Discrete Fourier Transform (slow) in R

Cody Frisby

12/2/2017

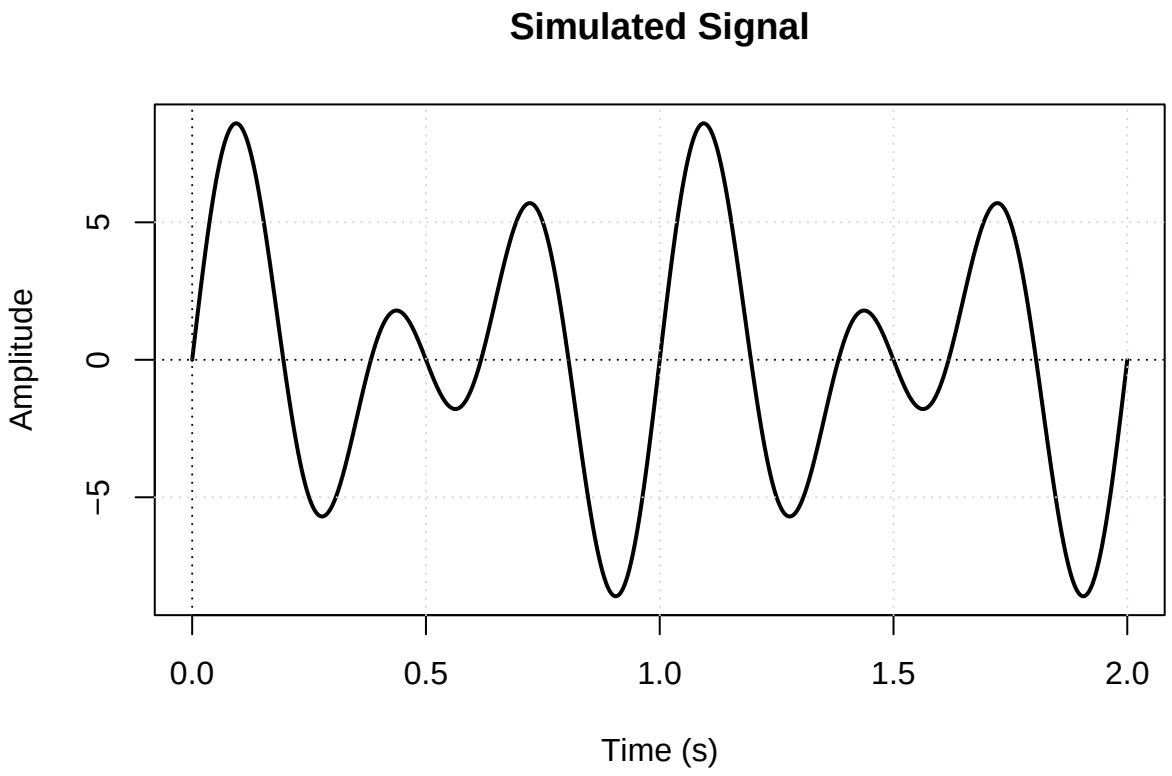
Below is some code to simulate a signal. I also display the signal.

```
f1 <- 2
f2 <- 3
a1 <- 4
a2 <- 5
```

Note:

$$a_1 = 4,\ a_2 = 5,\ f_1 = 2,\ f_2 = 3$$

```
srate <- 1000 # sampling rate
# Matlab -> time = 0:1/srate:2-1/srate;
time <- seq(0, 2, 1/srate)
n <- length(time)

## generate a signal
signal <- a1 * sin(2 * pi * f1 * time) + a2 * sin(2 * pi * f2 * time)

### Here's the plot of our signal
plot(time, signal, type = "l", ylab = "Amplitude", lwd = 2,
     main = "Simulated Signal", xlab = "Time (s)")
grid()
abline(h = 0, lty =3); abline(v = 0, lty = 3)
```

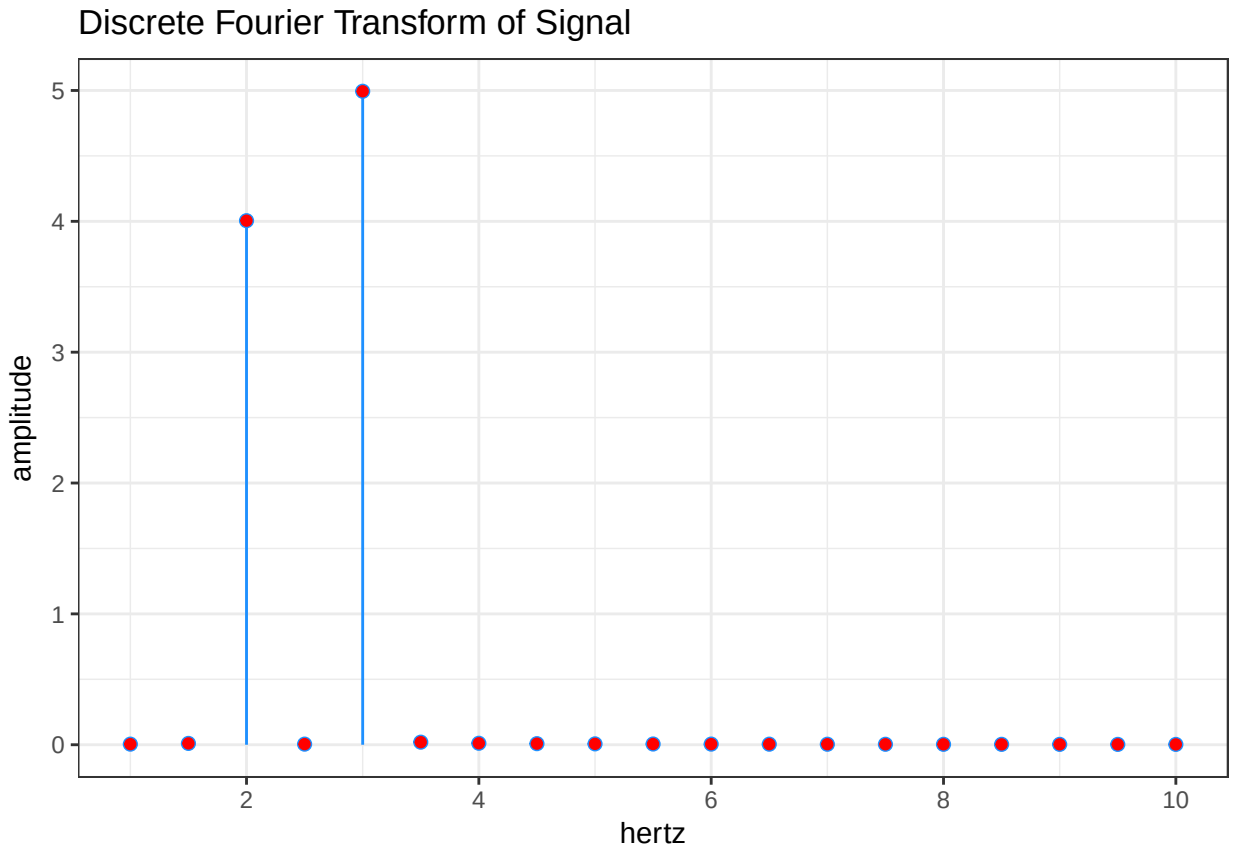


Here is a for loop to compute the (slow) discrete fourier transform.

```
i <- 0+1i # define i, imaginary number

# initialize two vectors to use in loop
ftime <- seq(0, (n-1)/n, length.out = n)
fcoef <- rep(0, length(signal))

### start loop
for (fi in 1:n) {
  ## complex sine wave
  csw <- exp(-i * 2 * pi * (fi - 1) * ftime)
  # dot product between sine wave and signal
  fcoef[fi] <- sum(signal * csw) / n
} ## end fourier for loop

# compute amplitudes
ampl <- 2 * abs(fcoef)

# compute frequencies
hz <- seq(0, srate/2, length.out = n/2)
```

Use `ggplot2` package to plot the solution to my (slot) DFT and compare to `R`'s `fft` funciton.

```
# get data ready frist
df <- data.frame(amplitude = ampl[1:length(hz)], hertz = hz)

## R FFT function for comparison
Rfft <- data.frame(FFT = 2 * abs(fft(signal) / n)[1:length(hz)])

# now use ggplot
library(ggplot2)
library(scales)
gg <- ggplot(data = df, aes(x = hertz, y = amplitude))
gg <- gg + geom_segment(aes(xend = hertz, yend = amplitude - amplitude),
                  color = "dodgerblue") +
  geom_point(size = 2, color = "dodgerblue") +
  xlim(0, 10) + ylab("amplitude") + theme_bw() +
  ggtitle("Discrete Fourier Transform of Signal")
print(gg + geom_point(data = Rfft,
                mapping = aes(x = df$hertz, y = FFT),
                color = "red") +
       scale_x_continuous(limits = c(1, 10) , breaks = pretty_breaks()))
```



As can be seen, where blue is my calculation of the signal and red is the `fft` function in R, we are in complete agreement and we have recovered the original amplitudes and frequencies.

Again, the above code is the SLOW DFT. What about the FFT? How do I implement this in R?