FFTWTOOLS Timing and Simple Use (Version 0.9-5)

Karim J. Rahim

June 26, 2013

1 Overview

The package `fftwtool` provides a wrapper for `FFTW` code discussed in [1]. This vignette is an example comparing execution times of `fftw` from the package `fftwtools` to the standard `R fft` function. The vignette also shows how to replace the `R` function `fft` with `fftw`. The functions `fftw` and `mvfftw` mimic the behaviour of the `R` functions `fft` and `mvfft`. I note that there is a `R` package called `FFTW` which offers different functionality than this package, specifically it allows the user to set *plans*, and improve the speed with multiple calls to `fftw` when using data sets of the same size, see [1].

2 Timing Example

We begin with a quick demonstration of the speed difference between a simple call to `fftw` and the default `fft` function. The performance improvement is visible with large data sets. Using the example below with 2^{20}, over one million, data points I observed the following execution times:

- `R`'s `fft`: 19.069 seconds
- `fftw` default call: 8.804 seconds
- `fftw` with `HermConj` set to `FALSE`: 7.594 seconds.

You can test timing at any size and decide which function to use. As the speed advantage from the use of the package `fftw` is only seen in large examples, I currently use the `R fft` functions, and I change to `fftw` only when I encounter significantly large data sets and speed becomes a concern.

To reduce the check and install times of this package, I reduced `fftLength` from 2^{20} to 2^8 in the code shown in this vignette. The standard `R` routine is faster using this number of samples, but it is *not* faster with very large data sets.

To compare times, first we look at the time required for he default `R fft` routine.

```
> library("fftwtools")
> ## we try power of 2 but we can try other values
> ## we do ffts of 2^20 points
>
> ## reduced to 2^8 for package distribution.
> fftLength <- 2^8
> set.seed(10)
> g <- rnorm(fftLength)
> ##timing # Start the clock!
> ptm <- proc.time()
> # Loop through
> for (i in 1:100){
```

```
+     fft(g)
+ }
> # Stop the clock
> proc.time() - ptm

   user  system elapsed
  0.000   0.000   0.001

>
```

Next we look at replacing `fft` with `fftw` without any other changes.

```
> ##timing # Start the clock!
> ptm <- proc.time()
> # Loop through
> for (i in 1:100){
+     fftw(g)
+ }
> # Stop the clock
> proc.time() - ptm

   user  system elapsed
  0.008   0.000   0.008

>
```

Finally we look to see how much additional improvement can by had by not returning the complex conjugate which is not required for real data. I suspect that this speed up is partially due to decreased memory allocation.

```
> ##timing # Start the clock!
> ptm <- proc.time()
> # Loop through
> for (i in 1:100){
+     fftw(g, HermConj=FALSE)
+ }
> # Stop the clock
> proc.time() - ptm

   user  system elapsed
  0.004   0.000   0.005

>
```

3 Replace R's `fft` call with `fftw`

I do not recommend you do the following in general. However, it is a simple way to speed up code or code in packages that call `fft`. This is done by replacing all calls to `fft` with calls to `fftw`.

```
> ## basic option ot overwrite calls
> fft <- function(z, inverse = FALSE) {
+     fftwtools::fftw(z, inverse=inverse)
+ }
> mvfft <- function(z, inverse=FALSE) {
+     fftwtools::mvfftw(z, invese=inverse)
+ }
```

The above is a simple method of replacing all `fft` calls with `fftw` calls in the `multitaper` package which I maintain. If you are interested in the additional improvement available from not returning the unnecessary complex conjugate when using real data, you can overwrite the call setting `HermConj` to `FALSE`.

```
> fft <- function(z, inverse = FALSE) {
+     fftwtools::fftw(z, inverse=inverse, HermConj=FALSE)
+ }
```

The last method is only valid when the input is not complex, and it may break certain calls depending on when the complex conjugate is discarded. I note that if you discard the complex conjugate, you will need the length of the original data to perform an inverse `fft`. If you are using the latter method then it may be wise to look into further functionality provided in the `R` packages `FFTW` and `fftwtools`.

3.1 Clean up

If you replace the `R`'s call to `fft` with `fftw` then it is good practice to clean up the replacement and restore calls to `fft` and `mvfft` to the standard `R` routine when you are finished using `fftw`. The following code shows how this cleanup is performed.

```
> rm(fft, mvfft)
```

References

[1] Matteo Frigo and Steven˜G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".