Manual for the freq_map() function

Chitran Ghosal

May 3, 2023

1 Abstract

This manual shows how to use the *freq_map()* package for mapping intensity dependence of a certain spectral window(in fourier space) in real space. This frequency mapping is similar to how dark-field microscopy is carried out in Low Energy Electron Microscopy(LEEM).

2 Introduction

Often, within the parlance of data, the obtained data contains a range of periodicities. The obvious solution is to look at a frequency domain representation(Fourier Transform) of the signals in question. However, this makes sense when the frequency composition of the signal is uniform throughout the sample. In reality this is not the case.

In the event of much more real signals, example audio signals or sunlight, the fourier space also isn't comprised of individual signal frequencies, rather they form a continuum in frequency space. In this case it makes sense to be interseted in the composition of the signal in real space(with respect to a given spectral-frequency window). The function *freq_map()* aims to solve this problem of frequency (window) mapping.

3 Examples

In the examples that follow, the use-cases and the proof of concept of the *freq_map()* package are shown

3.1 Example 1

The code in listing.1 is used to build a noisy mix of three sine waves with $\omega = 2\pi f$. With f being chosen from 1000Hz, 5000Hz and 7734Hz.

```
rm(list=ls())
library('StatsChitran') #Call library
ClearPlot() #Clear the plot
t <- seq(0,10, by=1/60000) #define the time axis
fcomp <- c(1000, 5000, 7734) #set which frequencies to build
Y <- 3*sin(2*pi*fcomp[1]*t)*gauss(t, amp =5, sig = 0.7, mu=1 ) +
    5*sin(2*pi*fcomp[2]*t)*gauss(t, amp = 4, sig = 1, mu=3) +
    4*sin(2*pi*fcomp[3]*t)*gauss(t, amp = 4.3, sig = 3, mu=7) #define signal
```

```
Y <- Y + runif(length(Y), min = 0.25*min(Y), max = 0.25*max(Y)) #add noise
plot(t,Y, col=rgb(0,0,1,0.25), typ='l')
fY <- ft(t,Y,w = T) #function available with StatsChitran
plot(fY$wf, abs(fY$fy), col= 'grey', typ='l', xlab='w', ylab='Amp')
```

Listing 1: code for simulating three frequency signal in fig.1a and its fourier transform in fig.1b

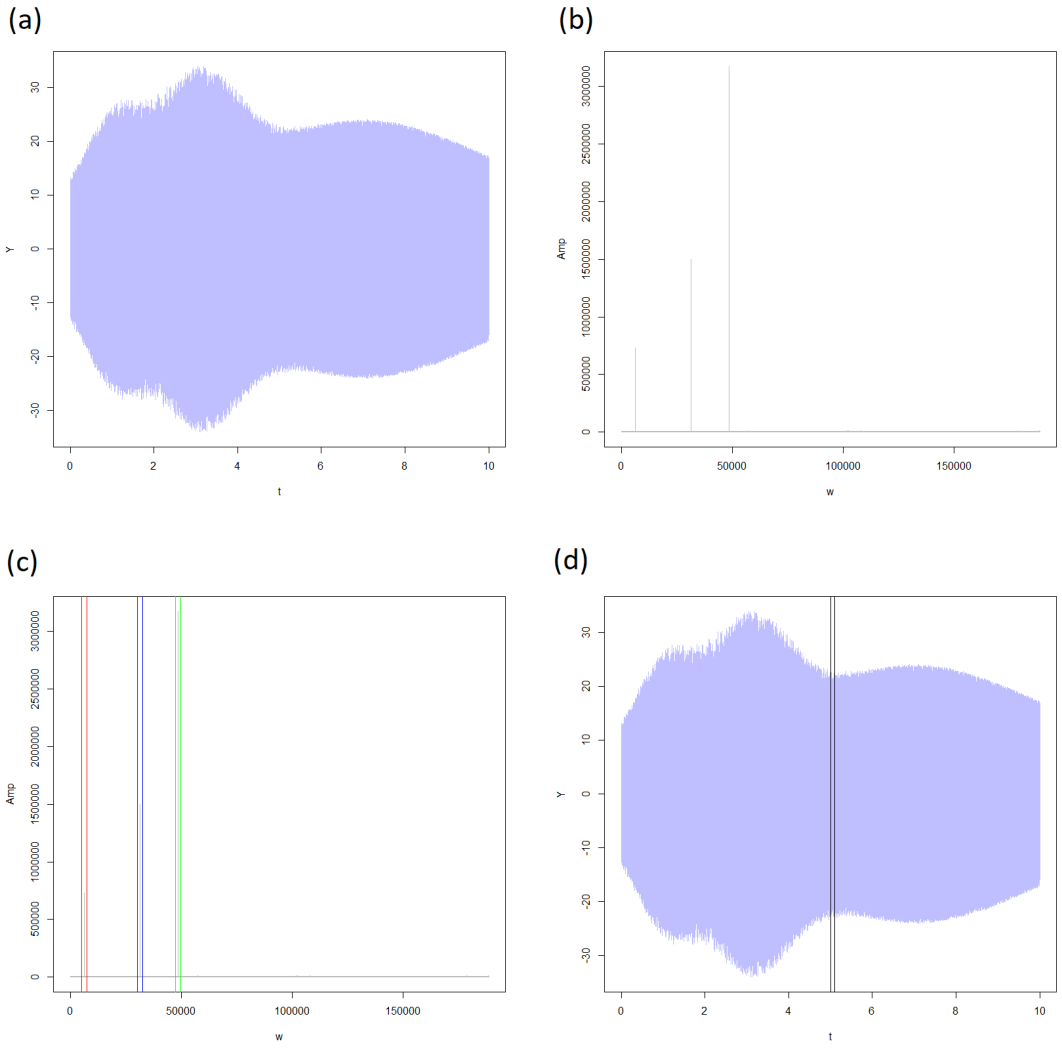

Figure 1: (a)Plot of Y vs t generated using R. (b)Amplitude spectra of the plot in (a). (c)frequency windows selected for the frequencies in (b). (d)real space xbox window size created for (a)

As shown in fig.1a, a trimodal signal is visible and as is derived from the code in listing.1 there exist only three frequencies in the `fcomp` variable. To check the mappings of the three frequency components in fig.1b, three frequency windows were created.

```
##Plotting the frequency windows
fY <- ft(t,Y,w=T) #function available with StatsChitran
plot(fY$wf, abs(fY$fy), col= 'grey', typ='l', xlab='w', ylab='Amp')

##for the 1000Hz freq
abline(v=2*pi*800, col='red')
abline(v=2*pi*1200, col='red')

##for the 5000Hz freq
abline(v=2*pi*4800, col='blue')
abline(v=2*pi*5200, col='blue')

##for the 7734Hz freq
abline(v=2*pi*7534, col='green')
abline(v=2*pi*7934, col='green')
```

Listing 2: code for plotting the three frequency windows in fig.1c

As shown in listing.2, the code for plotting the lines around the relevant frequencies (see fig.1c) were used. The code for plotting the real-space window in fig.1d is shown in listing.4.
As seen in listing.2, the frequency windows selected were $\omega = 2\pi(800 - - - -1200)$ for the 1000Hz peak, $\omega = 2\pi(4800 - - - -5200)$ for the 5000Hz peak and $\omega = 2\pi(7534 - - - -7934)$ for the 7734Hz peak rerspectively.
Hence, the selected windows were equispaced in f/ω space with a $\Delta\omega = 2\pi \times 400$ for each. This selection has consequences which will be shown.

3.1.1 Creating the data-structures required for the arguments of freq_map()

• The `w_int` dataframe
The function *freq_map()* first and foremost needs a `w_int` argument, which denotes the integration windows of the signals **in ω-space**[1]. `w_int` is always a n column dataframe with two observations. Here n is the no. of integration windows under consideration. For this case, as shown in listing.2 and fig.1c, $n = 3$, while the two observations are the two limits of the integration window shown in fig.1c

```
##building the datasets for freq_map()
#build dataframe for w_int (Integration window)
w1 <- c(2*pi*800, 2*pi*1200)
w2 <- c(2*pi*4800, 2*pi*5200)
w3 <- c(2*pi*7534, 2*pi*7934)
w_dat <- data.frame(w1, w2, w3)
```

Listing 3: code for the w_int dataframe for the three frequency windows in fig.1c

• The `xbox` number
The *freq_map()* function also needs to know the resolution of its window in real space as shown in fig.1d. **For this either `xbox` or `nbin` can be used**. The case with `xbox` is shown here.

[1]Donot use f-space values in `w_int`, use ONLY ω-space values, or else the results received will be incorrect

```
#build the xbox data(Box size in real space)
box_len <- 0.1
plot(t,Y, col=rgb(0,0,1,0.25), typ='l')
abline(v=5, col='black')
abline(v=5+box_len, col='black')
```

Listing 4: code for the xbox number fig.1d

The argument `xbox` is simply a number in R as shown. The first `abline(v=5, col='black')` was chosen just for convenience of representation. What is important to know is that `box_len <-0.1`. To show the comparative size of the window, the second vertical line was plotted using `abline(v=5+box_len, col='black')` (listing.4, fig.1d)

• The `color` vec
The `color` argument is a vector which is needed anytime a plot is required from *freq_map()*. `color` is necessarily of length n and a vector of colours[2]. As shown in listing.5, the `color` argument is just a vector of colors.

```
#build the color vector(use only when plt=T)
col_vec <- c('red','blue', 'green')
```

Listing 5: vector of colours to use in the plot

• Running the frequency mapping with `xbox`

```
##run the frequency mapping
L <- freq_map(t,Y, w_int = w_dat, xbox = box_len, color = col_vec, plt = T,
    plt.leg = T)
```

Listing 6: Run the frequency map with xbox

freq_map() is run as shown in listing.6. The `plt.leg` command forces a legend on the graph when set to `TRUE`.
The returned variable `L` is a list with length equal to n. Each index of the list contains the dataframes from the corresponding maps in `w_int`. The dataframe calculated as a result of the window `w_int[,i]` is present in `L[[i]]`.

• The `nbin` vector
The *freq_map()* can also be run with `nbin` instead of `xbox`. The function throws an error when both arguments are provided simultaneously, because they are conflicting arguments.
The `nbin` argument denotes the no. of periodicities of each frequency window that will be used to calculate the resolution of the window in realspace.[3]

```
#build the nbin vecor
x_bin <- c(100,100,100)
```

Listing 7: code for the nbin vector

As seen in lsiting.7, it is obvious that the length of `nbin` will be equal to n.

[2] n is the no. of integration windows or the no. of columns in `w_int`
[3] There are consequences to using either `nbin` or `xbox`.

• Running the frequency mapping with the `nbin` vector

```
##run the frequency mapping
L <- freq_map(t,Y, w_int = w_dat, nbin = x_bin, color = col_vec, plt = T, plt.leg
    = T)
```

Listing 8: Run the frequency map with nbin

3.1.2 Comparing the plots using xbox and nbin

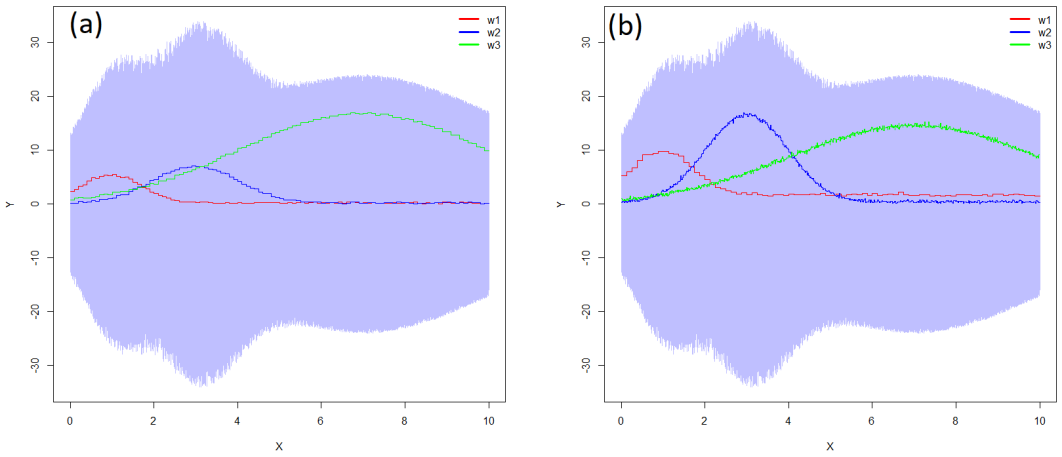

Figure 2: Overlay `freq_map` plots showing contribution of individual frequency windows with (a)`xbox` and (b)`nbin`

As can be seen, the plots in fig.2a and fig.2b generated with `xbox` and `nbin` respectively. Both plots show the expected frequency distributions, but with slight difference. Using the `xbox` in fig.2a, we have equal resolution(bin sizes) on all three frequency windows ω_1, ω_2 and ω_3, while with `nbin` in fig.2b we can see that the red curve (ω_1) has a worse resolution than the blue curve, ω_2, which in turn has a worse resolution than the green curve, ω_3, but the overall shape provided by the distribution tend to fit the original trimodal distribution better.

Thus, `xbox` and `nbin` allow trade-offs.

- `xbox`:
  - Provides uniform real-space resolution for all frequency windows
  - Intensity ratios of competing frequency windows distributions in real-space can't be trusted
- `nbin`:
  - Intensity ratios of competing frequency windows distributions in real-space can be trusted
  - Non-uniform resolution of real-space distributions from different frequency windows

This issue occurs, because different frequency windows have different wavelengths/Time periods and hence have a different number of waves/periods that can be fit into a given `xbox`, however with `nbin`, we can normalize for the number of waves/periods that we fit into our real space bins, but as a result have different bin sizes/resolution in real space.

3.1.3 Proof of concept

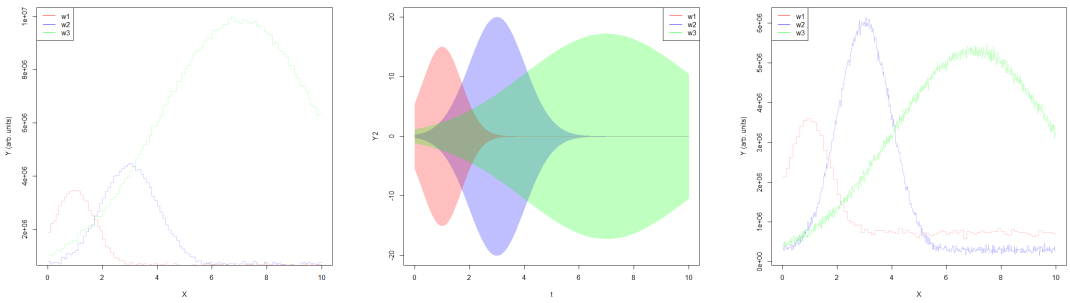

Figure 3: (a)Plot of t vs Y data generated using `xbox` in listing.10 (noise removed). (b)Plot of t vs Y data generated using listing.9 (noise removed). (c)Plot of t vs Y data generated using `nbin` in listing.11 (noise removed)

As a proof of concept fig.3b was plotted with the noise from the code removed as shown in listing.9. It can be seen that whether `xbox` or `nbin` is used, the generated data follows the waveforms well with respect to the position of the waveforms w.r.t the x-axis. However, as can be seen in fig.3a, the peak amplitudes are slightly distrorted in the case of using `xbox` rather than `nbin` (fig.3c).

```
library('StatsChitran') #Call library
ClearPlot() #Clear the plot
t <- seq(0,10, by=1/60000) #define the time axis
fcomp <- c(1000, 5000, 7734) #set which frequencies to build
Y1 <- 3*sin(2*pi*fcomp[1]*t)*gauss(t, amp =5, sig = 0.7, mu=1 ) #define signal w1
    or Y1
Y2 <- 5*sin(2*pi*fcomp[2]*t)*gauss(t, amp = 4, sig = 1, mu=3) #define signal w2 or
    Y2
Y3 <- 4*sin(2*pi*fcomp[3]*t)*gauss(t, amp = 4.3, sig = 3, mu=7) #define signal w3
    or Y3
plot(t,Y2, col=rgb(0,0,1,0.25), typ='l') #plot w2 or Y2
lines(t,Y1, col=rgb(1,0,0,0.25), typ='l') #plot w1 or Y1
#lines(t,Y2, col=rgb(0,0,1,0.25), typ='l')
lines(t,Y3, col=rgb(0,1,0,0.25), typ='l') #plot w3 or Y3
# Add a legend
legend("topright", legend = c("w1", "w2", "w3"), col = c("red", "blue", "green"),
    lty = 1, bg='transparent')
```

Listing 9: code for signal building (noise removed and coloured in accordance to ω_i). See fig.3b

```
#plot with xbox
box_len <- 0.1
L <- freq_map(t,Y, w_int = w_dat, xbox = box_len, color = col_vec, plt = F)
plot(L[[3]]$f_data_X, L[[3]]$f_data_Y, col=rgb(0,1,0,0.25), type = 'l', xlab =
    'X', ylab = 'Y (arb. units)')
lines(L[[1]]$f_data_X, L[[1]]$f_data_Y, col=rgb(1,0,0,0.25))
lines(L[[2]]$f_data_X, L[[2]]$f_data_Y, col=rgb(0,0,1,0.25))
legend("topleft", legend = c("w1", "w2", "w3"), col = c("red", "blue", "green"),
    lty = 1, bg='transparent')
```

Listing 10: code for frequency mapping using xbox (noise removed and coloured in accordance to ω_i). See fig.3a

```
#plot with nbin
x_bin <- c(100, 100, 100)
L <- freq_map(t,Y, w_int = w_dat, nbin = x_bin, color = col_vec, plt = F)
plot(L[[2]]$f_data_X, L[[2]]$f_data_Y, col=rgb(0,0,1,0.25), type = 'l', xlab =
    'X', ylab = 'Y (arb. units)')
lines(L[[1]]$f_data_X, L[[1]]$f_data_Y, col=rgb(1,0,0,0.25))
lines(L[[3]]$f_data_X, L[[3]]$f_data_Y, col=rgb(0,1,0,0.25))
legend("topleft", legend = c("w1", "w2", "w3"), col = c("red", "blue", "green"),
    lty = 1, bg='transparent')
```

Listing 11: code for frequency mapping using nbin (noise removed and coloured in accordance to ω_i). See fig.3c