FFTWTOOLS Timing and Simple Use (Version 0.9-11)

Karim J. Rahim

November 14, 2024

1 Overview

The package `fftwtool` provides a wrapper for `FFTW` code discussed in [1]. This vignette provides example code for comparing execution times of `fftw` from the package `fftwtools` to the standard `R` `fft` function, and it demonstrates how to replace the `R` function `fft` with `fftw`. The functions `fftw` and `mvfftw` mimic the behaviour of the `R` functions `fft` and `mvfft`. There is an `R` package called `FFTW` which offers different functionality than fftwtools, 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 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 the default `R` `fft` routine.

```
> library("fftwtools")
> ## We increment by powers of 2, but one can use other incrementes
> ## We choose fftlength = 2^20, but
> ## **this was reduced to 2^8 for the vignette distribution.**
> fftLength <- 2^8
> set.seed(10)
> g <- rnorm(fftLength)
> ##Start the clock
> ptm <- proc.time()
> ## Loop through
> for (i in 1:100){
+     fft(g)
```

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

   user  system elapsed
  0.001   0.001   0.002

>
```

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
  0.006   0.000   0.007

>
```

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. It is likely this speed up is partially due to decreased memory allocation.

```
> ## 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
  0.002   0.002   0.004

>
```

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

The following is a quick and dirty way to replace `fftw` with `fftw` in existing code. It is more appropriate to use a conditional statement in the replacement functions to call `fftw` and `mvfftw` when the length of z is appropriately large.

```
> ## 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. If you discard the complex conjugate, you will need the length of the original data to perform an inverse Fourier transform. 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".