TDMS File Reader and Analyzer

Abhishek Hanchate

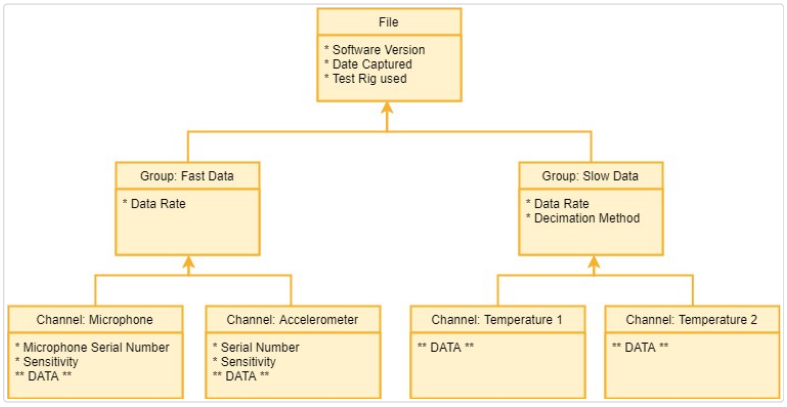
2021-12-13

Introduction and Motivation

In this vignette, we go over the readTDMS package, an R-based TDMS file format reader and analyzer, which is used for reading an TDMS file and performing subsequent analyses on the extracted data streams from the file. Not a lot of attention or consideration is given to data storage options at forefront of application planning and data storage. Decisions are often made arbitrarily on an as-needed, per-application basis without much thought to reusability, scalability, and recyclability or reproducibility, which leads to tedious and costly software rearchitecture. Given that applications and requirements change over time, even the most popular file storing formats quickly fall short of meeting the demands of engineers and scientists storing time-based measurement data. This is especially common in fields such as smart manufacturing, IoT, and healthcare, wherein large amounts of data is generated without much metadata associated with it.

NI TDMS Format

An NI TDMS (National Instruments Technical Data Management Streaming) file format is used for saving well-documented measurement data which is stored in the most efficient, organized, and scalable fashion to disk. For example, a typical TDMS file can store the same amount of data in about one-fourth disk space as compared to that required by an Microsoft Excel file. Given its structured and binary format nature, each type of measurement is stored in its own channel. Every channel belongs to a certain channel group and a given TDMS file can contain several such groups of multiple channels. At each of these levels, I can assign properties which keep a track of the associated metadata. This complex yet useful structure boosts traceability and lets us search for relevant data and track any problems that pop up. Given the binary format storage, TDMS files tend to be fast to read with no loss in precision for a data stream. The figure below gives us an example of a TDMS structure.



Given the fairly complex binary format, these files are often difficult to directly read in R. This package lets us test the use of R for data processing requirements without changing anything with the data acquisition process. Also, having this direct reading process assists in reducing barriers to others trying to use R for such a purpose. The package will have a dependency on tdmsreader by msuefishlab and contains further analyses and functionalities such as frequency domain visuals as well as time-frequency domain spectral plots.

Source Code

https://github.com/abhishekhanchate/readTDMS

Installation Guide and Troubleshooting

readTDMS is designed as an R package and we can install it from GitHub via:

```
devtools::install_github("abhishekhanchate/readTDMS")
```

In case of errors, please make sure that you have devtools installed as well. If not, you can install it and then install readTDMS using:

```
install.packages('devtools')
devtools::install_github("abhishekhanchate/readTDMS")
```

Potential Windows Installation Errors

Sometimes, the install_github function doesn't work properly on Windows as it fails to install some of the dependencies. So, you may have to install them manually. Possible dependencies include the following:

```
install.packages(c('tdmsreader', 'oce', 'signal', 'ggplot2'))
```

Note that this is only a problem on windows, other platforms should install all dependencies automatically when you use:
`devtools::install_github('abhishekhanchate/readTDMS')`
or
`install_github('abhishekhanchate/readTDMS')`

Using readTDMS

Once the package has been installed, you can run it by running:

```
library(readTDMS)
#> Loading required package: tdmsreader
```

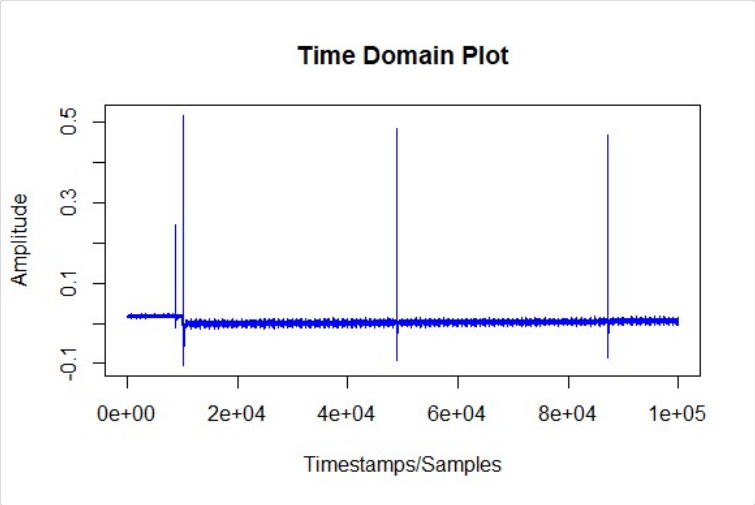
Example Use-case

We demonstrate the functionalities of the package readTDMS in reading and analyzing data streams coming from a TDMS file. The first step after installing and importing the library is to read a TDMS file from a given repository/directory. (Note: The TDMS_Index file associated with the TDMS file must also be present in the same folder)

```
f <- file('./data/file.tdms', 'rb')
```

Now, by using the tdmsread() function, we can read in the required TDMS file. This function provides us with the extracted data stream in form of the amplitude values of the data as well as the timestamps associated with it. We also get a brief summary of the data stream along with an optional (by default, False) visual into the time domain illustration.

```
out <- tdmsread(f, plot = TRUE)
```

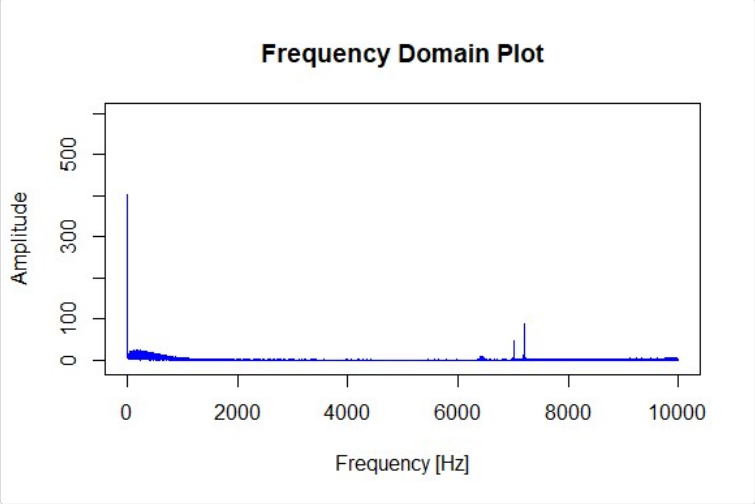



```
head(out$datastream, n = 50)
#>  [1] 0.01800260 0.01964717 0.01668694 0.01833151 0.01701586 0.01767369
#>  [7] 0.01800260 0.01602911 0.01833151 0.01800260 0.01635803 0.01898934
#> [13] 0.01931826 0.01635803 0.01701586 0.01866043 0.01701586 0.01964717
#> [19] 0.01767369 0.01635803 0.01800260 0.01701586 0.01635803 0.01931826
#> [25] 0.01800260 0.01701586 0.01931826 0.01833151 0.01833151 0.01866043
#> [31] 0.01668694 0.01767369 0.01800260 0.01866043 0.01898934 0.01701586
#> [37] 0.01800260 0.01964717 0.01537129 0.01898934 0.01833151 0.01668694
#> [43] 0.01898934 0.01767369 0.01833151 0.01931826 0.01767369 0.01800260
#> [49] 0.01767369 0.01833151
head(out$timestamps, n = 50)
#>  [1] 0.00001 0.00002 0.00003 0.00004 0.00005 0.00006 0.00007 0.00008 0.00009
#> [10] 0.00010 0.00011 0.00012 0.00013 0.00014 0.00015 0.00016 0.00017 0.00018
#> [19] 0.00019 0.00020 0.00021 0.00022 0.00023 0.00024 0.00025 0.00026 0.00027
#> [28] 0.00028 0.00029 0.00030 0.00031 0.00032 0.00033 0.00034 0.00035 0.00036
#> [37] 0.00037 0.00038 0.00039 0.00040 0.00041 0.00042 0.00043 0.00044 0.00045
#> [46] 0.00046 0.00047 0.00048 0.00049 0.00050
out$summ
#>       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
#> -0.1046824  0.0002412  0.0032015  0.0040188  0.0058328  0.5172946
close(f)
```

The Time domain visual helps us look into the original data stream measurements. We can visually analyze various spikes associated with the data stream.

After this, we can look into the Frequency domain features. The readTDMS package has two functions, namely datafreq() and fftprofile(), for this purpose. With the former, we can visually look at the frequency domain plot of the data stream along with its frequency components and associated frequencies. While with the latter, we can get more into the details of frequency components and extract the desired number of top peaks associated with the frequency domain of the data stream.

```
f <- file('./data/file.tdms', 'rb')
freq_out <- datafreq(f, frequencyPoints = 10000, xlim1 = 0, xlim2 = 10000, ylim1 = -10, ylim2 = 600)
```




```
head(freq_out$freq_comps, n = 50)
#>  [1] 401.884651 203.648187 161.584601 129.173084 115.990626  95.983773
#>  [7]  76.327874  57.803480  38.616069  21.498538  15.376536  16.799750
#> [13]  25.293358  31.345824  34.140336  32.650052  29.338313  23.770582
#> [19]  18.644607   9.761841   4.107758  11.624497  14.277370  18.269960
#> [25]  19.219514  20.035269  19.430813  15.662691  12.012762   5.941508
#> [31]   4.802850   8.279639   9.528799  11.687038  14.208923  14.051744
#> [37]  15.069140   9.197068   8.151103   5.938243   4.099005   5.388219
#> [43]   8.453789  10.026828  13.822040   9.588169   9.994985   7.416769
#> [49]   6.501648   7.150824
head(freq_out$frequency, n = 50)
#>  [1] 0.000000 0.200004 0.400008 0.600012 0.800016 1.000020 1.200024 1.400028
#>  [9] 1.600032 1.800036 2.000040 2.200044 2.400048 2.600052 2.800056 3.000060
#> [17] 3.200064 3.400068 3.600072 3.800076 4.000080 4.200084 4.400088 4.600092
#> [25] 4.800096 5.000100 5.200104 5.400108 5.600112 5.800116 6.000120 6.200124
#> [33] 6.400128 6.600132 6.800136 7.000140 7.200144 7.400148 7.600152 7.800156
#> [41] 8.000160 8.200164 8.400168 8.600172 8.800176 9.000180 9.200184 9.400188
#> [49] 9.600192 9.800196
close(f)
```

The top frequency components can then be extracted as below:

```
f <- file('./data/file.tdms', 'rb')
top_freq <- fftprofile(f, frequencyPoints = 10000, n = 25) # Top 25 Frequencies
top_freq
#>  [1]    0.000000    0.200004    0.400008    0.600012    0.800016    1.000020
#>  [7] 7212.344247    1.200024    1.400028 7011.940239    1.600032    2.800056
#> [13] 7011.740235    3.000060    2.600052    3.200064  174.803496    2.400048
#> [19]  172.203444  185.203704  161.803236  240.004800  226.004520    3.400068
#> [25]  227.004540
close(f)
```

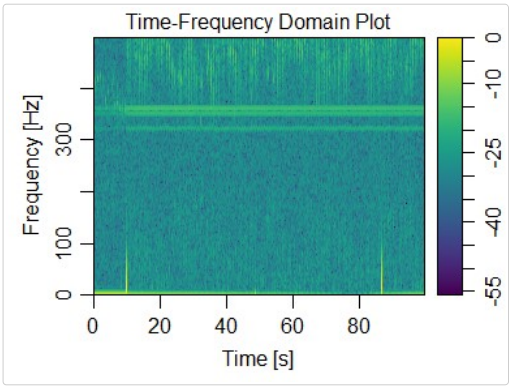
Similarly, we can get into the Time-frequency domain features associated with the data stream. However, it is usually advisable to first demean your data stream to get rid of any unwanted trend associated with it. We do so as illustrated by example below:

```
f <- file('./data/file.tdms', 'rb')
out <- tdmsread(f, plot = FALSE)
data <- out$datastream
data_demeaned <- demeaner(data)
mean(data)
#> [1] 0.004018806
mean(data_demeaned)
#> [1] 3.221332e-19
close(f)
```

It is clear from the above mean values that we have accomplished what we wanted to do. Now, we can work on extracting the desired Time-frequency domain features.

```
f <- file('./data/file.tdms', 'rb')
```

```r
out <- tdmsread(f, plot = FALSE)
data <- out$datastream
time_freq_out <- spectro(x = data, n = 1024, Fs = 1000, window = 256, overlap = 128)
```




```r
time_freq_out$freq_comps[1:5, 1:5]
#>           [,1]      [,2]      [,3]      [,4]      [,5]
#> [1,] -5.647600 -5.709226 -5.703281 -5.677897 -5.718273
#> [2,] -5.821739 -5.884484 -5.878615 -5.851735 -5.891992
#> [3,] -6.352857 -6.419167 -6.413388 -6.381750 -6.421543
#> [4,] -7.269659 -7.342718 -7.336501 -7.295909 -7.334492
#> [5,] -8.630719 -8.715513 -8.706776 -8.650918 -8.686523
head(time_freq_out$timestamps, n = 5)
#> [1] 0.001 0.129 0.257 0.385 0.513
head(time_freq_out$frequencystamps, n = 5)
#> [1] 0.0000000 0.9765625 1.9531250 2.9296875 3.9062500
close(f)
```

Conclusion

The readTDMS package offers a streamlined approach to reading and analyzing any given TDMS file. You have a direct read functionality in R along with access to several spectral analyses techniques such as FFT and Spectrograms. You can also create any desired plots with a variety of changeable parameters and get summary statistics.

Notes

This package only supports a subset of the TDMS spec and has been tested on single channel 32-bit float data stream.

Feedback is welcomed!

Credit

This package relies on the initial direct reading functionality on the R package, tdmsreader, which in-turn is a port of the Python package, npTDMS, into R and shares the same LGPL license.