8245</td>
<td></td>
</tr>
</tbody>
</table>
The output of example 5 shows that for the specified bond, there is little variation in temporal structure across the DCCs. Specifically, with \( DCC \in \{1, 2, 3, 4, 10, 11, 12, 14, 15\} \), the values of \( RD\_indexes \) are \(-0.4945, 1, 2, 3, 3.8407\); with \( DCC \in \{5, 9, 13\} \), it holds \( RD\_indexes = \{-0.4862, 1, 2, 3, 3.8353\} \), and with \( DCC \in \{7, 8\} \), we get \( RD\_indexes = \{-0.4972, 1, 2, 3, 3.8380\} \). Only the DCCs 6 and 16 produce a unique temporal structure for this bond. Nevertheless, it would be wrong to infer from this that the temporal structure always has so little variation across the DCCs, as example 6 illustrates.

```
> # example 6
> library(BondValuation)
> TempStruct.by.DCC <- data.frame(Em = rep(as.Date("2019-10-31"), 16),
+ Mat = rep(as.Date("2024-02-29"), 16),
+ CpY = rep(2, 16),
+ FIPD = rep(as.Date("2020-03-30"), 16),
+ LIPD = rep(as.Date("2023-03-30"), 16),
+ FIAD = rep(as.Date("2019-10-31"), 16),
+ DCC = seq(1, 16),
+ EOM = rep(0, 16),
+ DCC.Name = List.DCC[, 2])
>
> # Applying AnnivDates() to the data frame TempStruct.by.DCC for EOM = 0
> suppressWarnings(
+ FullAnalysis.EOM0 <- apply(
+ TempStruct.by.DCC[, c("Em","Mat","CpY","FIPD","LIPD","FIAD","DCC","EOM")],
+ 1, function(y) AnnivDates(
+ y[1], y[2], y[3], y[4], y[5], y[6], , , y[7], y[8])
+ )
+ )
>
> FCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, 
+ c("Em","Mat","CpY","FIPD","LIPD","FIAD","DCC","EOM")),
+ , na.omit)
+ 1, function(y) AnnivDates(
+ y[1], y[2], y[3], y[4], y[5], y[6], , , y[7], y[8])
+ )
+ )
> FCPLength.EOM0 <- apply(lapply(FCPLength.EOM0, "\[", 2, "\[", 15
+ , na.omit)
> FCPLength.EOM0 <- as.data.frame(do.call(rbind, FCPLength.EOM0, round, 4))
> LCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, 
+ c("Em","Mat","CpY","FIPD","LIPD","FIAD","DCC","EOM")),
+ , na.omit)
+ 1, function(y) AnnivDates(
+ y[1], y[2], y[3], y[4], y[5], y[6], , , y[7], y[8])
+ )
+ )
> LCPLength.EOM0 <- apply(lapply(LCPLength.EOM0, "\[", 2, "\[", 17
+ , na.omit)
> LCPLength.EOM0 <- as.data.frame(do.call(rbind, LCPLength.EOM0, round, 4))
> TempStruct.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, 
+ c("Em","Mat","CpY","FIPD","LIPD","FIAD","DCC","EOM")),
+ , na.omit)
+ 1, function(y) AnnivDates(
+ y[1], y[2], y[3], y[4], y[5], y[6], , , y[7], y[8])
+ )
+ )
> TempStruct.EOM0 <- apply(TempStruct.EOM0, "\[", 3, "\[", 2
+ , na.omit)
> TempStruct.EOM0 <- lapply(TempStruct.EOM0, function(y) bounds(y, 
+ max(lengths(TempStruct.EOM0))))
> TempStruct.EOM0 <- as.data.frame(do.call(rbind, TempStruct.EOM0, round, 4))
> TempStruct.by.DCC.EOM0 <- cbind(TempStruct.by.DCC, TempStruct.EOM0,
+ FCPLength.EOM0, LCPLength.EOM0)
+ names(TempStruct.by.DCC.EOM0)[c(10:20)] <- c("E","01","02","03","04","05","06",
+ "07","M","FCPLength","LCPLength")
> print(TempStruct.by.DCC.EOM0[, c(9:ncol(TempStruct.by.DCC.EOM0))],
+ row.names = FALSE)
```

```
DCC_Name E 01 02 03 04 05 06 07 M FCPLength LCPLength
ACT/ACT (ISDA) 0.1706 1 2 3 4 5 6 7 8 8.8354 0.8294 1.8354
ACT/ACT (ICMA) 0.1703 1 2 3 4 5 6 7 8 8.8352 0.8297 1.8352
ACT/ACT (AFB) 0.1708 1 2 3 4 5 6 7 8 8.8375 0.8292 1.8375
ACT/365L 0.1703 1 2 3 4 5 6 7 8 8.8352 0.8297 1.8352
30/360 0.1703 1 2 3 4 5 6 7 8 8.8278 0.8333 1.8278
30E/360 0.1703 1 2 3 4 5 6 7 8 8.8278 0.8333 1.8278
30E/360 (ISDA) 0.1703 1 2 3 4 5 6 7 8 8.8278 0.8333 1.8278
30/360 (German) 0.1703 1 2 3 4 5 6 7 8 8.8333 0.8333 1.8333
30/360 US 0.1703 1 2 3 4 5 6 7 8 8.8278 0.8333 1.8278
ACT/365 (Fixed) 0.1703 1 2 3 4 5 6 7 8 8.8352 0.8297 1.8352
ACT(NL)/365 0.1713 1 2 3 4 5 6 7 8 8.8398 0.8287 1.8398
ACT/360 0.1703 1 2 3 4 5 6 7 8 8.8352 0.8297 1.8352
30/365 0.1702 1 2 3 4 5 6 7 8 8.8352 0.8297 1.8352
ACT/365 (Canadian Bond) 0.1703 1 2 3 4 5 6 7 8 8.8352 0.8297 1.8352
ACT/364 0.1703 1 2 3 4 5 6 7 8 8.8352 0.8297 1.8352
BusDay/252 (Brazilian) 0.1840 1 2 3 4 5 6 7 8 8.8279 0.8160 1.8279
```

The output of example 6 reveals that, all else equal, the specified bond can feature 7 different temporal structures, depending on the stipulated DCC. While in example 5 the “ACT/ACT” family of DCCs produced the same temporal structure, in example 6 most of the “30/360” DCCs result in the same day count.

The R Journal Vol. XX/YY, AAAA 20ZZZ
ISSN 2073-4859
Cash flows, accrued interest, and dirty price

In the preceding examples of `AnnivDates()`, no information on nominal interest rate (`Coup`) and redemption value (`RV`) was passed to the function. If this information, however, is provided, then `AnnivDates()` generates the data frame `PaySched`, consisting of the scheduled coupon dates and the corresponding cash flows. As ? points out, precise application of the respective DCC’s mathematical rule results in varying interest payments. While this is intended for calculation of the cash flows paid at the ends of irregular first and final coupon periods, most issuers design their bonds to pay the same cash flow at the end of each regular period. With default RegCF.equal = 0 the function `AnnivDates()` calculates all cash flows according to the mathematical rule of the respective DCC. Passing any other value to RegCF.equal forces all regular cash flows to be equal sized. The following, example 7, uses the same input as example 6 supplemented by information on nominal interest rate p.a. (`Coup` = 10%) and redemption value (`RV` = 100%) and illustrates the differences in cash flows (in percent of the bond’s par value) by DCC between the two modes of RegCF.equal.

```r
> library(BondValuation)
> CashFlows.by.DCC <- data.frame(Em = rep(as.Date("2019-10-31"), 16),
+ Mat = rep(as.Date("2024-02-29"), 16),
+ Cpf = rep(2, 16),
+ FIPD = rep(as.Date("2020-03-30"), 16),
+ LIPD = rep(as.Date("2023-03-30"), 16),
+ FIAD = rep(as.Date("2019-10-31"), 16),
+ RV = rep(100, 16),
+ Coup = rep(10, 16),
+ DCC = seq(1, 16),
+ EOM = rep(0, 16),
+ DCC.Name = List.DCC[, 2])
> # Applying AnnivDates() to the data frame CashFlows.by.DCC for EOM = 0
> # with option RegCF.equal = 0 and RegCF.equal = 1
> Suffix <- c("RegCFvary","RegCFequal")
> for (i in c(0,1)) {
+ suppressWarnings{
+ FullAnalysis <- apply(
+ CashFlows.by.DCC[, c("Em","Mat","Cpf","FIPD","LIPD","FIAD","RV",
+ "Coup","DCC","EOM")],
+ 1, function(y) AnnivDates(
+ y[1],y[2],y[3],y[4],y[5],y[6],y[7],y[8],y[9],y[10], RegCF.equal = i)
+ )
+ )
+ Cashflows <- lapply(lapply(lapply(FullAnalysis,
+ `\[\[`, 4), `\[\[`, 2)
+ , na.omit)
+ Cashflows <- as.data.frame(do.call(rbind, lapply(Cashflows, round, 4)))
+ Cashflows <- cbind(Cashflows.by.DCC, Cashflows)
+ names(Cashflows)[c(12:19)] <- c(
+ "CF.1","CF.2","CF.3","CF.4","CF.5","CF.6","CF.7","CF.M")
+ assign(paste0("CashFlows.by.DCC.",Suffix[i+1]),Cashflows)
+ rm(FullAnalysis,Cashflows)
+ }
> # RegCF.equal = 0, \textit{i.e.,} regular cash flows may vary
> print(CashFlows.by.DCC.RegCFvary[, c(11:ncol(CashFlows.by.DCC.RegCFvary))],
+ row.names = FALSE)
```

Here are the results for regular cash flows:

<table>
<thead>
<tr>
<th>DCC.Name</th>
<th>CF.1</th>
<th>CF.2</th>
<th>CF.3</th>
<th>CF.4</th>
<th>CF.5</th>
<th>CF.6</th>
<th>CF.7</th>
<th>CF.M</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT/ACT (ICMA)</td>
<td>4.1484</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>9.2011</td>
</tr>
<tr>
<td>ACT/365L</td>
<td>4.1257</td>
<td>5.0273</td>
<td>4.9589</td>
<td>5.0411</td>
<td>4.9589</td>
<td>5.0411</td>
<td>4.9589</td>
<td>9.1803</td>
</tr>
<tr>
<td>30/360</td>
<td>4.1667</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>9.1389</td>
</tr>
<tr>
<td>30/360 (German)</td>
<td>4.1667</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>9.1667</td>
</tr>
<tr>
<td>30/360 US</td>
<td>4.1667</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>5.0000</td>
<td>9.1389</td>
</tr>
<tr>
<td>ACT/365 (Fixed)</td>
<td>4.1370</td>
<td>5.0411</td>
<td>4.9589</td>
<td>5.0411</td>
<td>4.9589</td>
<td>5.0411</td>
<td>4.9589</td>
<td>9.2055</td>
</tr>
</tbody>
</table>
Please note that, irrespective of the value of RegCF.equal passed to AnnivDates(), the cash flows at the ends of all regular coupon periods are equal sized with the conventions ACT/ACT (ICMA), ACT/365 (Canadian Bond), and BusDay/252 (Brazilian). This is due to the DCC-specific rules described in ?. While with the majority of DCCs, the cash flows are computed based upon the ratio of the nominal interest rate p.a. and the number of interest payments per year, which yields regular cash flows of 5%, BusDay/252 (Brazilian) determines them exponentially, resulting in regular cash flows of 4.8809%.

The vast majority of bonds are quoted clean, i.e., their observable prices do not contain accrued interest. The actual price that a bond buyer pays to the seller is called full or dirty price and computed as the sum of the quoted clean price and accrued interest, which is calculated according to the respective DCC. Accrued interest and the dirty price of a specific bond can be calculated using the function DP(). In addition to the input parameters required by AnnivDates(), the clean price (CP) and the settlement date (SETT) need to be passed to the function DP(). The following, example 8, returns the accrued interest and dirty price by DCC for the same bond as used in example 7, assuming that on the settlement dates SETT1 = 2020-09-28, SETT2 = 2023-03-30, and SETT3 = 2024-01-15, the quoted clean price is 105% of the bond’s par value.

> # example 8
> library(BondValuation)
> AccrIntDP.by.DCC <- data.frame(CP = 105,
+  SETT1 = rep(as.Date("2020-09-28"), 16),
+  SETT2 = rep(as.Date("2023-03-30"), 16),
+  SETT3 = rep(as.Date("2024-01-15"), 16),
+  Em = rep(as.Date("2019-10-31"), 16),
+  Mat = rep(as.Date("2024-02-29"), 16),
+  CpY = rep(2, 16),
+  FIPD = rep(as.Date("2020-03-30"), 16),
+  LIPD = rep(as.Date("2023-03-30"), 16),
+  FIAD = rep(as.Date("2019-10-31"), 16),
+  RV = rep(100, 16),
+  Coup = rep(10, 16),
+  DCC = seq(1, 16),
+  EOM = rep(0, 16),
+  DCC.Name = List.DCC[, 2])
>
> Suffix <- c("SETT1","SETT2","SETT3")
> for (i in c(1:3)) {
+  DP.Output<-suppressWarnings(}
The yield to maturity p.a. is determined as the value \( y \) that fulfills equation (1).

\[
DP_t = CP_t + AC(t_t) = \frac{CN(t_t)}{1 + \frac{y}{R}} + \sum_{i=1}^{\eta} \frac{CF_{t+k}}{1 + \frac{y}{R}}^{w+i} + \frac{CF_M + RV}{1 + \frac{y}{R}}^{w+\eta+i}. \tag{1}
\]

Yield to maturity, duration, and convexity

The yield to maturity p.a. is determined as the value \( y \) that fulfills equation (1).

\[
DP_t = CP_t + AC(t_t) = \frac{CN(t_t)}{1 + \frac{y}{R}} + \sum_{i=1}^{\eta} \frac{CF_{t+k}}{1 + \frac{y}{R}}^{w+i} + \frac{CF_M + RV}{1 + \frac{y}{R}}^{w+\eta+i}. \tag{1}
\]

In equation (1), \( DP_t \) denotes the dirty price, consisting of the quoted clean price \( CP_t \) and accrued interest \( AC(t_t) \). Conformal with the notation in \( t_t \) is the settlement date and \( \tau \) its index in the temporal structure established by the function \( \text{AnnivDates()} \). On the right side of equation (1), \( CN(t_t) \) denotes the next coupon payment after the settlement date \( t_t \), and \( w \) is the fraction of a regular coupon period left until this payment. The set \( CF_{t+k} \) with \( i \in \{ x \in \mathbb{N} \mid x \in [1, \eta] \} \) contains all interest payments after \( t_t \), excluding the final coupon payment, \( CF_M \), where \( k \) is the index of the next coupon date after \( t_t \), \( \eta \) is the number of interest payment dates between \( t_t \) and the penultimate coupon date, and \( M \) is the index corresponding to the bond’s maturity date. \( RV \) denotes the redemption payment, \( z \) represents the length of the final coupon period, and \( h \) represents the number of regular interest payments per year.

The dirty price \( DP_t \) and the accrued interest \( AC(t) \) are computed as illustrated in example 8. The cash flows \( CN(t_t), CF_{t+k}, \) and \( CF_M \) are calculated as demonstrated in example 7. The powers in the denominators in equation (1) are found based on the temporal structure established by the function \( \text{AnnivDates()} \), as shown in example 6.

Essentially, the same \( \text{DCC} \) is used for computation of cash flows, accrued interest and the indexes of the temporal structure. Nevertheless, the option \( \text{Calc. Method} \) in the functions \( \text{BondVal.Price()} \) and \( \text{BondVal.Yield()} \) allows for switching the calculation method for the temporal structure to \( \text{DCC} = 2 \), \( i.e., \text{ACT/ACT (ICMA)} \), while keeping the \( \text{DCC} \) passed to the function for determination of cash flows and accrued interest.

The function \( \text{BondVal.Price()} \) can be used to compute a bond’s clean price, \( (CP_t) \), given its yield to maturity p.a. \( (y) \), while the function \( \text{BondVal.Yield()} \) returns \( y \) given \( CP_t \). Besides accrued interest \( (AC(t_t)) \) and dirty price \( (DP_t) \), both functions return \( \tau \), Macaulay duration, modified duration, and
The convexity of the specified bond. The following, example 9, demonstrates the use of the function BondVal.Yield() for the bond analyzed in example 8. In the output of example 9, YtM denotes the bond’s yield to maturity p.a. in percent, DUR is the bond’s modified duration in years, and Conv is the bond’s convexity in years. The suffixes .S1, .S2, and .S3 correspond to the three analyzed settlement dates, SETT1 = 2020-09-28, SETT2 = 2023-03-30, and SETT3 = 2024-01-15. For space reasons, the first column of the displayed data frame contains the DCC-codes instead of their names.

```r
# example 9
library(BondValuation)
YtM.by.DCC <- data.frame(CP = 105,
+ SETT1 = rep(as.Date("2020-09-28"), 16),
+ SETT2 = rep(as.Date("2023-03-30"), 16),
+ SETT3 = rep(as.Date("2024-01-15"), 16),
+ Em = rep(as.Date("2019-10-31"), 16),
+ Mat = rep(as.Date("2024-02-29"), 16),
+ CpY = rep(2, 16),
+ FIPD = rep(as.Date("2020-03-30"), 16),
+ LIPD = rep(as.Date("2023-03-30"), 16),
+ FIAD = rep(as.Date("2019-10-31"), 16),
+ RV = rep(100, 16),
+ Coup = rep(10, 16),
+ DCC = seq(1, 16),
+ EOM = rep(0, 16))
Suffix <- c("S1","S2","S3")
i<-1
for (i in c(1:3)) {
+ BondValYield.Output<-suppressWarnings(
+ apply(YtM.by.DCC[,c("CP","SETT",i),"Em","CpY","FIPD",
+ "LIPD","FIAD","RV","Coup","DCC")],
+ 1,function(y) BondVal.Yield(y[1],y[2],y[3],y[4],y[5],y[6],y[7],
+ y[8],y[9],y[10],y[11])))
+ YtM<-do.call(rbind,lapply(lapply(BondValYield.Output,
+ function(y) round(y[,4], 3))))
+ ModDUR<-do.call(rbind,lapply(lapply(BondValYield.Output,
+ function(y) round(y[,5], 4))))
+ Conv<-do.call(rbind,lapply(lapply(BondValYield.Output,
+ function(y) round(y[,7], 4))))
+ YtM.by.DCC<-cbind(YtM.by.DCC,YtM,ModDUR,Conv)
+ names(YtM.by.DCC)[
+ c(ncol(YtM.by.DCC) - 2) : ncol(YtM.by.DCC)] <- c("YtM",Suffix[i],"DUR",Suffix[i],
+ "Conv",Suffix[i])
+ rm(BondValYield.Output,YtM,ModDUR,Conv)
+ }
> print(YtM.by.DCC[,c(13,15:ncol(YtM.by.DCC))], row.names = FALSE)
```

<table>
<thead>
<tr>
<th>DCC</th>
<th>YtM.S1</th>
<th>DUR.S1</th>
<th>Conv.S1</th>
<th>YtM.S2</th>
<th>DUR.S2</th>
<th>Conv.S2</th>
<th>YtM.S3</th>
<th>DUR.S3</th>
<th>Conv.S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.244</td>
<td>2.7593</td>
<td>4.9777</td>
<td>4.360</td>
<td>0.8824</td>
<td>0.7786</td>
<td>-27.035</td>
<td>0.1277</td>
<td>0.0163</td>
</tr>
<tr>
<td>2</td>
<td>8.252</td>
<td>2.7590</td>
<td>4.9766</td>
<td>4.334</td>
<td>0.8825</td>
<td>0.7788</td>
<td>-26.956</td>
<td>0.1279</td>
<td>0.0164</td>
</tr>
<tr>
<td>3</td>
<td>8.245</td>
<td>2.7595</td>
<td>4.9793</td>
<td>4.360</td>
<td>0.8833</td>
<td>0.7803</td>
<td>-26.899</td>
<td>0.1282</td>
<td>0.0164</td>
</tr>
<tr>
<td>4</td>
<td>8.238</td>
<td>2.7604</td>
<td>4.9813</td>
<td>4.339</td>
<td>0.8824</td>
<td>0.7787</td>
<td>-27.002</td>
<td>0.1279</td>
<td>0.0164</td>
</tr>
<tr>
<td>5</td>
<td>8.251</td>
<td>2.7564</td>
<td>4.9676</td>
<td>4.313</td>
<td>0.8792</td>
<td>0.7730</td>
<td>-27.373</td>
<td>0.1265</td>
<td>0.0160</td>
</tr>
<tr>
<td>6</td>
<td>8.251</td>
<td>2.7564</td>
<td>4.9676</td>
<td>4.313</td>
<td>0.8792</td>
<td>0.7730</td>
<td>-27.373</td>
<td>0.1265</td>
<td>0.0160</td>
</tr>
<tr>
<td>7</td>
<td>8.251</td>
<td>2.7564</td>
<td>4.9676</td>
<td>4.313</td>
<td>0.8792</td>
<td>0.7730</td>
<td>-27.373</td>
<td>0.1265</td>
<td>0.0160</td>
</tr>
<tr>
<td>8</td>
<td>8.252</td>
<td>2.7584</td>
<td>4.9746</td>
<td>4.329</td>
<td>0.8817</td>
<td>0.7774</td>
<td>-26.568</td>
<td>0.1293</td>
<td>0.0167</td>
</tr>
<tr>
<td>9</td>
<td>8.251</td>
<td>2.7564</td>
<td>4.9676</td>
<td>4.313</td>
<td>0.8792</td>
<td>0.7730</td>
<td>-27.373</td>
<td>0.1265</td>
<td>0.0160</td>
</tr>
<tr>
<td>10</td>
<td>8.248</td>
<td>2.7587</td>
<td>4.9763</td>
<td>4.365</td>
<td>0.8822</td>
<td>0.7784</td>
<td>-26.973</td>
<td>0.1279</td>
<td>0.0164</td>
</tr>
<tr>
<td>11</td>
<td>8.243</td>
<td>2.7684</td>
<td>4.9824</td>
<td>4.354</td>
<td>0.8845</td>
<td>0.7823</td>
<td>-26.825</td>
<td>0.1286</td>
<td>0.0165</td>
</tr>
<tr>
<td>12</td>
<td>8.381</td>
<td>2.7505</td>
<td>4.9554</td>
<td>4.498</td>
<td>0.8812</td>
<td>0.7765</td>
<td>-26.824</td>
<td>0.1279</td>
<td>0.0163</td>
</tr>
<tr>
<td>13</td>
<td>8.120</td>
<td>2.7645</td>
<td>4.9882</td>
<td>4.183</td>
<td>0.8802</td>
<td>0.7748</td>
<td>-27.521</td>
<td>0.1265</td>
<td>0.0160</td>
</tr>
<tr>
<td>14</td>
<td>8.236</td>
<td>2.7593</td>
<td>4.9776</td>
<td>4.322</td>
<td>0.8826</td>
<td>0.7795</td>
<td>-26.983</td>
<td>0.1279</td>
<td>0.0164</td>
</tr>
<tr>
<td>15</td>
<td>8.274</td>
<td>2.7570</td>
<td>4.9722</td>
<td>4.391</td>
<td>0.8820</td>
<td>0.7780</td>
<td>-26.943</td>
<td>0.1279</td>
<td>0.0164</td>
</tr>
<tr>
<td>16</td>
<td>8.032</td>
<td>2.7729</td>
<td>5.0104</td>
<td>4.175</td>
<td>0.8803</td>
<td>0.7750</td>
<td>-26.038</td>
<td>0.1314</td>
<td>0.0173</td>
</tr>
</tbody>
</table>

This provides the theoretical background on the implemented key figures.
Application of the package BondValuation

This section demonstrates how the R package BondValuation can be applied for the analysis of large data frames. For this purpose, the two sample data frames, SomeBonds2016 and PanelSomeBonds2016, are used. SomeBonds2016 contains time-invariant information of 100 hypothetical bonds. PanelSomeBonds2016 provides daily clean prices and yields of the same bonds in long format.

Checking the data with AnnivDates()

Since erroneous entries in the data are often an issue, the function AnnivDates() performs several plausibility checks. Example 10 provides a summary of SomeBonds2016 and illustrates a strategy for error identification in this data frame.

```r
> # example 10
> library(BondValuation)
> summary(SomeBonds2016)
>
ID.No  Coup.Type  Issue.Date  FIAD.Input
Min. : 1.00  Length:100  Min. :2016-01-01  Min. :2016-01-01
1st Qu.: 25.75  Class :character  1st Qu.:2016-04-25  1st Qu.:2016-04-25
Median : 50.50  Mode :character  Median :2016-06-12  Median :2016-06-12
Mean : 50.50  Mean :2016-06-19  Mean :2016-06-19
3rd Qu.: 75.25  3rd Qu.:2016-08-23  3rd Qu.:2016-08-23
Max. :100.00  Max. :2016-10-28  Max. :2016-10-28

The summary information above reveals that all bonds in the data frame were issued (Issue.Date) and started to accrue interest (FIAD.Input) in 2016. The terms to maturity (Mat.Date) span from about 1 to approximately 40 years. The summary of variable CpY.Input shows that there are no zero coupon bonds in the dataset and the number of interest payments per year varies from 1 to 12. Nominal interest rates (Coup.Input) average 2.27%, varying from 0.01% to 24.02%. All bonds are redeemed (RV.Input) at 100% of their respective par values and 79% of them follow the End-of-Month rule (EOM.Input). Now AnnivDates() is used to analyze the data for plausibility.

> # example 10: continued (I)
> # Applying AnnivDates() to the data frame SomeBonds2016.
> FullAnalysis<-suppressWarnings(
+  apply(
+    SomeBonds2016[,c('Issue.Date', 'Mat.Date', 'CpY.Input', 'FIPD.Input', 'LIPD.Input', 'FIAD.Input', 'Coup.Input', 'DCC.Input', 'EOM.Input')], 1,
+    function(y) AnnivDates(y[1], y[2], y[3], y[4], y[5], y[6], y[7], y[8], y[9], y[10])
+  )
+ )
>
> # Extracting the data frame Warnings and binding the Warnings to the bonds
> BondsWithWarnings<-cbind(
>  SomeBonds2016, do.call(
>    rbind, lapply(FullAnalysis, `[[`, 1)
>  )
> )
```
The summary information in example 10: continued (I) reveals that 1% of the bonds suffer from a chronological error (ChronErrorFlag) and 9% feature inconsistencies between the coupon payment dates and the number of interest payment dates per year CpY (IPD_CpY_Corrupt). To illustrate the rationale behind the plausibility analysis, a manual inspection of the affected bonds is performed below.

```r
> # example 10: continued (II)
> # manual examination of the rows where ChronErrorFlag = 1
> print(BondsWithWarnings[
+    which(BondsWithWarnings$ChronErrorFlag == 1),
+    c("ID.No", "Issue.Date", "FIAD.Input", "FIPD.Input", "LIPD.Input", "Mat.Date"),
+    row.names = FALSE)
ID.No Issue.Date FIAD.Input FIPD.Input LIPD.Input Mat.Date
1 2016-08-23 2016-08-23 2017-08-23 2017-08-23 2017-08-23

The chronological error occurred because the provided penultimate coupon date (LIPD.Input) is located prior to the supplied first interest payment date (FIPD.Input). Since the authenticity of FIPD.Input and LIPD.Input is unclear in this case, both are automatically dropped by AnnivDates(), and the calculation continues based upon the provided values of Issue.Date and Mat.Date.

As can be seen in the manual examination of the rows where IPD_CpY_Corrupt = 1, for all of them, there are inconsistencies between the value of CpY.Input and the interval between FIPD.Input and LIPD.Input. In the first row, for example, CpY.Input indicates that coupons are paid quarterly.

---

6 Please refer to the package manual of BondValuation for detailed descriptions of the other warning flags.
the value of FIPD. Input is correct, coupon payments should occur on July 15th, October 15th, January 15th, and April 15th. If the value of LIPD. Input is genuine, however, interest should be paid on May 15th, August 15th, November 15th, and February 15th. Finally, in this case, FIPD. Input and LIPD. Input can both be correct, lying on each other’s anniversary dates for a value of CpY. Input = 6. Since it is not clear which of the three values is correct, AnnivDates() cannot automatically revise the input but only helps the user to identify the inconsistency. If the data are passed to AnnivDates() as they are, CpY. Input is assumed to be genuine, FIPD. Input and LIPD. Input are dropped, and the execution continues as if they were not provided in the first place, resulting in the temporal structure and cash flows provided below in example 10: continued (III).

> # example 10: continued (III)
> # Printing data frame DateVectors for bond with ID.No = 2
> print(
  + as.data.frame(do.call(rbind, lapply(FullAnalysis, `[[`, 3)[2])),
  + row.names = FALSE)

<table>
<thead>
<tr>
<th>RealDates</th>
<th>RD_indexes</th>
<th>CoupDates</th>
<th>CD_indexes</th>
<th>AnnivDates</th>
<th>AD_indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-06-23</td>
<td>0.08888889</td>
<td>2016-09-15</td>
<td>1</td>
<td>2016-06-15</td>
<td>0</td>
</tr>
<tr>
<td>2016-09-15</td>
<td>1.00000000</td>
<td>2016-12-15</td>
<td>2</td>
<td>2016-09-15</td>
<td>1</td>
</tr>
<tr>
<td>2016-12-15</td>
<td>2.00000000</td>
<td>2017-03-15</td>
<td>3</td>
<td>2016-12-15</td>
<td>2</td>
</tr>
<tr>
<td>2017-03-15</td>
<td>3.00000000</td>
<td>2017-06-15</td>
<td>4</td>
<td>2017-03-15</td>
<td>3</td>
</tr>
<tr>
<td>2017-06-15</td>
<td>4.00000000</td>
<td>2017-09-15</td>
<td>5</td>
<td>2017-06-15</td>
<td>4</td>
</tr>
<tr>
<td>2017-09-15</td>
<td>5.00000000</td>
<td>2017-12-15</td>
<td>6</td>
<td>2017-09-15</td>
<td>5</td>
</tr>
<tr>
<td>2017-12-15</td>
<td>6.00000000</td>
<td>2018-03-15</td>
<td>7</td>
<td>2017-12-15</td>
<td>6</td>
</tr>
<tr>
<td>2018-12-15</td>
<td>10.00000000</td>
<td>2019-03-15</td>
<td>11</td>
<td>2018-12-15</td>
<td>10</td>
</tr>
</tbody>
</table>

> > # Printing data frame PaySched for bond with ID.No = 2
> > print(
  + as.data.frame(do.call(rbind, lapply(FullAnalysis, `[[`, 4)[2])),
  + row.names = FALSE)

<table>
<thead>
<tr>
<th>CoupDates</th>
<th>CoupPayments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-09-15</td>
<td>0.7368611</td>
</tr>
<tr>
<td>2016-12-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2017-03-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2017-06-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2017-09-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2017-12-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2018-03-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2018-06-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2018-09-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2018-12-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2019-03-15</td>
<td>0.8087500</td>
</tr>
<tr>
<td>2019-06-15</td>
<td>0.8087500</td>
</tr>
</tbody>
</table>

The consequences of the plausibility-check-induced automated data revision by AnnivDates() are stored in the data frame Traits. Alongside the values that were initially provided and that are actually used in the subsequent calculations, Traits contains information on the types and lengths of the first and final coupon periods. Example 10: continued (IV) demonstrates how the data frame Traits can be extracted from the output of AnnivDates() and provides summary information on the lengths and types of the first and final coupon periods in the data frame SomeBonds2016. Of the 100 bonds in SomeBonds2016, only 20 have regular first coupon periods and 28 feature final coupon periods of regular length. The lengths of the first coupon periods vary from 1.37% to 1,200% of the bond-specific regular coupon period length, while the final coupon periods average 230% and span from 2.78% to 1,200% of the respective bond’s regular coupon period length.

> # example 10: continued (IV)
> # Extracting the data frame Warnings and binding the Warnings to the bonds
> BondsWithTraits<-cbind(
  + SomeBonds2016, do.call(
    + rbind, lapply(FullAnalysis, `[[`, 2)
  + )
)
> summary(BondsWithTraits[, c('FCPType', 'LCPType', 'FCPLength', 'LCPLength')])

<table>
<thead>
<tr>
<th></th>
<th>FCPType</th>
<th>LCPType</th>
<th>FCPLength</th>
<th>LCPLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>49</td>
<td>51</td>
<td>Min. 0.01366</td>
<td>Min. 0.02778</td>
</tr>
<tr>
<td>short</td>
<td>31</td>
<td>regular:28</td>
<td>1st Qu. 0.92150</td>
<td>1st Qu. 1.00000</td>
</tr>
<tr>
<td>regular</td>
<td>20</td>
<td>short:21</td>
<td>Median 1.00000</td>
<td>Median 1.25140</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean 2.19291</td>
<td>Mean 2.38417</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3rd Qu. 3.00000</td>
<td>3rd Qu. 3.00000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Max. 12.00000</td>
<td>Max. 12.00000</td>
</tr>
</tbody>
</table>

Applying BondVal.Yield() to long format data

In addition to the time-invariant information in SomeBonds2016, the data frame PanelSomeBonds2016 provides daily clean prices (CP.Input) and yields to maturity (YtM.Input) that correspond to the trade dates, TradeDate, and settlement dates, SETT. TradeDate is the calendar date on which the transaction is initiated and the quoted clean price is observed; SETT is the actual calendar date on which the transfer of cash and assets is completed. The settlement date is used for the following computation. Example 11 below shows that PanelSomeBonds2016 has 12,718 rows and 16 columns and provides summary information regarding the time-variant variables. The clean prices span from 90.38% to 224.16%, while the yields to maturity average −0.01593%, varying from −1.725% to 2%.

> # example 11
> library(BondValuation)
> dim(PanelSomeBonds2016)
> summary(PanelSomeBonds2016[, c(13:16)])

<table>
<thead>
<tr>
<th>TradeDate</th>
<th>SETT</th>
<th>CP.Input</th>
<th>YtM.Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>2016-01-29</td>
<td>Min. 90.38</td>
<td>Min. -1.72500</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>2016-08-03</td>
<td>1st Qu. 102.73</td>
<td>1st Qu. -0.35000</td>
</tr>
<tr>
<td>Median</td>
<td>2016-10-03</td>
<td>Median 105.79</td>
<td>Median -0.02500</td>
</tr>
<tr>
<td>Mean</td>
<td>2016-09-20</td>
<td>Mean 112.53</td>
<td>Mean -0.01593</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>2016-11-17</td>
<td>3rd Qu. 113.21</td>
<td>3rd Qu. 0.27500</td>
</tr>
<tr>
<td>Max.</td>
<td>2016-12-30</td>
<td>Max. 224.16</td>
<td>Max. 2.00000</td>
</tr>
</tbody>
</table>

In the following, example 12, the function BondVal.Yield() is used to determine τ, accrued interest, dirty price, yield to maturity, modified duration, Macaulay duration, and convexity for each bond and settlement date in PanelSomeBonds2016. In alternative 1, the function BondVal.Yield() is applied to every row of the data frame PanelSomeBonds. Alternative 2 demonstrates a significantly faster approach, where AnnivDates() is applied to every bond’s time-invariant characteristics before its output is passed to BondVal.Yield() for every settlement date. Alternative 2 takes less than half the time of alternative 1.

> # example 12
> # analysis of PanelSomeBonds2016 with BondValuation
> library(BondValuation)
> Panel <- PanelSomeBonds2016
> Vars <- c("tau","AccrInt","DP","YtM","ModDUR","MacDUR","Conv")
> Panel[, c((ncol(Panel) + 1) : (ncol(Panel) + length(Vars)))] <- as.numeric(NA)
> names(Panel)[(ncol(Panel) - length(Vars) + 1) : ncol(Panel)] <- Vars
>
> # Alternative 1: loop through the data frame
> # applying BondVal.Yield to each row
> Time.Alt01 <- system.time(
+ for (i in c(1:nrow(Panel))) {
+   BondVal.Out <- suppressWarnings(
+     BondVal.Yield(CP = Panel$CP.Input[i],
+                   SETT = Panel$SETT[i],
+                   Em = Panel$Issue.Date[i],
+                   Mat = Panel$Mat.Date[i],
+                   CpY = Panel$CpY.Input[i],
+                   FIPD = Panel$FIPD.Input[i],
+                   LIPD = Panel$LIPD.Input[i],
+       )
+     Panel$tau[i] <- BondVal.Out$tau
+     Panel$AccrInt[i] <- BondVal.Out$AccrInt
+     Panel$DP[i] <- BondVal.Out$DP
+     Panel$YtM[i] <- BondVal.Out$YtM
+     Panel$ModDUR[i] <- BondVal.Out$ModDUR
+     Panel$MacDUR[i] <- BondVal.Out$MacDUR
+     Panel$Conv[i] <- BondVal.Out$Conv
+   }
+ )

On an "Intel(R) Core(TM) i7-3687U CPU @ 2.10GHz" machine, alternative 1 takes about 350 seconds, while alternative 2 takes ca. 170 seconds.
\begin{verbatim}
+ FIAD = Panel$FIAD.Input[i],
+ RV = Panel$RV.Input[i],
+ Coup = Panel$Coup.Input[i],
+ DCC = Panel$DCC.Input[i],
+ EOM = Panel$EOM.Input[i],
+ Precision = .Machine$double.eps^0.5
+ )
+ )
+ Panel[i, c((ncol(Panel) - length(Vars) + 1) : ncol(Panel))] <-
+ round(as.numeric(BondVal.Out[c(11, 2 : 7)]), 4)
+ }, gcFirst = TRUE
+
+>
+>
+> # Alternative 2: Run AnnivDates() once per Bond-ID and pass its output
+> # to BondVal.Yield() for every row with the same Bond-ID
+> NonDuplID <- c(which(!duplicated(Panel$ID.No)), (nrow(Panel)+1))
+> Time.Alt02 <- system.time(
+ + for (i in c(1 : (length(NonDuplID) - 1))) {
+ + BondCount <- NonDuplID[i]
+ + AnnivDates.Out <- suppressWarnings(
+ + AnnivDates.Em = Panel$Issue.Date[BondCount],
+ + Mat = Panel$Mat.Date[BondCount],
+ + CpY = Panel$CpY.Input[BondCount],
+ + FIPD = Panel$FIPD.Input[BondCount],
+ + LIPD = Panel$LIPD.Input[BondCount],
+ + FIAD = Panel$FIAD.Input[BondCount],
+ + RV = Panel$RV.Input[BondCount],
+ + Coup = Panel$Coup.Input[BondCount],
+ + DCC = Panel$DCC.Input[BondCount],
+ + EOM = Panel$EOM.Input[BondCount]
+ + )
+ + }
+ + for (j in c(NonDuplID[i] : (NonDuplID[i + 1] - 1))) {
+ + BondVal.Out <- suppressWarnings(
+ + BondVal.Yield(CP = Panel$CP.Input[j],
+ + SETT = Panel$SETT[j],
+ + Em = Panel$Issue.Date[j],
+ + Mat = Panel$Mat.Date[j],
+ + CpY = Panel$CpY.Input[j],
+ + FIPD = Panel$FIPD.Input[j],
+ + LIPD = Panel$LIPD.Input[j],
+ + FIAD = Panel$FIAD.Input[j],
+ + RV = Panel$RV.Input[j],
+ + Coup = Panel$Coup.Input[j],
+ + DCC = Panel$DCC.Input[j],
+ + EOM = Panel$EOM.Input[j],
+ + InputCheck = 0,
+ + Precision = .Machine$double.eps^0.5,
+ + AnnivDatesOutput = AnnivDates.Out
+ + )
+ + }
+ + Panel[j, c((ncol(Panel) - length(Vars) + 1) : ncol(Panel))] <-
+ + round(as.numeric(BondVal.Out[c(11, 2 : 7)]), 4)
+ + }
+ + }, gcFirst = TRUE
+ + )
+>
+round(Time.Alt02[[3]] / Time.Alt01[[3]], 2)
+ [1] 0.48
\end{verbatim}
Conclusion

This article introduces the R package BondValuation and provides guidance on its application for analysis of large data frames of fixed coupon bonds. The theoretical foundation of the package is the generalized valuation methodology developed by ?. Its seamless implementation in BondValuation is framed by a set of routines that assist the user in data quality evaluation and automatically correct corrupted entries.

BondValuation is the first R package that properly handles irregular first and final coupon periods of fixed coupon bonds and provides a comprehensive coverage of the day count conventions (DCC) used in the global bond markets. Currently, 16 different DCCs are implemented, which account for the vast majority of the methods used in the global fixed income markets. Within its scope, the R package BondValuation performs correctly and efficiently. Nevertheless, the current version of the software remains open for further development and refinement. Essentially, the calculations are performed under the assumption that interest accrual and temporal structure follow the same DCC. The option CalcMethod in the functions BondVal.Price() and BondVal.Yield() can be used to force the temporal structure to follow the ACT/ACT (ICMA) method, while the DCC passed to the respective function is used to compute accrued interest. In future versions of the package, I intend to implement an explicit assignment of DCC to both interest accrual and temporal structure, which will increase the flexibility of the package.

A further limitation is that the calendar dates of the temporal structure are currently returned, regardless of whether or not they are business days. Although this is the common approach in theoretical bond valuation, including the possibility of business day adjustments for cash flows would be particularly appealing to practitioners. Along with business day adjustments, future versions of BondValuation can be extended by methods for bond portfolio analysis.

The current version of BondValuation is designed for processing non-callable, option-free, non-sinkable fixed coupon bonds and zero bonds. With the implemented methods, callable bonds can be analyzed through appropriate adjustment of the maturity date to the next call date, returning the so-called yield-to-worst and the corresponding duration and convexity measures. Based on the implemented functions, the R package BondValuation can be extended to incorporate methods for explicit treatment of callable, sinkable and convertible fixed and floating rate bonds.

The R package BondValuation provides the computational foundation for the exploration of a variety of interesting research questions related to the analysis of fixed income securities across markets. Even considering the limitations described above, the software is also useful to practitioners. I intend to continuously extend and improve the package, and I highly appreciate feedback from the users.

Acknowledgments

I would like to thank Ingo Geishecker for our frequent discussions and his profound advice. I am also grateful to Karl Ludwig Keiber and Philipp Otto for their helpful comments and suggestions, and to Inna Keil for her excellent research assistance. All remaining errors are my own responsibility.

Bibliography


Wadim Djatschenko
European University Viadrina
Frankfurt (Oder)
Germany
ORCID: 0000-0003-0653-8779
wadim.djatschenko@gmx.de