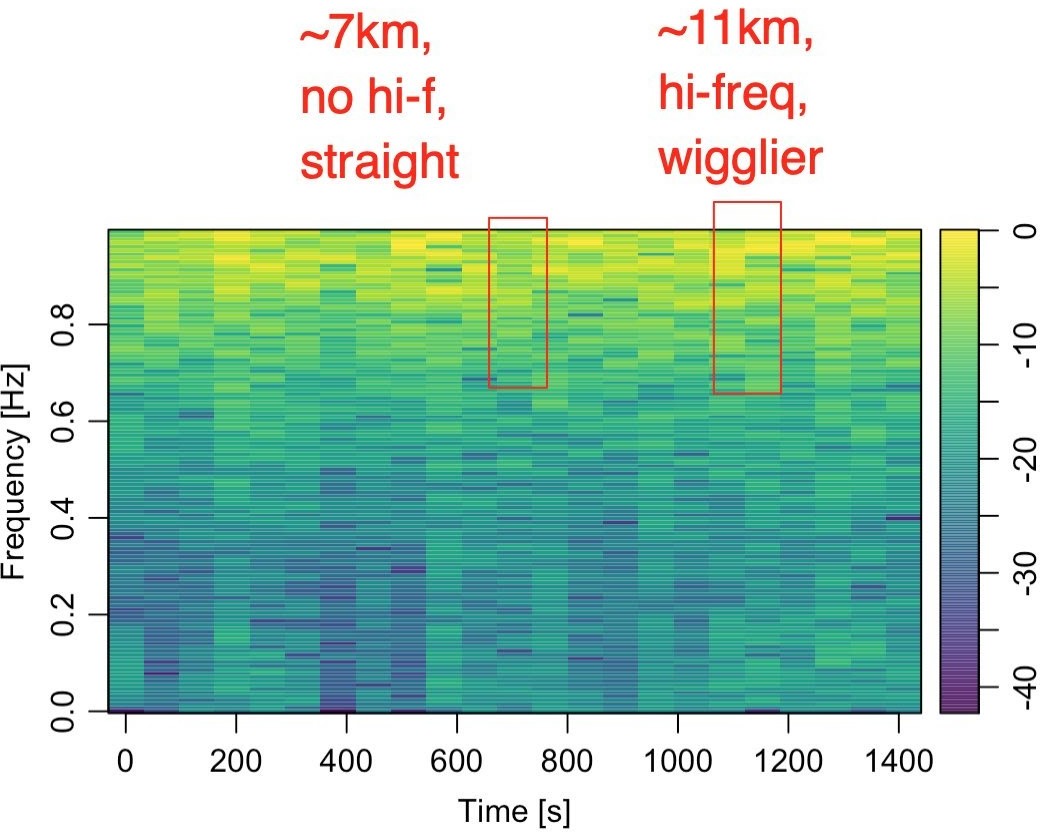
If I’m reading it right, I think the following spectrogram *does* show some possible differences in wiggliness for different segments along the stage?



The question then becomes: what signal (as a function of distance along line) to use? The above spectrogram is based on the perpendicular distance of the route from the straight line connecting the start and end points of the route.

# trj is a trajr route

straight = st\_linestring(data.matrix(rbind(head(trj[,c('x','y')], 1),

tail(trj[,c('x','y')], 1))))

straight\_sf = st\_sfc(straight,

crs=st\_crs(utm\_routes))

trj\_d = TrajRediscretize(trj, 10) utm\_discretised = trj\_d %>%

sf::st\_as\_sf(coords = c("x","y")) %>% sf::st\_set\_crs(st\_crs(utm\_routes[route\_index,]))

# Get the rectified distance from the midline

# Can we also get whether it's to left or right? perp\_distances = data.frame(d\_ = st\_distance(utm\_discretised,

straight\_sf))

# Returned distance is given as units perp\_distances$d = as.integer(perp\_distances$d\_)

perp\_distances$i = 10 \* (1:nrow(perp\_distances))

#perp\_distances$i = units::set\_units(10 \* (1:nrow(perp\_distances)), 'm')

We can then do something like a low pass filter:

library(signal)

# High pass filter

bf <- butter(2, 0.9, type="high") perp\_distances$d\_hi <- filter(bf, perp\_distances$d)

and generate the spectrogram show above:

# We could just plot this direct spec = specgram(perp\_distances$d\_hi)

library(oce)

# discard phase information P = abs(spec$S)

# normalize P = P/max(P)

# convert to dB P = 10\*log10(P)

# config time axis t = spec$t

# plot spectrogram imagep(x = t,

y = spec$f, z = t(P),

col = oce.colorsViridis, ylab = 'Frequency [Hz]', xlab = 'Time [s]', drawPalette = T, decimate = F

)

However, it would possibly make more sense to use something line the angle of turn, convexity index, or radius of curvature at each 10m step as the signal…

Hmmm…Spectogram in R

Read an audio file

First we have to find a signal to plot. I’m going to take an example of a right whale upcall we recorded off the Scotian Shelf a couple years ago, but *hopefully* this will work for any .wav file.

library(tuneR, warn.conflicts = F, quietly = T) # nice functions for reading and manipulating .wav files

# define path to audio file

fin = '~/Data/online\_sound\_examples/right\_whale\_upcall1.wav'

# read in audio file

data = readWave(fin)

# extract signal

snd = data@left

# determine duration

dur = length(snd)/data@samp.rate

dur # seconds

## [1] 3.588

# determine sample rate

fs = data@samp.rate

fs # Hz

## [1] 2000

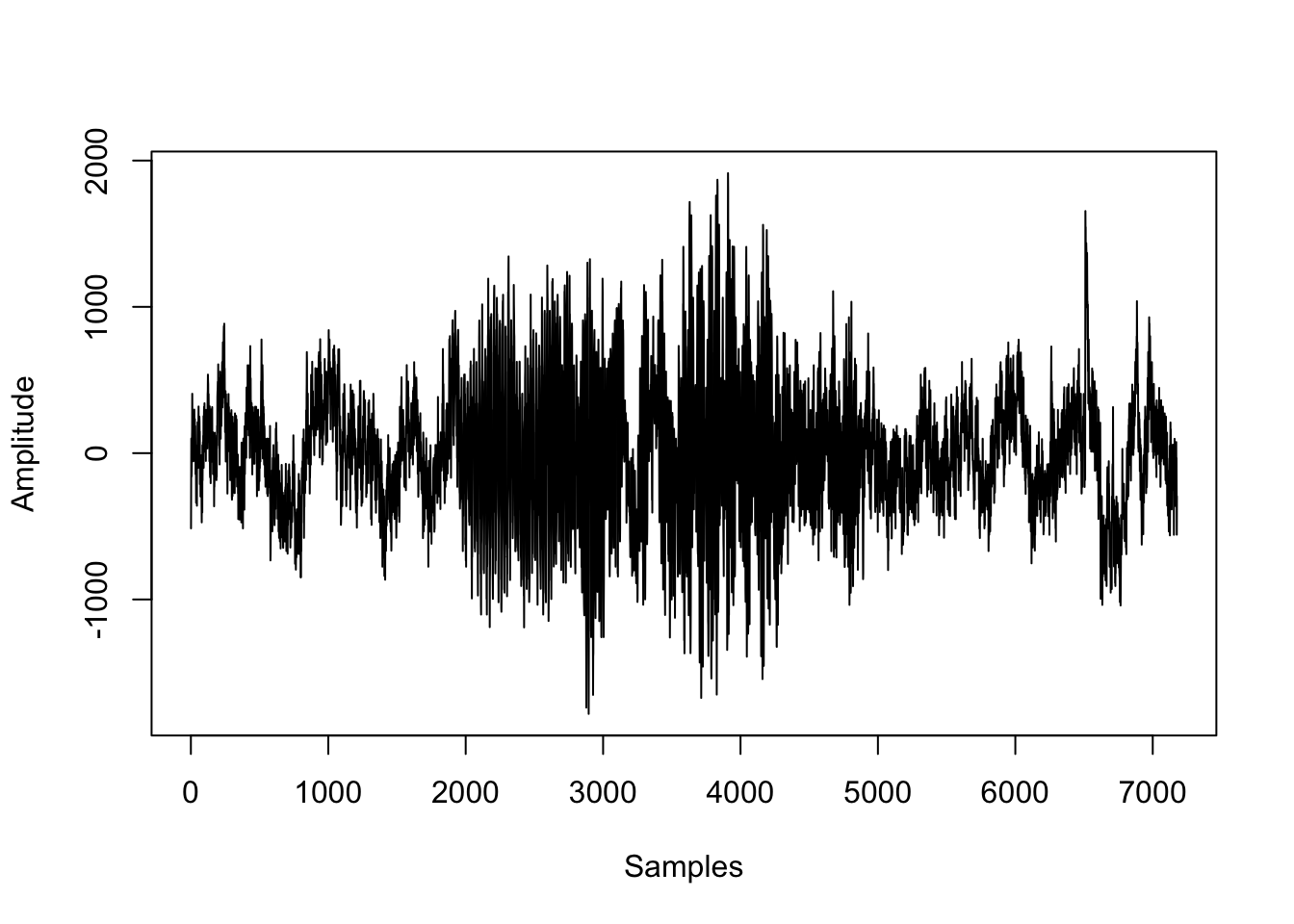
Plot waveform

# demean to remove DC offset

snd = snd - mean(snd)

# plot waveform

plot(snd, type = 'l', xlab = 'Samples', ylab = 'Amplitude')



Define spectrogram parameters

The next step is to define the parameters used to construct the spectrogram. What each parameter is and how to choose them appropriately is the topic for another discussion. I’ll just pick some reasonable values as a demonstration.

# number of points to use for the fft

nfft=1024

# window size (in points)

window=256

# overlap (in points)

overlap=128

Plot the spectrogram

library(signal, warn.conflicts = F, quietly = T) # signal processing functions

library(oce, warn.conflicts = F, quietly = T) # image plotting functions and nice color maps

# create spectrogram

spec = specgram(x = snd,

n = nfft,

Fs = fs,

window = window,

overlap = overlap

)

# discard phase information

P = abs(spec$S)

# normalize

P = P/max(P)

# convert to dB

P = 10\*log10(P)

# config time axis

t = spec$t

# plot spectrogram

imagep(x = t,

y = spec$f,

z = t(P),

col = oce.colorsViridis,

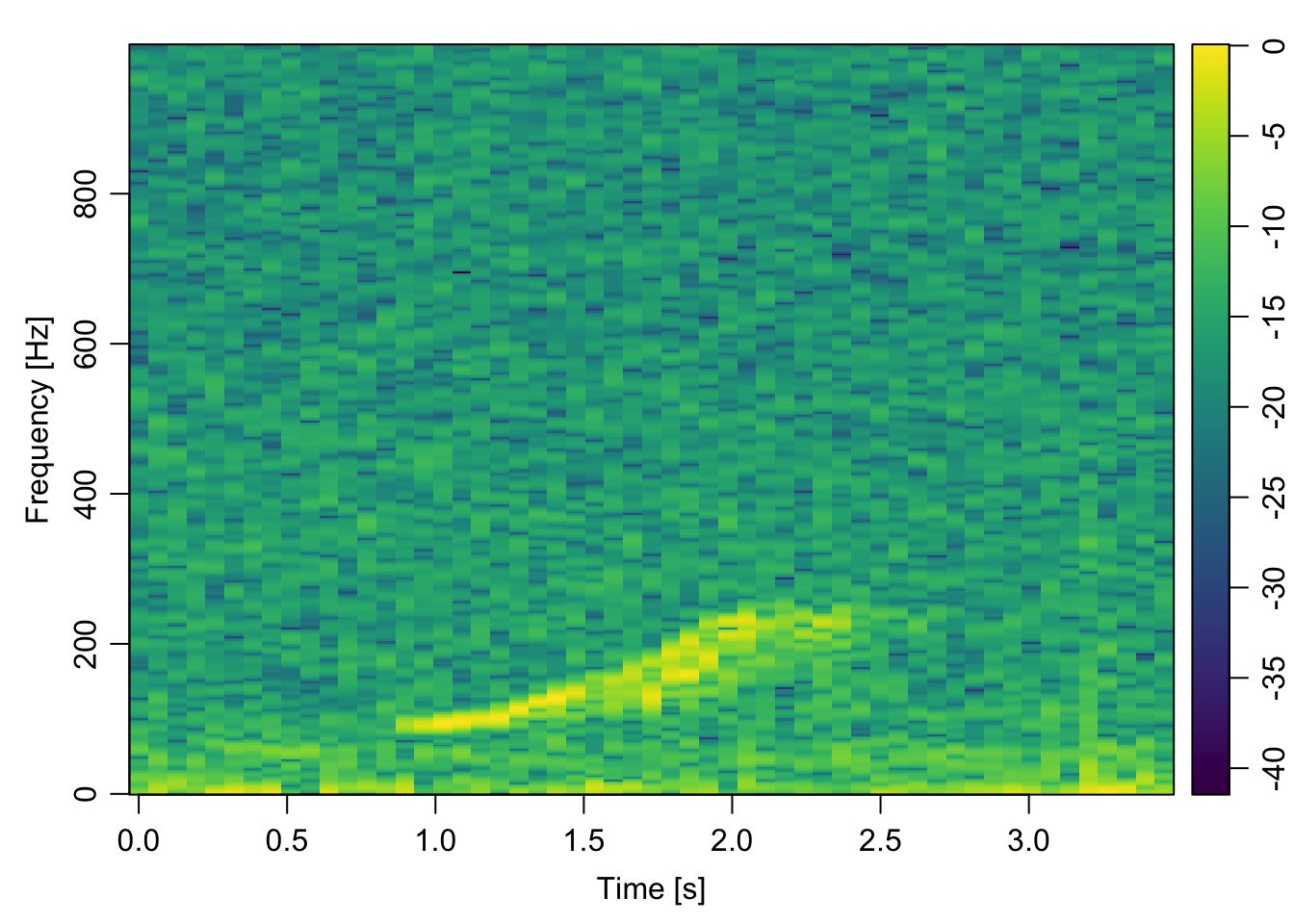
ylab = 'Frequency [Hz]',

xlab = 'Time [s]',

drawPalette = T,

decimate = F

)



That’s it! Happy plotting :)

Bonus: functionize it

Here’s an example of how to put everything above into a tidy plotting function. I’ve made a few changes here that were specific to my application at the time:

* The main data input is a Formal class Wave object in R (i.e. input from tuneR), but you could easily change things around to accept the path to a .wav file.
* If the t0 input is 0, the time axis is in seconds elapsed since the start of the file, but you can also pass a POSIXct value to display the time/date.
* I have included switches to optionally plot the spectrogram (plot\_spec), normalize the waveform (normalize), and/or return the spectrogram data (return\_data).
* The ... means that I can pass additional arguments to the imagep() plotting function
* I took some stylistic liberties and chose to create a spectrogram against a dark background, just because I like the way it looks

spectro = function(data, nfft=1024, window=256, overlap=128, t0=0, plot\_spec = T, normalize = F, return\_data = F,...){

library(signal)

library(oce)

# extract signal

snd = data@left

# demean to remove DC offset

snd = snd-mean(snd)

# determine duration

dur = length(snd)/data@samp.rate

# create spectrogram

spec = specgram(x = snd,

n = nfft,

Fs = data@samp.rate,

window = window,

overlap = overlap

)

# discard phase info

P = abs(spec$S)

# normalize

if(normalize){

P = P/max(P)

}

# convert to dB

P = 10\*log10(P)

# config time axis

if(t0==0){

t = as.numeric(spec$t)

} else {

t = as.POSIXct(spec$t, origin = t0)

}

# rename freq

f = spec$f

if(plot\_spec){

# change plot colour defaults

par(bg = "black")

par(col.lab="white")

par(col.axis="white")

par(col.main="white")

# plot spectrogram

imagep(t,f, t(P), col = oce.colorsViridis, drawPalette = T,

ylab = 'Frequency [Hz]', axes = F,...)

box(col = 'white')

axis(2, labels = T, col = 'white')

# add x axis

if(t0==0){

axis(1, labels = T, col = 'white')

}else{

axis.POSIXct(seq.POSIXt(t0, t0+dur, 10), side = 1, format = '%H:%M:%S', col = 'white', las = 1)

mtext(paste0(format(t0, '%B %d, %Y')), side = 1, adj = 0, line = 2, col = 'white')

}

}

if(return\_data){

# prep output

spec = list(

t = t,

f = f,

p = t(P)

)

return(spec)

}

}

**Test it!**

Here’s the function in action on the same data file

# call the spectrogram function

spectro(data,

nfft=1024,

window=256,

overlap=128,

t0=as.POSIXct('2014-01-02 11:01:58'),

plot\_spec = T,

normalize = T,

return\_data = F

)

