

- J. A. John and N. R. Draper. An alternative family of transformations. *Journal of the Royal Statistical Society C*, 29(2):190–197, 1980. URL <https://www.jstor.org/stable/2986305>. [p]
- N. L. Johnson. Systems of frequency curves generated by methods of translation. *Biometrika*, 36(1/2): 149–176, 1949. URL <http://www.jstor.org/stable/2332539>. [p]
- D. M. Kelmansky, E. J. Martínez, and V. Leiva. A new variance stabilizing transformation for gene expression data analysis. *Statistical Applications in Genetics and Molecular Biology*, 12(6):653–666, 2013. URL <https://doi.org/10.1515/sagmb-2012-0030>. [p]
- A. Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giornale dell'Istituto Italiano degli Attuari*, 4(1):1–11, 1933. [p]
- M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5): 1–26, 2008. URL <https://www.jstatsoft.org/article/view/v028i05>. [p]
- S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22 (1):79–86, 1951. URL <https://www.jstor.org/stable/2236703>. [p]
- S. Mangiafico. *Rcompanion: Functions to Support Extension Education Program Evaluation*, 2019. URL <https://CRAN.R-project.org/package=rcompanion>. R package version 3.3. [p]
- B. F. J. Manly. Exponential data transformations. *Journal of the Royal Statistical Society D*, 25(1):37–42, 1976. URL <https://www.jstor.org/stable/2988129>. [p]
- L. Medina, P. Castro, A.-K. Kreutzmann, and N. Rojas-Perilla. *Trafo: Estimation, Comparison and Selection of Transformations*, 2018. URL <https://CRAN.R-project.org/package=trafo>. R package version 1.0.0. [p]
- I. Molina and Y. Marhuenda. Sae: An R package for small area estimation. *The R Journal*, 7(1):81–98, 2015. URL <https://journal.r-project.org/archive/2015/RJ-2015-007>. [p]
- M. Morozova, K. Koschutnig, E. Klein, and G. Wood. Monotonic non-linear transformations as a tool to investigate age-related effects on brain white matter integrity: A Box-Cox investigation. *NeuroImage*, 125:1119–1130, 2016. URL <https://doi.org/10.1016/j.neuroimage.2015.08.003>. [p]
- M. Mosimann, L. Frossard, M. Keiler, R. Weingartner, and A. Zischg. A robust and transferable model for the prediction of flood losses on household contents. *Multidisciplinary Digital Publishing Institute*, 10(11):1596, 2018. URL <https://doi.org/10.3390/w10111596>. [p]
- J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society A*, 135(3):370–384, 1972. URL <https://www.jstor.org/stable/2344614>. [p]
- J. Pek, O. Wong, and C. M. Wong. Data transformations for inference with linear regression: Clarifications and recommendations. *Practical Assessment, Research and Evaluation*, 22(9):1–11, 2017. [p]
- R. A. Peterson. *bestNormalize: Normalizing Transformation Functions*, 2019. URL <https://CRAN.R-project.org/package=bestNormalize>. R package version 3.1. [p]
- P. J. Ribeiro Jr. and P. J. Diggle. geoR: Analysis of geostatistical data. *R News*, 1(2):15–18, 2016. URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.3971&rep=rep1&type=pdf>. [p]
- D. M. Rocke and B. Durbin. A model for measurement error for gene expression arrays. *Journal of Computational Biology*, 8(6):557–569, 2001. URL <https://doi.org/10.1089/106652701753307485>. [p]
- N. Rojas-Perilla. *The Use of Data-Driven Transformations and Their Application in Small Area Estimation*. PhD thesis, Freie Universität Berlin, 2018. URL <http://dx.doi.org/10.17169/refubium-1006>. [p]
- P. Royston, P. C. Lambert, and others. *Flexible Parametric Survival Analysis Using Stata: Beyond the Cox Model*. Stata Press, College Station, 2011. [p]
- S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality. *Biometrika*, 52(3/4):591–611, 1965. URL <https://www.jstor.org/stable/2333709>. [p]
- H. shyong Chang. Functional forms and the demand for meat in the United States. *The Review of Economics and Statistics*, 59(3):355–359, 1977. URL <https://www.jstor.org/stable/1925054>. [p]
- N. Smirnov. Sur les écarts de la courbe de distribution empirique. *Recueil Mathématique (Matematicheskii Sbornik)*, 6(1):3–26, 1939. [p]

- J. M. G. Taylor. Power transformations to symmetry. *Biometrika*, 72(1):145–152, 1985. URL <https://doi.org/10.1093/biomet/72.1.1450>. [p]
- J. W. Tukey. On the comparative anatomy of transformations. *Ann. Math. Statist.*, 28(3), 1957. URL <https://www.jstor.org/stable/2237223>. [p]
- W. J. Vaughan, C. S. Russell, and M. Hazilla. A note on the use of travel cost models with unequal zonal populations: Comment. *Land Economics*, 58(3):400–407, 1982. URL <https://www.jstor.org/stable/3145948>. [p]
- W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, New York, 2002. URL [10.1007/978-0-387-21706-2](https://doi.org/10.1007/978-0-387-21706-2). [p]
- Y. Wang. *Jtrans: Johnson Transformation for Normality*, 2015. URL <https://CRAN.R-project.org/package=jtrans>. R package version 0.2.1. [p]
- J. Whittaker, C. Whitehead, and M. Somers. The neglog transformation and quantile regression for the analysis of a large credit scoring database. *Journal of the Royal Statistical Society C*, 54(4):863–878, 2005. URL [doi:10.1111/j.1467-9876.2005.00520.x](https://doi.org/10.1111/j.1467-9876.2005.00520.x). [p]
- Z. Yang. A modified family of power transformations. *Economics Letters*, 92(1):14–19, 2006. URL [doi:10.1016/j.econlet.2006.01.011](https://doi.org/10.1016/j.econlet.2006.01.011). [p]
- I.-K. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000. URL <https://www.jstor.org/stable/2673623>. [p]
- P. Zarembka. Functional form in the demand for money. *Journal of the American Statistical Association*, 63(322):502–511, 1968. URL <https://www.jstor.org/stable/2284021>. [p]
- P. Zarembka. Transformation of variables in econometrics. In P. Macmillan, editor, *The New Palgrave Dictionary of Economics*, pages 1–3. Palgrave Macmillan, London, 1974. URL [https://doi.org/10.1057/978-1-349-95121-5\\_1882-1](https://doi.org/10.1057/978-1-349-95121-5_1882-1). [p]
- R. F. Ziemer, W. N. Musser, and R. C. Hill. Recreation demand equations: Functional form and consumer surplus. *American Journal of Agricultural Economics*, 62(1):136–141, 1980. URL <https://www.jstor.org/stable/1239482>. [p]

Lily Medina  
Freie Universität Berlin  
Garystraße 21, 14195 Berlin  
Germany  
[lilymiru@gmail.com](mailto:lilymiru@gmail.com)

Ann-Kristin Kreutzmann  
Freie Universität Berlin  
Garystraße 21, 14195 Berlin  
Germany  
[Ann-Kristin.Kreutzmann@fu-berlin.de](mailto:Ann-Kristin.Kreutzmann@fu-berlin.de)

Natalia Rojas-Perilla  
Freie Universität Berlin  
Garystraße 21, 14195 Berlin  
Germany  
[natalia.rojas@fu-berlin.de](mailto:natalia.rojas@fu-berlin.de)

Piedad Castro  
Freie Universität Berlin  
Garystraße 21, 14195 Berlin  
Germany  
[piedad@madmath.co](mailto:piedad@madmath.co)

# 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 [Djatschenko \(2019\)](#).

## 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.<sup>1</sup>

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 [Djatschenko \(2019\)](#) 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.

[Djatschenko \(2019\)](#) 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, [Djatschenko \(2019\)](#) 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

<sup>1</sup>[Djatschenko \(2019\)](#) provides a comprehensive overview of most day count conventions currently used in the global bond markets and demonstrates their interactions with irregular coupon periods.

in the isolation of each DCC-dependent parameter, resulting in a modular structure that can easily integrate any conceivable DCC. In addition, [Djatschenko \(2019\)](#) 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 [Djatschenko \(2019\)](#) 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 R package **BondValuation** consists of five functions, `AccrInt()`, `AnnivDates()`, `BondVal.Price()`, `BondVal.Yield()`, and `DP()`, and four data frames, `List.DCC`, `NonBusDays.Brazil`, `PanelSomeBonds2016`, `SomeBonds2016`.

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**<sup>2</sup>:

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

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

<sup>2</sup>[Djatschenko \(2019\)](#) 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<sup>3</sup>, *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`.

```
> # 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)
```

DCC.Name	Start	End	Days_Accrued	Accrued_Interest
ACT/ACT (ISDA)	2011-08-31	2012-02-29	182	2.615490
ACT/ACT (ICMA)	2011-08-31	2012-02-29	182	2.625000
ACT/ACT (AFB)	2011-08-31	2012-02-29	182	2.617808
ACT/365L	2011-08-31	2012-02-29	182	2.610656
30/360	2011-08-31	2012-02-29	179	2.610417
30E/360	2011-08-31	2012-02-29	179	2.610417
30E/360 (ISDA)	2011-08-31	2012-02-29	180	2.625000
30/360 (German)	2011-08-31	2012-02-29	180	2.625000
30/360 US	2011-08-31	2012-02-29	179	2.610417
ACT/365 (Fixed)	2011-08-31	2012-02-29	182	2.617808
ACT(NL)/365	2011-08-31	2012-02-29	182	2.617808
ACT/360	2011-08-31	2012-02-29	182	2.654167
30/365	2011-08-31	2012-02-29	179	2.574658
ACT/365 (Canadian Bond)	2011-08-31	2012-02-29	182	2.625000
ACT/364	2011-08-31	2012-02-29	182	2.625000
BusDay/252 (Brazilian)	2011-08-31	2012-02-29	124	2.549769

### 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`.<sup>4</sup> 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`.

<sup>3</sup>See manual to the R package **BondValuation** for details on implementation and [Krgin \(2002\)](#) for the theoretical background of the End-of-Month rule.

<sup>4</sup>See [Djatschenko \(2019\)](#) 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.

```
> # example 3
> library(BondValuation)
> AnnivDates(as.Date("2019-05-31"), "2021-07-31")
$`Warnings`
Em_FIAD_differ      EmMatMissing      CpYOverride      RV_set100percent      NegLifeFlag
              0              0              1              1              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
1970-01-01      2      <NA>      2019-05-31      2019-05-31      <NA>
FIPD_Orig      est_FIPD      LIPD      LIPD_Orig      est_LIPD      Mat
<NA>      2019-07-31      <NA>      <NA>      2021-01-31      2021-07-31
Refer      FCPTType      FCPLength      LCPTType      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()`:

```
> # 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)$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	2020-08-31	2
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)$DateVectors
```

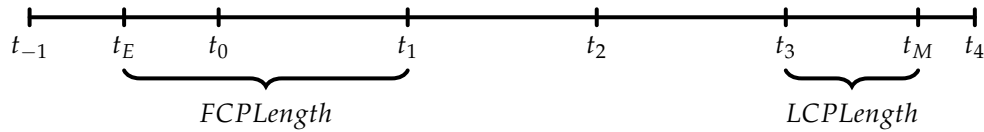
	RealDates	RD_indexes	CoupDates	CD_indexes	AnnivDates	AD_indexes
1	2019-05-31	-0.4945055	2020-02-29	1.000000	2019-02-28	-1
2	2020-02-29	1.0000000	2020-08-29	2.000000	2019-08-29	0
3	2020-08-29	2.0000000	2021-02-28	3.000000	2020-02-29	1
4	2021-02-28	3.0000000	2021-07-31	3.840659	2020-08-29	2
5	2021-07-31	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 anniversary dates 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.



**Figure 1:** Timeline illustration of the bonds in examples 4 and 5.

```
> # 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, `[[`, 2), `[[`, 15)
+                           , na.omit)
> FCPLength.EOM0 <- as.data.frame(do.call(rbind, lapply(FCPLength.EOM0, round, 4)))
> LCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[[`, 2), `[[`, 17)
+                           , na.omit)
> LCPLength.EOM0 <- as.data.frame(do.call(rbind, lapply(LCPLength.EOM0, round, 4)))
> TempStruct.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[[`, 3), `[[`, 2)
+                           , na.omit)
> TempStruct.EOM0 <- lapply(TempStruct.EOM0, `length<-`,
+                           max(lengths(TempStruct.EOM0)))
> TempStruct.EOM0 <- as.data.frame(do.call(rbind, lapply(TempStruct.EOM0, round, 4)))
> TempStruct.by.DCC.EOM0 <- cbind(TempStruct.by.DCC, TempStruct.EOM0,
+                                FCPLength.EOM0, LCPLength.EOM0)
> names(TempStruct.by.DCC.EOM0)[c(10:16)] <- c("E", "01", "02", "03", "M",
+                                              "FCPLength", "LCPLength")
> print(TempStruct.by.DCC.EOM0[, c(9:ncol(TempStruct.by.DCC.EOM0))],
+       row.names = FALSE)
```

DCC.Name	E	01	02	03	M	FCPLength	LCPLength
ACT/ACT (ISDA)	-0.4945	1	2	3	3.8407	1.4945	0.8407
ACT/ACT (ICMA)	-0.4945	1	2	3	3.8407	1.4945	0.8407
ACT/ACT (AFB)	-0.4945	1	2	3	3.8407	1.4945	0.8407
ACT/365L	-0.4945	1	2	3	3.8407	1.4945	0.8407
30/360	-0.4862	1	2	3	3.8453	1.4862	0.8453
30E/360	-0.4917	1	2	3	3.8398	1.4917	0.8398
30E/360 (ISDA)	-0.4972	1	2	3	3.8380	1.4972	0.8380
30/360 (German)	-0.4972	1	2	3	3.8380	1.4972	0.8380
30/360 US	-0.4862	1	2	3	3.8453	1.4862	0.8453
ACT/365 (Fixed)	-0.4945	1	2	3	3.8407	1.4945	0.8407
ACT(NL)/365	-0.4945	1	2	3	3.8407	1.4945	0.8407
ACT/360	-0.4945	1	2	3	3.8407	1.4945	0.8407
30/365	-0.4862	1	2	3	3.8453	1.4862	0.8453
ACT/365 (Canadian Bond)	-0.4945	1	2	3	3.8407	1.4945	0.8407
ACT/364	-0.4945	1	2	3	3.8407	1.4945	0.8407
BusDay/252 (Brazilian)	-0.5040	1	2	3	3.8425	1.5040	0.8425



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  $RD\_indexes = \{-0.4862, 1, 2, 3, 3.8453\}$ , 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, `[`, 2), `[`, 15)
+                           , na.omit)
> FCPLength.EOM0 <- as.data.frame(do.call(rbind, lapply(FCPLength.EOM0, round, 4)))
> LCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[`, 2), `[`, 17)
+                           , na.omit)
> LCPLength.EOM0 <- as.data.frame(do.call(rbind, lapply(LCPLength.EOM0, round, 4)))
> TempStruct.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[`, 3), `[`, 2)
+                           , na.omit)
> TempStruct.EOM0 <- lapply(TempStruct.EOM0, `length<-`,
+                           max(lengths(TempStruct.EOM0)))
> TempStruct.EOM0 <- as.data.frame(do.call(rbind, lapply(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	8.8354	0.8294	1.8354
ACT/ACT (ICMA)	0.1703	1	2	3	4	5	6	7	8.8352	0.8297	1.8352
ACT/ACT (AFB)	0.1708	1	2	3	4	5	6	7	8.8375	0.8292	1.8375
ACT/365L	0.1703	1	2	3	4	5	6	7	8.8352	0.8297	1.8352
30/360	0.1667	1	2	3	4	5	6	7	8.8278	0.8333	1.8278
30E/360	0.1667	1	2	3	4	5	6	7	8.8278	0.8333	1.8278
30E/360 (ISDA)	0.1667	1	2	3	4	5	6	7	8.8278	0.8333	1.8278
30/360 (German)	0.1667	1	2	3	4	5	6	7	8.8333	0.8333	1.8333
30/360 US	0.1667	1	2	3	4	5	6	7	8.8278	0.8333	1.8278
ACT/365 (Fixed)	0.1703	1	2	3	4	5	6	7	8.8352	0.8297	1.8352
ACT(NL)/365	0.1713	1	2	3	4	5	6	7	8.8398	0.8287	1.8398
ACT/360	0.1703	1	2	3	4	5	6	7	8.8352	0.8297	1.8352
30/365	0.1667	1	2	3	4	5	6	7	8.8278	0.8333	1.8278
ACT/365 (Canadian Bond)	0.1703	1	2	3	4	5	6	7	8.8352	0.8297	1.8352
ACT/364	0.1703	1	2	3	4	5	6	7	8.8352	0.8297	1.8352
BusDay/252 (Brazilian)	0.1840	1	2	3	4	5	6	7	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.

## 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 [Djatschenko \(2019\)](#) 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`.

```
> # example 7
> library(BondValuation)
> CashFlows.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),
+                                     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', 'CpY', '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)
```

DCC.Name	CF.1	CF.2	CF.3	CF.4	CF.5	CF.6	CF.7	CF.M
ACT/ACT (ISDA)	4.1303	5.0273	4.9519	5.0411	4.9589	5.0411	4.9589	9.2011
ACT/ACT (ICMA)	4.1484	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	9.1758
ACT/ACT (AFB)	4.1257	5.0411	4.9589	5.0411	4.9589	5.0411	4.9589	9.2055
ACT/365L	4.1257	5.0273	4.9589	5.0411	4.9589	5.0411	4.9589	9.1803
30/360	4.1667	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	9.1389
30E/360	4.1667	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	9.1389
30E/360 (ISDA)	4.1667	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	9.1389
30/360 (German)	4.1667	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	9.1667
30/360 US	4.1667	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	9.1389
ACT/365 (Fixed)	4.1370	5.0411	4.9589	5.0411	4.9589	5.0411	4.9589	9.2055

```

      ACT(NL)/365 4.1096 5.0411 4.9589 5.0411 4.9589 5.0411 4.9589 9.2055
      ACT/360 4.1944 5.1111 5.0278 5.1111 5.0278 5.1111 5.0278 9.3333
      30/365 4.1096 4.9315 4.9315 4.9315 4.9315 4.9315 4.9315 9.0137
ACT/365 (Canadian Bond) 4.1370 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1644
      ACT/364 4.1484 5.0549 4.9725 5.0549 4.9725 5.0549 4.9725 9.2308
BusDay/252 (Brazilian) 3.9332 4.8809 4.8809 4.8809 4.8809 4.8809 4.8809 9.0060
>
> # RegCF.equal = 1, \textit{i.e.}, regular cash flows forced to be equal
> print(CashFlows.by.DCC.RegCFequal[, c(11:ncol(CashFlows.by.DCC.RegCFequal))],
+       row.names = FALSE)
      DCC.Name CF.1 CF.2 CF.3 CF.4 CF.5 CF.6 CF.7 CF.M
ACT/ACT (ISDA) 4.1303 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2011
ACT/ACT (ICMA) 4.1484 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1758
ACT/ACT (AFB) 4.1257 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2055
ACT/365L 4.1257 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1803
      30/360 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
      30E/360 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
      30E/360 (ISDA) 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
      30/360 (German) 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1667
      30/360 US 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
ACT/365 (Fixed) 4.1370 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2055
      ACT(NL)/365 4.1096 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2055
      ACT/360 4.1944 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.3333
      30/365 4.1096 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.0137
ACT/365 (Canadian Bond) 4.1370 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1644
      ACT/364 4.1484 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2308
BusDay/252 (Brazilian) 3.9332 4.8809 4.8809 4.8809 4.8809 4.8809 4.8809 9.0060

```

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 [Djatschenko \(2019\)](#). 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(
+   apply(AccrIntDP.by.DCC[,c('CP',paste0('SETT',i),'Em','Mat','CpY','FIPD',
+     'LIPD','FIAD','RV','Coupe','DCC')],
+     1,function(y) DP(y[1],y[2],y[3],y[4],y[5],y[6],y[7],
+       y[8],y[9],y[10],y[11]))
+ AI<-do.call(rbind,lapply(lapply(lapply(DP.Output, `[[`, 2), `[[`, 3), round, 4))
+ DP<-do.call(rbind,lapply(lapply(lapply(DP.Output, `[[`, 2), `[[`, 1), round, 4))
+ AccrIntDP.by.DCC<-cbind(AccrIntDP.by.DCC,AI,DP)
+ names(AccrIntDP.by.DCC)[
+   c((ncol(AccrIntDP.by.DCC) - 1) : ncol(AccrIntDP.by.DCC))] <- c(
+     paste0("AI.", Suffix[i]), paste0("DP.", Suffix[i]))
+ rm(DP.Output,AI,DP)
+ }
> print(AccrIntDP.by.DCC[,c(15:ncol(AccrIntDP.by.DCC))], row.names = FALSE)
      DCC.Name AI.SETT1 DP.SETT1 AI.SETT2 DP.SETT2 AI.SETT3 DP.SETT3
ACT/ACT (ISDA)  4.9727 109.9727      0      105  7.9716 112.9716
ACT/ACT (ICMA)  4.9457 109.9457      0      105  7.9396 112.9396
ACT/ACT (AFB)   4.9863 109.9863      0      105  7.9726 112.9726
ACT/365L        4.9727 109.9727      0      105  7.9508 112.9508
30/360          4.9444 109.9444      0      105  7.9167 112.9167
30E/360         4.9444 109.9444      0      105  7.9167 112.9167
30E/360 (ISDA)  4.9444 109.9444      0      105  7.9167 112.9167
30/360 (German) 4.9444 109.9444      0      105  7.9167 112.9167
30/360 US       4.9444 109.9444      0      105  7.9167 112.9167
ACT/365 (Fixed) 4.9863 109.9863      0      105  7.9726 112.9726
ACT(NL)/365     4.9863 109.9863      0      105  7.9726 112.9726
ACT/360         5.0556 110.0556      0      105  8.0833 113.0833
30/365          4.8767 109.8767      0      105  7.8082 112.8082
ACT/365 (Canadian Bond) 4.9863 109.9863      0      105  7.9315 112.9315
ACT/364         5.0000 110.0000      0      105  7.9945 112.9945
BusDay/252 (Brazilian) 4.8412 109.8412      0      105  7.7354 112.7354

```

### Yield to maturity, duration, and convexity

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

$$DP_{\tau} = CP_{\tau} + AC(t_{\tau}) = \frac{CN(t_{\tau})}{\left(1 + \frac{y}{h}\right)^w} + \sum_{i=1}^{\eta} \frac{CF_{i+k}}{\left(1 + \frac{y}{h}\right)^{w+i}} + \frac{CF_M + RV}{\left(1 + \frac{y}{h}\right)^{w+\eta+z}}. \quad (1)$$

In equation (1),  $DP_{\tau}$  denotes the dirty price, consisting of the quoted clean price  $CP_{\tau}$  and accrued interest  $AC(t_{\tau})$ . Conformal with the notation in [Djatschenko \(2019\)](#),  $t_{\tau}$  is the settlement date and  $\tau$  its index in the temporal structure established by the function `AnnivDates()`. On the right side of equation (1),  $CN(t_{\tau})$  denotes the next coupon payment after the settlement date  $t_{\tau}$ , and  $w$  is the fraction of a regular coupon period left until this payment. The set  $CF_{i+k}$  with  $i \in \{x \in \mathbb{N} \mid x \in [1, \eta]\}$  contains all interest payments after  $t_{\tau}$ , excluding the final coupon payment,  $CF_M$ , where  $k$  is the index of the next coupon date after  $t_{\tau}$ ,  $\eta$  is the number of interest payment dates between  $t_{\tau}$  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  $DP_{\tau}$  and the accrued interest  $AC(\tau)$  are computed as illustrated in *example 8*. The cash flows  $CN(t_{\tau})$ ,  $CF_{i+k}$ , 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 `AnnivDates()`, as shown in *example 6*.

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

The function `BondVal.Price()` can be used to compute a bond's clean price, ( $CP_{\tau}$ ), given its yield

to maturity p.a. ( $y$ ), while the function `BondVal.Yield()` returns  $y$  given  $CP_\tau$ . Besides accrued interest ( $AC(t_\tau)$ ) and dirty price ( $DP_\tau$ ), both functions return  $\tau$ , MacAulay duration, modified duration, and convexity of the specified bond.<sup>5</sup> 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.

```
> # 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',paste0('SETT',i),'Em','Mat','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, `[`, 4), round, 3))
+   ModDUR<-do.call(rbind,lapply(lapply(BondValYield.Output, `[`, 5), round, 4))
+   Conv<-do.call(rbind,lapply(lapply(BondValYield.Output, `[`, 7), round, 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(
+       paste0("YtM.", Suffix[i]), paste0("DUR.", Suffix[i]),
+       paste0("Conv.", Suffix[i]))
+   rm(BondValYield.Output,YtM,ModDUR,Conv)
+ }
> print(YtM.by.DCC[,c(13,15:ncol(YtM.by.DCC))], row.names = FALSE)
```

DCC	YtM.S1	DUR.S1	Conv.S1	YtM.S2	DUR.S2	Conv.S2	YtM.S3	DUR.S3	Conv.S3
1	8.244	2.7593	4.9777	4.360	0.8824	0.7786	-27.035	0.1277	0.0163
2	8.252	2.7590	4.9766	4.334	0.8825	0.7788	-26.956	0.1279	0.0164
3	8.245	2.7595	4.9793	4.360	0.8833	0.7803	-26.899	0.1282	0.0164
4	8.238	2.7604	4.9813	4.339	0.8824	0.7787	-27.002	0.1279	0.0164
5	8.251	2.7564	4.9676	4.313	0.8792	0.7730	-27.373	0.1265	0.0160
6	8.251	2.7564	4.9676	4.313	0.8792	0.7730	-27.373	0.1265	0.0160
7	8.251	2.7564	4.9676	4.313	0.8792	0.7730	-27.373	0.1265	0.0160
8	8.252	2.7584	4.9746	4.329	0.8817	0.7774	-26.568	0.1293	0.0167
9	8.251	2.7564	4.9676	4.313	0.8792	0.7730	-27.373	0.1265	0.0160
10	8.248	2.7587	4.9763	4.365	0.8822	0.7784	-26.973	0.1279	0.0164
11	8.243	2.7604	4.9824	4.354	0.8845	0.7823	-26.825	0.1286	0.0165
12	8.381	2.7505	4.9554	4.498	0.8812	0.7765	-26.824	0.1279	0.0163
13	8.120	2.7645	4.9882	4.183	0.8802	0.7748	-27.521	0.1265	0.0160
14	8.236	2.7593	4.9776	4.322	0.8826	0.7789	-26.983	0.1279	0.0164
15	8.274	2.7570	4.9722	4.391	0.8820	0.7780	-26.943	0.1279	0.0164
16	8.032	2.7729	5.0104	4.175	0.8803	0.7750	-26.038	0.1314	0.0173

<sup>5</sup>Djatschenko (2019) 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.

```
> # 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. :2016-10-14	Max. :2016-10-28

FIPD.Input	LIPD.Input	Mat.Date	CpY.Input
Min. :2016-04-24	Min. :2016-08-23	Min. :2017-01-24	Min. : 1.0
1st Qu.:2016-09-30	1st Qu.:2019-02-14	1st Qu.:2019-07-24	1st Qu.: 2.0
Median :2016-12-15	Median :2020-06-08	Median :2020-11-20	Median : 2.5
Mean :2016-12-15	Mean :2021-11-11	Mean :2022-05-18	Mean : 4.3
3rd Qu.:2017-03-02	3rd Qu.:2022-11-04	3rd Qu.:2023-05-05	3rd Qu.: 6.0
Max. :2017-08-23	Max. :2056-05-20	Max. :2056-08-31	Max. :12.0

Coup.Input	RV.Input	DCC.Input	EOM.Input
Min. : 0.010	Min. :100	Min. : 1.00	Min. :0.00
1st Qu.: 0.800	1st Qu.:100	1st Qu.: 5.00	1st Qu.:1.00
Median : 1.410	Median :100	Median :10.00	Median :1.00
Mean : 2.270	Mean :100	Mean : 8.95	Mean :0.79
3rd Qu.: 2.869	3rd Qu.:100	3rd Qu.:13.00	3rd Qu.:1.00
Max. :24.020	Max. :100	Max. :16.00	Max. :1.00

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', 'RV.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)
+   )
+ )
```



```
> summary(BondsWithWarnings[,c((ncol(SomeBonds2016)+1):ncol(BondsWithWarnings))])
```

Em_FIAD_differ	EmMatMissing	CpYOverride	RV_set100percent	NegLifeFlag
Min. :0.00	Min. :0	Min. :0	Min. :0	Min. :0
1st Qu.:0.00	1st Qu.:0	1st Qu.:0	1st Qu.:0	1st Qu.:0
Median :0.00	Median :0	Median :0	Median :0	Median :0
Mean :0.04	Mean :0	Mean :0	Mean :0	Mean :0
3rd Qu.:0.00	3rd Qu.:0	3rd Qu.:0	3rd Qu.:0	3rd Qu.:0
Max. :1.00	Max. :0	Max. :0	Max. :0	Max. :0

ZeroFlag	Em_Mat_SameMY	ChronErrorFlag	FIPD_LIPD_equal	IPD_CpY_Corrupt
Min. :0	Min. :0	Min. :0.00	Min. :0.00	Min. :0.00
1st Qu.:0	1st Qu.:0	1st Qu.:0.00	1st Qu.:0.00	1st Qu.:0.00
Median :0	Median :0	Median :0.00	Median :0.00	Median :0.00
Mean :0	Mean :0	Mean :0.01	Mean :0.02	Mean :0.09
3rd Qu.:0	3rd Qu.:0	3rd Qu.:0.00	3rd Qu.:0.00	3rd Qu.:0.00
Max. :0	Max. :0	Max. :1.00	Max. :1.00	Max. :1.00

EOM_Deviation	EOMOverride	DCCOverride	NoCoups
Min. :0.00	Min. :0.00	Min. :0	Min. :0.00
1st Qu.:0.00	1st Qu.:0.00	1st Qu.:0	1st Qu.:0.00
Median :1.00	Median :1.00	Median :0	Median :0.00
Mean :0.69	Mean :0.68	Mean :0	Mean :0.01
3rd Qu.:1.00	3rd Qu.:1.00	3rd Qu.:0	3rd Qu.:0.00
Max. :1.00	Max. :1.00	Max. :0	Max. :1.00

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.<sup>6</sup>

```
> # 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
17	2016-08-23	2016-08-23	2017-08-23	2016-08-23	2017-08-23

```
>
> # manual examination of the rows where IPD_CpY_Corrupt = 1
> print(BondsWithWarnings[
+   which(BondsWithWarnings$IPD_CpY_Corrupt == 1),
+   c('ID.No', 'Issue.Date', 'FIAD.Input', 'FIPD.Input', 'LIPD.Input', 'Mat.Date',
+     'CpY.Input')], row.names = FALSE)
```

ID.No	Issue.Date	FIAD.Input	FIPD.Input	LIPD.Input	Mat.Date	CpY.Input
2	2016-06-23	2016-06-23	2016-07-15	2019-05-15	2019-06-15	4
4	2016-05-24	2016-05-24	2016-05-31	2017-04-30	2017-05-31	2
19	2016-09-28	2016-09-28	2017-02-28	2021-08-31	2021-09-28	1
56	2016-07-26	2016-07-26	2017-01-26	2020-07-26	2020-10-26	1
64	2016-04-13	2016-04-13	2016-04-24	2017-03-24	2017-04-24	6
65	2016-09-30	2016-09-30	2016-10-31	2018-02-28	2018-03-29	1
70	2016-08-26	2016-08-26	2016-11-20	2056-05-20	2056-08-31	1
82	2016-06-30	2016-06-30	2016-07-15	2028-09-15	2028-12-15	2
84	2016-07-20	2016-07-20	2016-07-24	2016-09-24	2017-01-24	2

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. If the

<sup>6</sup>Please refer to the package manual of **BondValuation** for detailed descriptions of the other warning flags.

value of FIPD. Input is correct, coupon payments should occur on July 15<sup>th</sup>, October 15<sup>th</sup>, January 15<sup>th</sup>, and April 15<sup>th</sup>. If the value of LIPD. Input is genuine, however, interest should be paid on May 15<sup>th</sup>, August 15<sup>th</sup>, November 15<sup>th</sup>, and February 15<sup>th</sup>. 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)
  RealDates   RD_indexes   CoupDates   CD_indexes   AnnivDates   AD_indexes
2016-06-23    0.08888889   2016-09-15           1   2016-06-15           0
2016-09-15    1.00000000   2016-12-15           2   2016-09-15           1
2016-12-15    2.00000000   2017-03-15           3   2016-12-15           2
2017-03-15    3.00000000   2017-06-15           4   2017-03-15           3
2017-06-15    4.00000000   2017-09-15           5   2017-06-15           4
2017-09-15    5.00000000   2017-12-15           6   2017-09-15           5
2017-12-15    6.00000000   2018-03-15           7   2017-12-15           6
2018-03-15    7.00000000   2018-06-15           8   2018-03-15           7
2018-06-15    8.00000000   2018-09-15           9   2018-06-15           8
2018-09-15    9.00000000   2018-12-15          10   2018-09-15           9
2018-12-15   10.00000000   2019-03-15          11   2018-12-15          10
2019-03-15   11.00000000   2019-06-15          12   2019-03-15          11
2019-06-15   12.00000000      <NA>         NA   2019-06-15          12
>
> # Printing data frame PaySched for bond with ID.No = 2
> print(
+   as.data.frame(do.call(rbind, lapply(FullAnalysis, `[`, 4)[2])),
+   row.names = FALSE)
  CoupDates   CoupPayments
2016-09-15    0.7368611
2016-12-15    0.8087500
2017-03-15    0.8087500
2017-06-15    0.8087500
2017-09-15    0.8087500
2017-12-15    0.8087500
2018-03-15    0.8087500
2018-06-15    0.8087500
2018-09-15    0.8087500
2018-12-15    0.8087500
2019-03-15    0.8087500
2019-06-15    0.8087500
```

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 238% 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', 'LCPTType', 'FCPLength', 'LCPLength')])
```

FCPType		LCPTType		FCPLength		LCPLength	
long	:49	long	:51	Min.	: 0.01366	Min.	: 0.02778
short	:31	regular	:28	1st Qu.:	0.92150	1st Qu.:	1.00000
regular	:20	short	:21	Median	: 1.00000	Median	: 1.25140
				Mean	: 2.19291	Mean	: 2.38417
				3rd Qu.:	3.00000	3rd Qu.:	3.00000
				Max.	:12.00000	Max.	:12.00000

### 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)
[1] 12718 16
> summary(PanelSomeBonds2016[, c(13:16)])
```

TradeDate		SETT		CP.Input		YtM.Input	
Min.	:2016-01-29	Min.	:2016-02-02	Min.	: 90.38	Min.	:-1.72500
1st Qu.:	:2016-08-03	1st Qu.:	:2016-08-05	1st Qu.:	102.73	1st Qu.:	:-0.35000
Median	:2016-10-03	Median	:2016-10-05	Median	:105.79	Median	:-0.02500
Mean	:2016-09-20	Mean	:2016-09-23	Mean	:112.53	Mean	:-0.01593
3rd Qu.:	:2016-11-17	3rd Qu.:	:2016-11-21	3rd Qu.:	113.21	3rd Qu.:	0.27500
Max.	:2016-12-30	Max.	:2017-01-03	Max.	:224.16	Max.	: 2.00000

In the following, *example 12*, the function `BondVal.Yield()` is used to determine  $\tau$ , 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.<sup>7</sup>

```
> # 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],
```

<sup>7</sup>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.

```

+           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

```