Time Series Database Interface: R PostgreSQL (TSPostgreSQL)

October 27, 2011

1 Introduction

The code from the vignette that generates this guide can be loaded into an editor with *edit(vignette("TSPostgreSQL"))*. This uses the default editor, which can be changed using *options()*. It should be possible to view the pdf version of the guide for this package with *print(vignette("TSPostgreSQL"))*.

Once R is started, the functions in this package are made available with

```
> library("TSPostgreSQL")
```

This will also load required packages *TSdbi*, *DBI*, *RPostgreSQL*, *methods*, and *tframe*. Some examples below also require *zoo*, and *tseries*.

WARNING: running these example will overwrite tables in the PostgreSQL "test" database on the server.

The PostgreSQL user, and password, should be set in PostgreSQL configuration file (.pgpass) before starting R. The Postgress documentation suggests that it should be possible to get the host from the .pgpass file too, but I have not been able to make that work. The PostgreSQL alternative to the configuration file is to use environment variables PGDATABASE, PGHOST, PGPORT, and PGUSER. This package (and CRAN) support another alternatively to set this information with environment variables POSTGRES_USER, POSTGRES_PASSWD and POSTGRES_HOST. (An environment variable POSTGRES_DATABASE can also be set, but "test" is specified below.) Below, the environment variable POSTGRES_USER is used to determine how the user and password are set. If this environment variable is empty then it is assumed the PostgreSQL mechanism will be used (i.e. the driver consults the PG* variables or the configuration file). However, the host is determined by the following logic:

```
> user <- Sys.getenv("POSTGRES_USER")
> host <- Sys.getenv("POSTGRES_HOST")
> if ("" == host) host <- Sys.getenv("PGHOST")
> if ("" == host) host <- "localhost"
> if ("" != user) {
```

```
    passwd <- Sys.getenv("POSTGRES_PASSWD")
    if ("" == passwd)
        passwd <- NULL
}
```

The next small section of code is necessary to setup database tables that are used in the examples below. It needs to be done only once for a database and might typically be done by an administrator setting up the database, rather than by an end user.

```
> m <- dbDriver("PostgreSQL")
> con <- if ("" == user) dbConnect(m, dbname = "test", host = host) else dbConnect(m,
      dbname = "test", user = user, password = passwd, host = host)
> source(system.file("TSsql/CreateTables.TSsql", package = "TSdbi"))
> dbDisconnect(con)
```

More detailed description of the instructions for building the database tables is given in the vignette for the *TSdbi* package. Those instruction show how to build the database using database utilites rather than R, which might be the way a system administrator would build the database.

2 Using the Database - TSdbi Functions

This section gives several simple examples of putting series on and reading them from the database. (If a large number of series are to be loaded into a database, one would typically do this with a batch process using the database program's utilities for loading data.) The first thing to do is to establish a connection to the database:

```
> m <- dbDriver("PostgreSQL")
> con <- if ("" == user) TSconnect(m, dbname = "test", host = host) else TSconnect(m,
      dbname = "test", user = user, password = passwd, host = host)
```

TSconnect uses *dbConnect* from the *DBI* package, but checks that the database has expected tables, and checks for additional features. (It cannot be used before the tables are created, as done in the previous section.)

This puts a series called *vec* on the database and then reads is back

```
> z <- ts(rnorm(10), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- "vec"
> if (TSexists("vec", con)) TSdelete("vec", con)
> TSput(z, con)
> z <- TSget("vec", con)
```

If the series is printed it is seen to be a "ts" time series with some extra attributes.

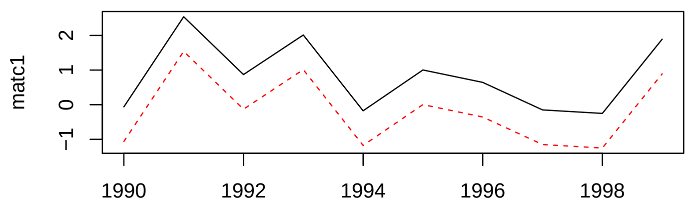
TSput fails if the series already exists on the *con*, so the above example checks and deletes the series if it already exists. *TSreplace* does not fail if the series does not yet exist, so examples below use it instead. Several plots below show original data and the data retrieved after it is written to the database. One is added to the original data so that both lines are visible.

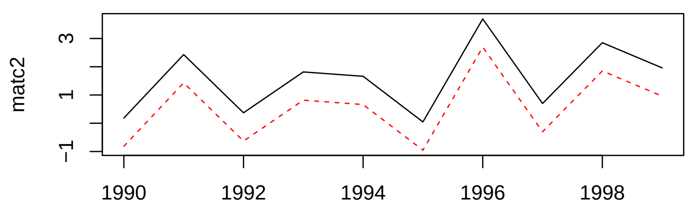
And now more examples:

```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)

[1] TRUE

> TSget("matc1", con)

Time Series:
Start = 1990
End = 1999
Frequency = 1
         1          2          3          4          5          6          7
 1.4425866 -0.7136610  0.4509371  1.8044119 -0.1759813 -0.3365572 -1.0622540
         8          9         10
 1.0986818  1.2052594  0.8081231
attr(,"seriesNames")
[1] matc1
attr(,"TSrefperiod")
[1] NA
attr(,"TSmeta")
serIDs:  matc1
 from dbname  test using  TSPostgreSQLConnection

> TSget("matc2", con)

Time Series:
Start = 1990
End = 1999
Frequency = 1
         1          2          3          4          5          6          7
 0.1655087 -1.2389493 -1.5731851  0.1184609 -0.3734899  0.7421656 -0.4723029
         8          9         10
 2.4858792 -0.8235961 -0.2102828
attr(,"seriesNames")
[1] matc2
attr(,"TSrefperiod")
[1] NA
attr(,"TSmeta")
serIDs:  matc2
 from dbname  test using  TSPostgreSQLConnection

> TSget(c("matc1", "matc2"), con)
```

```
Time Series:
Start = 1990
End = 1999
Frequency = 1
          matc1      matc2
1990  1.4425866  0.1655087
1991 -0.7136610 -1.2389493
1992  0.4509371 -1.5731851
1993  1.8044119  0.1184609
1994 -0.1759813 -0.3734899
1995 -0.3365572  0.7421656
1996 -1.0622540 -0.4723029
1997  1.0986818  2.4858792
1998  1.2052594 -0.8235961
1999  0.8081231 -0.2102828
attr(,"TSrefperiod")
[1] NA NA
attr(,"TSmeta")
serIDs:  matc1 matc2
 from dbname  test using  TSPostgreSQLConnection
```

```
> tfplot(z + 1, TSget(c("matc1", "matc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```

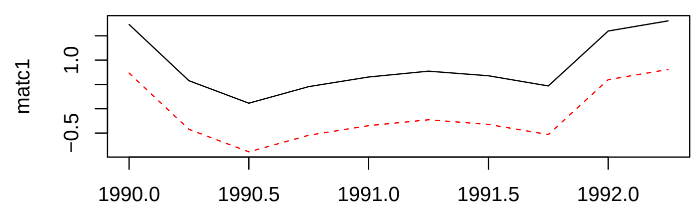


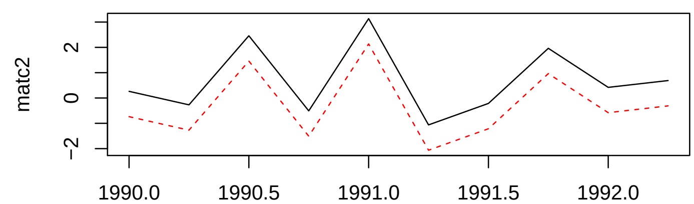



```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 4)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)

[1] TRUE

> TSget(c("matc1", "matc2"), con)

             matc1       matc2
1990 Q1 -0.9023415  0.91482344
1990 Q2  1.8469283  0.50396131
1990 Q3  0.3409156  0.66463531
1990 Q4 -0.3678032  0.56039372
1991 Q1  1.2877172  0.47863461
1991 Q2  1.2577031 -0.57044740
1991 Q3  1.5007864 -1.67569898
1991 Q4 -1.5895321  1.30205278
1992 Q1 -0.8870720 -0.72386851
1992 Q2 -0.1913366 -0.06507924
attr(,"TSrefperiod")
[1] NA NA
attr(,"TSmeta")
```

```
serIDs:  matc1 matc2
 from dbname  test using  TSPostgreSQLConnection
```

```
> tfplot(z + 1, TSget(c("matc1", "matc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```

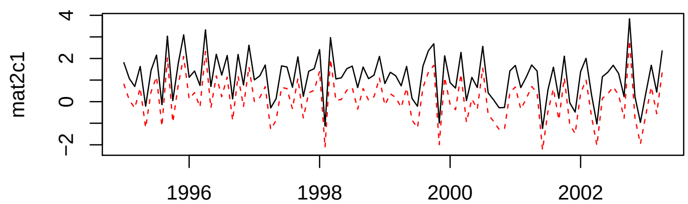


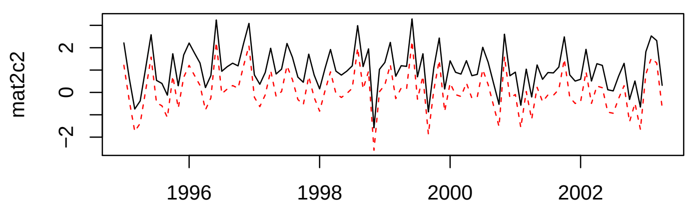



```
> z <- ts(matrix(rnorm(200), 100, 2), start = c(1995, 1), frequency = 12)
> seriesNames(z) <- c("mat2c1", "mat2c2")
> TSreplace(z, con)
```

```
[1] TRUE
```

```
> tfplot(z + 1, TSget(c("mat2c1", "mat2c2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```





The following extract information about the series from the database, although not much information has been added for these examples.

```
> TSmeta("mat2c1", con)

serIDs:  mat2c1
 from dbname  test using  TSPostgreSQLConnection

> TSmeta("vec", con)

serIDs:  vec
 from dbname  test using  TSPostgreSQLConnection

> TSdates("vec", con)

     [,1]
[1,] "vec from 1990 1 to 1999 1 A       "

> TSdescription("vec", con)

[1] NA

> TSdoc("vec", con)

[1] NA
```

Below are examples that make more use of *TSdescription* and codeTSdoc.

Often it is convenient to set the default connection:

```
> options(TSconnection = con)
```

and then the *con* specification can be omitted from the function calls unless another connection is needed. The *con* can still be specified, and some examples below do specify it, just to illustrate the alternative syntax.

```
> z <- TSget("mat2c1")
> TSmeta("mat2c1")
```

```
serIDs:  mat2c1
 from dbname  test using  TSPostgreSQLConnection
```

Data documentation can be in two forms, a description specified by *TSdescription* or longer documentation specified by *TSdoc*. These can be added to the time series object, in which case they will be written to the database when *TSput* or *TSreplace* is used to put the series on the database. Alternatively, they can be specified as arguments to *TSput* or *TSreplace*. The description or documentation will be retrieved as part of the series object with *TSget* only if this is specified with the logical arguments *TSdescription* and *TSdoc*. They can also be retrieved directly from the database with the functions *TSdescription* and *TSdoc*.

```
> z <- ts(matrix(rnorm(10), 10, 1), start = c(1990, 1), frequency = 1)
> TSreplace(z, serIDs = "Series1", con)
```

```
[1] TRUE
```

```
> zz <- TSget("Series1", con)
> TSreplace(z, serIDs = "Series1", con, TSdescription = "short rnorm series",
      TSdoc = "Series created as an example in the vignette.")
```

```
[1] TRUE
```

```
> zz <- TSget("Series1", con, TSdescription = TRUE, TSdoc = TRUE)
> start(zz)
```

```
[1] 1990    1
```

```
> end(zz)
```

```
[1] 1999    1
```

```
> TSdescription(zz)
```

```
[1] "short rnorm series"
```

```
> TSdoc(zz)
```

```
[1] "Series created as an example in the vignette."
```

```
> TSdescription("Series1", con)
```

```
[1] "short rnorm series"
```

```
> TSdoc("Series1", con)
```

```
[1] "Series created as an example in the vignette."
```

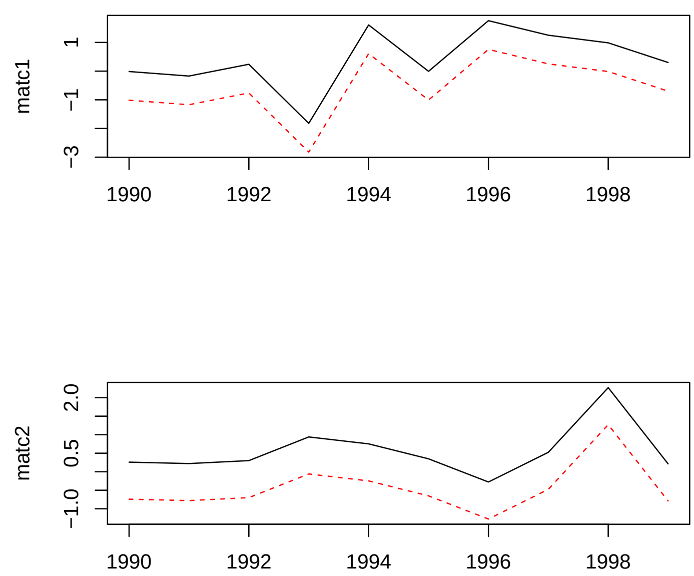
```
> z <- ts(rnorm(10), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- "vec"
> TSreplace(z, con)
```

```
[1] TRUE
```

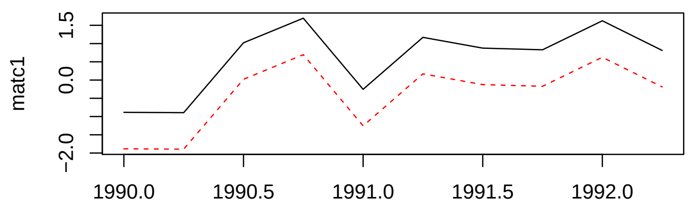
```
> zz <- TSget("vec", con)
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)
```

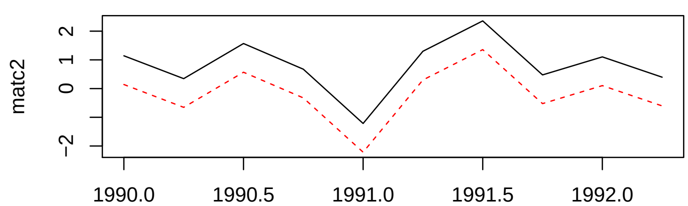
```
[1] TRUE
```

```
> tfplot(z + 1, TSget(c("matc1", "matc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```

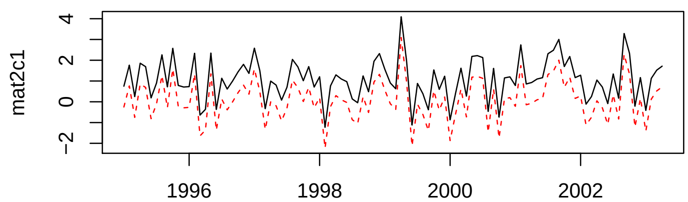


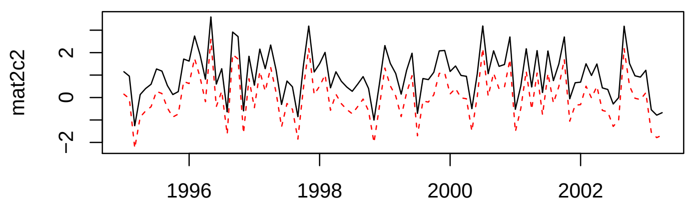
```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 4)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)

[1] TRUE

> tfplot(z + 1, TSget(c("matc1", "matc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```





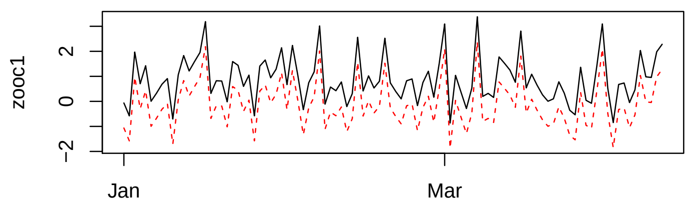

```
> z <- ts(matrix(rnorm(200), 100, 2), start = c(1995, 1), frequency = 12)
> seriesNames(z) <- c("mat2c1", "mat2c2")
> TSreplace(z, con)

[1] TRUE

> tfplot(z + 1, TSget(c("mat2c1", "mat2c2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```

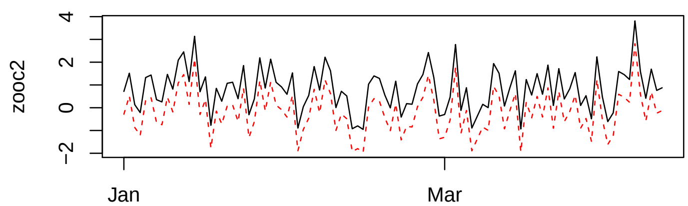




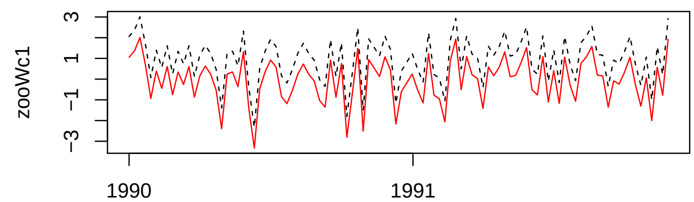
The following examples use dates and times which are not handled by *ts*, so the *zoo* time representation is used.

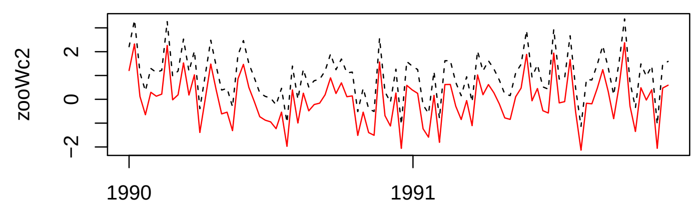
```
> require("zoo")
> z <- zoo(matrix(rnorm(200), 100, 2), as.Date("1990-01-01") +
      0:99)
> seriesNames(z) <- c("zooc1", "zooc2")
> TSreplace(z, con, Table = "D")

[1] TRUE

> tfplot(z + 1, TSget(c("zooc1", "zooc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```






```
> z <- zoo(matrix(rnorm(200), 100, 2), as.Date("1990-01-01") +
      0:99 * 7)
> seriesNames(z) <- c("zooWc1", "zooWc2")
> TSreplace(z, con, Table = "W")

[1] TRUE

> tfplot(z + 1, TSget(c("zooWc1", "zooWc2"), con), col = c("black",
      "red"), lty = c("dashed", "solid"))
```






```
> dbDisconnect(con)
```

3 Examples Using Web Data

This section illustrates fetching data from a web server and loading it into the database. This would be a very slow way to load a database, but provides examples of different kinds of time series data. The fetching is done with *TShistQuote* which provides a wrapper for *get.hist.quote* from package *tseries* to give syntax consistent with the *TSdbi*.

Fetching data may fail due to lack of an Interenet connection or delays.

First establish a connection to the database where data will be saved:

```
> con <- if ("" == user) TSconnect("PostgreSQL", dbname = "test",
      host = host) else TSconnect("PostgreSQL", dbname = "test",
      user = user, password = passwd, host = host)
```

Now connect to the web server and fetch data:

```
> require("TShistQuote")
> Yahoo <- TSconnect("histQuote", dbname = "yahoo")
> x <- TSget("^gspc", quote = "Close", con = Yahoo)
> plot(x)
```

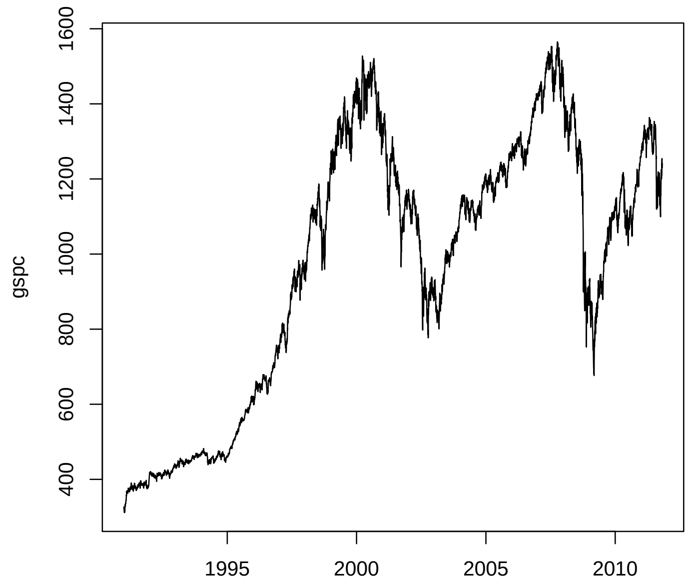
```
> tfplot(x)
> TSrefperiod(x)

[1] "Close"

> TSdescription(x)

[1] "^gspc Close  from  yahoo"

> TSdoc(x)

[1] "^gspc Close  from  yahoo retrieved  2011-10-27 20:49:29"

> TSlabel(x)

[1] "^gspc Close"

> TSsource(x)

[1] "yahoo"
```

Then write the data to the local server, specifying table B for business day data (using TSreplace in case the series is already there from running this example previously):

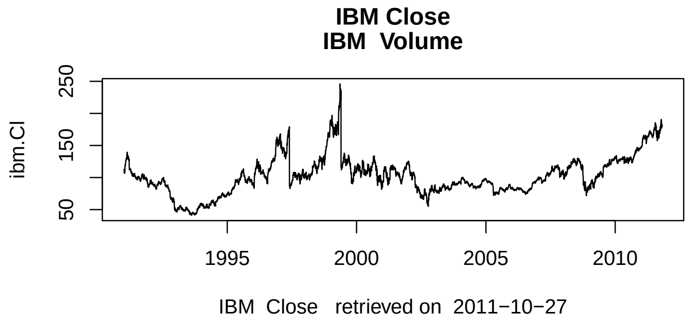
```
> TSreplace(x, serIDs = "gspc", Table = "B", con = con)

[1] TRUE
```

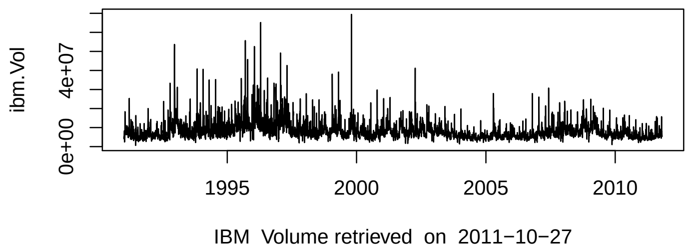
and check the saved version:

```
> TSrefperiod(TSget(serIDs = "gspc", con = con))

[1] "Close"

> TSdescription("gspc", con = con)

[1] "^gspc Close  from  yahoo"

> TSdoc("gspc", con = con)

[1] "^gspc Close  from  yahoo retrieved  2011-10-27 20:49:29"

> TSlabel("gspc", con = con)

[1] NA

> tfplot(TSget(serIDs = "gspc", con = con))
```




```
> x <- TSget("ibm", quote = c("Close", "Vol"), con = Yahoo)
> TSreplace(x, serIDs = c("ibm.Cl", "ibm.Vol"), con = con, Table = "B",
      TSdescription. = c("IBM Close", "IBM  Volume"), TSdoc. = paste(c("IBM  Close   retriev
          "IBM  Volume retrieved  on "), Sys.Date()))

[1] TRUE

> z <- TSget(serIDs = c("ibm.Cl", "ibm.Vol"), TSdescription = TRUE,
      TSdoc = TRUE, con = con)
> TSdescription(z)

[1] "IBM Close"   "IBM  Volume"

> TSdoc(z)

[1] "IBM  Close   retrieved on  2011-10-27"
[2] "IBM  Volume retrieved  on  2011-10-27"

> tfplot(z, xlab = TSdoc(z), Title = TSdescription(z))
> tfplot(z, Title = "IBM", start = "2007-01-01")
```





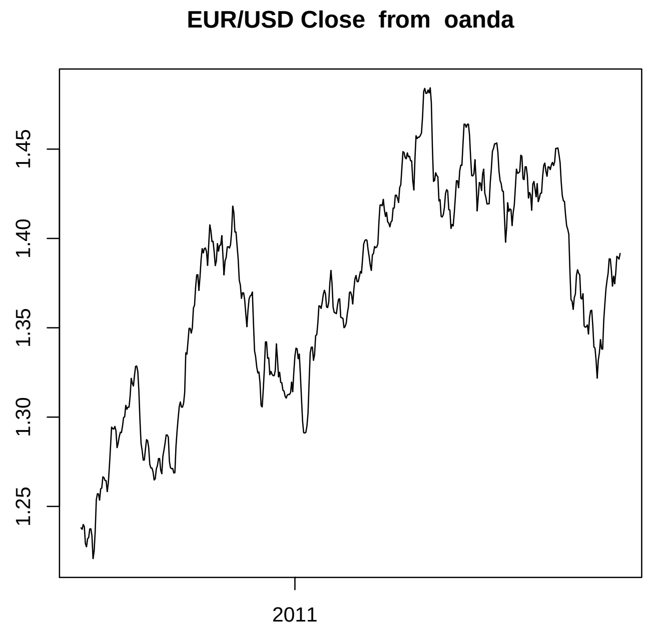
Oanda has maximum of 500 days, so the start date is specified here so as to not exceed that.

```
> Oanda <- TSconnect("histQuote", dbname = "oanda")
> x <- TSget("EUR/USD", start = Sys.Date() - 495, con = Oanda)
> TSreplace(x, serIDs = "EUR/USD", Table = "D", con = con)
```

```
[1] TRUE
```

Then check the saved version:

```
> z <- TSget(serIDs = "EUR/USD", TSlabel = TRUE, TSdescription = TRUE,
      con = con)
> tfplot(z, Title = TSdescription(z), ylab = TSlabel(z))
> tfplot(z, Title = "EUR/USD", start = "2007-01-01")
> tfplot(z, Title = "EUR/USD", start = "2007-03-01")
> tfplot(z, Title = "EUR/USD", start = Sys.Date() - 14, end = Sys.Date(),
      xlab = format(Sys.Date(), "%Y"))
```




```
> dbDisconnect(con)
> dbDisconnect(Yahoo)
> dbDisconnect(Oanda)
```

3.1 Examples Using TSdbi with ets

These examples use a database called "ets" which is available at the Bank of Canada. This set of examples illustrates how the programs might be used if a larger database is available. Typically a large database would be installed using database scripts directly rather than from R with *TSput* or *TSreplace*.

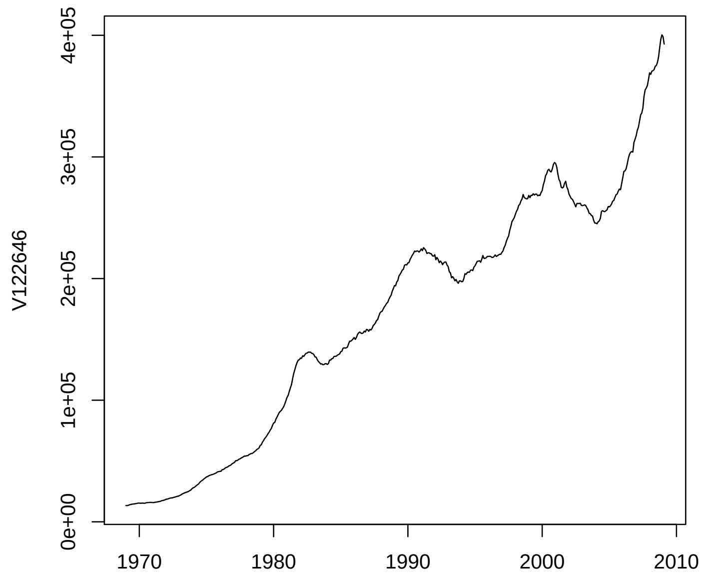
The following are wrapped in *if (!inherits(conets, "try-error"))* so that the vignette will build even when the database is not available. This seems to require an explicit call to *print()*, but that is not usually needed to display results below. Another artifact of this is that results printed in the if block do not display until the end of the block.

```
> m <- dbDriver("PostgreSQL")
> conets <- try(if ("" == user) TSconnect(m, dbname = "ets", host = host) else TSconnect(m,
      dbname = "ets", user = user, password = passwd, host = host))
> if (!inherits(conets, "try-error")) {
      options(TSconnection = conets)
      print(TSmeta("M.SDR.CCUSMA02.ST"))
```

```
    EXCH.IDs <- t(matrix(c("M.SDR.CCUSMA02.ST", "SDR/USD exchange rate",
        "M.CAN.CCUSMA02.ST", "CAN/USD exchange rate", "M.MEX.CCUSMA02.ST",
        "MEX/USD exchange rate", "M.JPN.CCUSMA02.ST", "JPN/USD exchange rate",
        "M.EMU.CCUSMA02.ST", "Euro/USD exchange rate", "M.OTO.CCUSMA02.ST",
        "OECD /USD exchange rate", "M.G7M.CCUSMA02.ST", "G7   /USD exchange rate",
        "M.E15.CCUSMA02.ST", "Euro 15. /USD exchange rate"),
        2, 8))
    print(TSdates(EXCH.IDs[, 1]))
    z <- TSdates(EXCH.IDs[, 1])
    print(start(z))
    print(end(z))
    tfplot(TSget(serIDs = "V122646", conets))
}
```

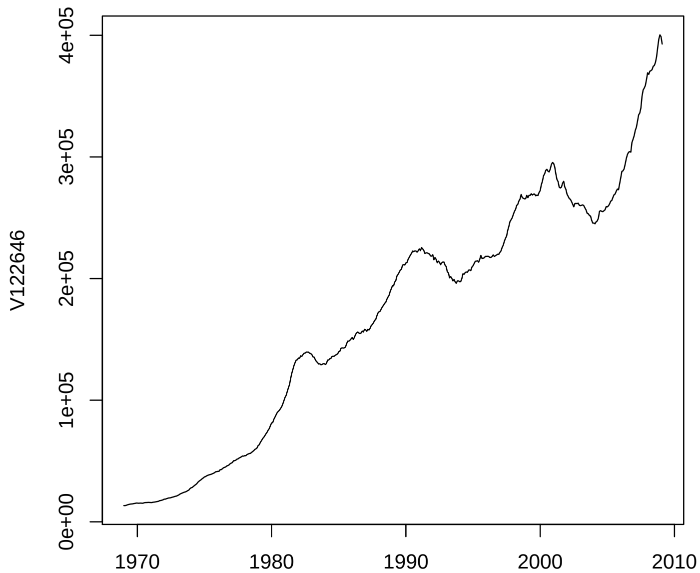
```
serIDs:  M.SDR.CCUSMA02.ST
 from dbname  ets using  TSPostgreSQLConnection
description:  Special Drawing Right---Currency Conversions/US$ exchange rate/Average of dail
documentaion:  Special Drawing Right---Currency Conversions/US$ exchange rate/Average of dai
     [,1]
[1,] "M.SDR.CCUSMA02.ST from 1960 1 to 2009 2 M   NA   "
[2,] "M.CAN.CCUSMA02.ST from 1960 1 to 2009 2 M   NA   "
[3,] "M.MEX.CCUSMA02.ST from 1963 1 to 2009 2 M   NA   "
[4,] "M.JPN.CCUSMA02.ST from 1960 1 to 2009 2 M   NA   "
[5,] "M.EMU.CCUSMA02.ST from 1979 1 to 2009 2 M   NA   "
[6,] "M.OTO.CCUSMA02.ST  not available"
[7,] "M.G7M.CCUSMA02.ST  not available"
[8,] "M.E15.CCUSMA02.ST  not available"
[[1]]
[1] 1960    1

[[2]]
[1] 1960    1

[[3]]
[1] 1963    1

[[4]]
[1] 1960    1

[[5]]
[1] 1979    1

[[6]]
[1] NA

[[7]]
```

```
[1] NA

[[8]]
[1] NA

[[1]]
[1] 2009    2

[[2]]
[1] 2009    2

[[3]]
[1] 2009    2

[[4]]
[1] 2009    2

[[5]]
[1] 2009    2

[[6]]
[1] NA

[[7]]
[1] NA

[[8]]
[1] NA
```

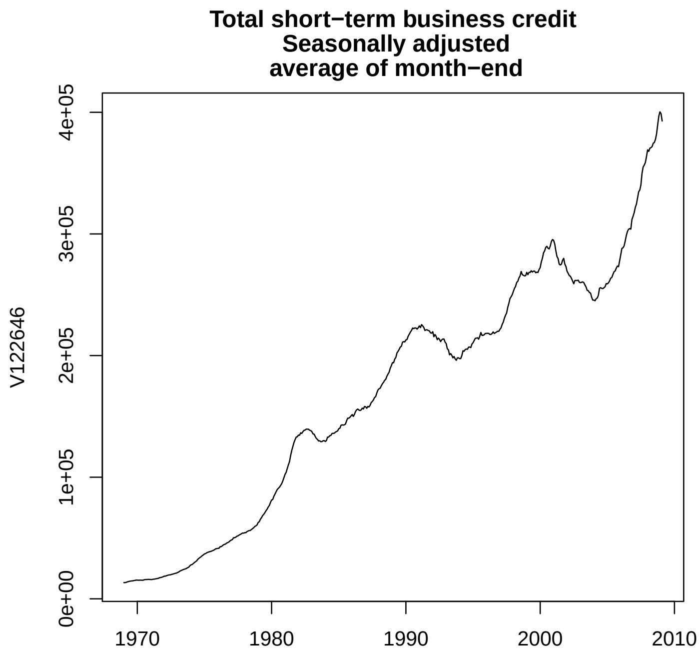



```
> if (!inherits(conets, "try-error")) {
      print(TSdescription(TSget("V122646", TSdescription = TRUE)))
      print(TSdescription("V122646"))
      print(TSdoc(TSget("V122646", TSdoc = TRUE)))
      print(TSdoc("V122646"))
      tfplot(TSget("V122646", names = "V122646", conets))
  }
```

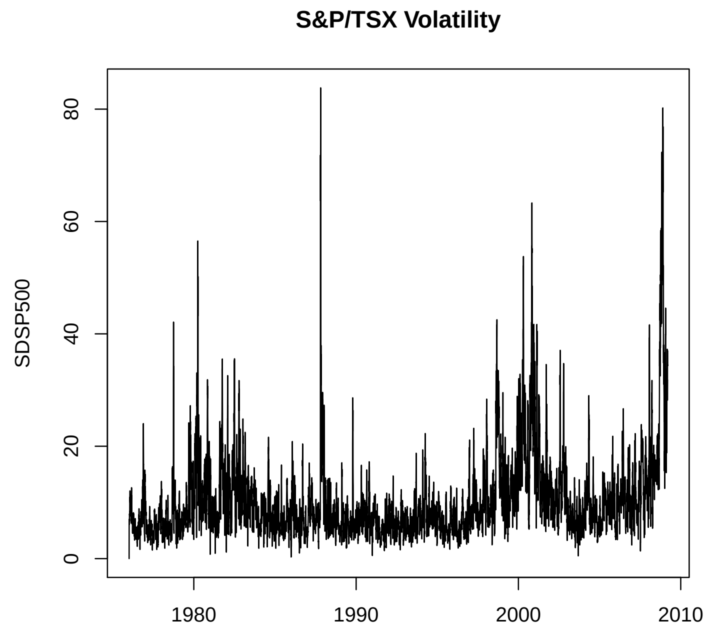
```
[1] "Total short-term business credit, Seasonally adjusted, average of month-end"
[1] "Total short-term business credit, Seasonally adjusted, average of month-end"
[1] "Same as B171"
[1] "Same as B171"
```



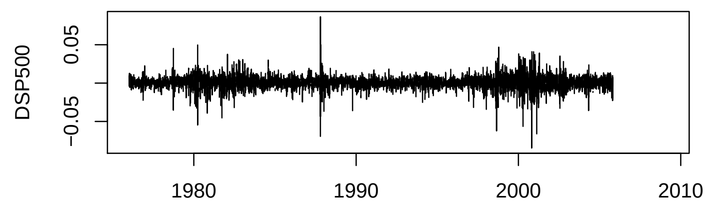

```
> if (!inherits(conets, "try-error")) {
      z <- TSget("V122646", TSdescription = TRUE)
      tfplot(z, Title = strsplit(TSdescription(z), ","))
  }
```

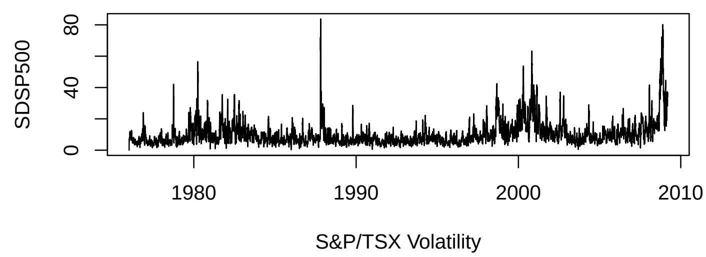



```
> if (!inherits(conets, "try-error")) {
      z <- TSget("SDSP500", TSdescription = TRUE)
      tfplot(z, Title = TSdescription(z))
      plot(z)
  }
```

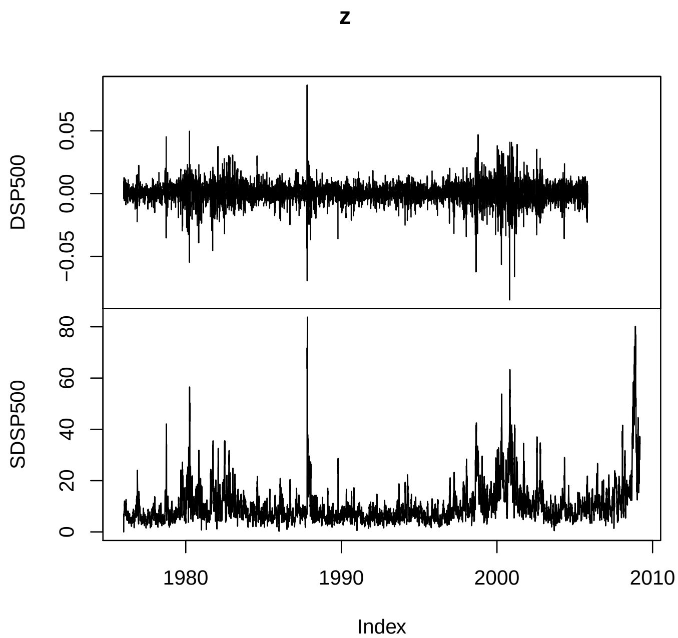



```
> if (!inherits(conets, "try-error")) {
      z <- TSget(c("DSP500", "SDSP500"), TSdescription = TRUE)
      tfplot(z, xlab = TSdescription(z))
  }
```

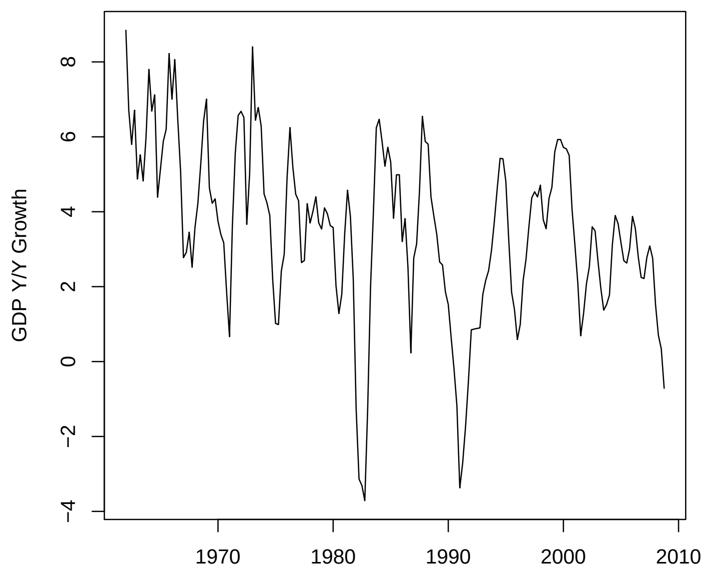





```
> if (!inherits(conets, "try-error")) {
      plot(z)
  }
```




```
> if (!inherits(conets, "try-error")) {
      ETSgdp <- annualizedGrowth(aggregate(TSget("V1992067"), nfrequency = 4,
          FUN = mean), lag = 4, names = "GDP Y/Y Growth")
      tfplot(ETSgdp)
  }
```




```
> if (!inherits(conets, "try-error")) {
      dbDisconnect(options()$TSconnection)
      options(TSconnection = NULL)
  }
```

4 Examples Using DBI and direct SQL Queries

The following examples are queries using the underlying "DBI" functions. They should not often be needed to access time series, but may be useful to get at more detailed information, or formulate special queries.

```
> m <- dbDriver("PostgreSQL")
> con <- if ("" == user) TSconnect(m, dbname = "test", host = host) else TSconnect(m,
      dbname = "test", user = user, password = passwd, host = host)
> options(TSconnection = con)
```

```
> dbListTables(con)
```

```
 [1] "meta" "a"    "b"    "d"    "m"    "u"    "q"    "s"    "w"    "i"
[11] "t"
```

If schema queries are supported then table information can be found in a generic SQL way, but on some systems this will fail because users do not have read priveleges on the INFORMATION_SCHEMA table, so the following are wrapped in *try()*. (SQLite does not seem to support this at all.)

```
> try(dbGetQuery(con, paste("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.Columns ",
      " WHERE TABLE_SCHEMA='test' AND table_name='A' ;")))

data frame with 0 columns and 0 rows

> try(dbGetQuery(con, paste("SELECT COLUMN_NAME, COLUMN_DEFAULT, COLLATION_NAME, DATA_TYPE,'
      "CHARACTER_SET_NAME, CHARACTER_MAXIMUM_LENGTH, NUMERIC_PRECISION",
      "FROM INFORMATION_SCHEMA.Columns WHERE TABLE_SCHEMA='test' AND table_name='A' ;")))

data frame with 0 columns and 0 rows

> try(dbGetQuery(con, paste("SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, NUMERI
      "FROM INFORMATION_SCHEMA.Columns WHERE TABLE_SCHEMA='test' AND table_name='M';")))

data frame with 0 columns and 0 rows
```

Finally, to disconnect gracefully, one should

```
> dbDisconnect(con)
> dbDisconnect(options()$TSconnection)
> options(TSconnection = NULL)
```