Fundamental plots for electrophysiological data

Vikram B. Baliga

2023-03-30

Contents

Chapter 1

About

Until this statement is deleted from this page, please consider everything you see here a work in progress. Ultimately, this site will provide a walkthrough on how to produce fundamental plots from electrophysiological data. The content was initialized using a bookdown[1] template; accordingly, as this site remains in a developmental stage, content from the template may linger.

The original content written here is intended to instruct trainees in the Altshuler Lab at the University of British Columbia to take raw recorded data from electrophysiological examinations and then produce preliminary plots that help characterize the recorded neural spike data.

To get started, please use the left navigation and work through chapters in order.

Citation

TBD

License

[1] https://bookdown.org/
[2] https://creativecommons.org/licenses/by/4.0/
[3] https://github.com/flightlab/ephys_fundamental_plots

Chapter 2

Preface

2.1 R packages & versioning

The R packages listed below will be necessary at some point over the course of this book. I recommend installing them all now. The block of code below is designed to first check if each of the listed packages is already installed on your computer. If any is missing, then an attempt is made to install it from CRAN. Finally, all of the packages are loaded into the environment.

```
## Specify the packages you'll use in the script
packages <- c("tidyverse",
              "zoo",
              "gridExtra",
              "R.matlab",
              "cowplot",
              "easystats",
              "circular",
              "splines",
              "MESS", ## area under curve
              "zoo" ## rolling means
)
## Now for each package listed, first check to see if the package is already
## installed. If it is installed, it's simply loaded. If not, it's downloaded
## from CRAN and then installed and loaded.
package.check <- lapply(packages,
                        FUN = function(x) {
                          if (!require(x, character.only = TRUE)) {
                            install.packages(x, dependencies = TRUE)
                            library(x, character.only = TRUE)
                          }
```

```
                        }
)
```

I will use the `sessionInfo()` command to detail the specific versions of packages I am using (along with other information about my R session). Please note that I am not suggesting you obtain exactly the same version of each package listed below. Instead, the information below is meant to help you assess whether package versioning underlies any trouble you may encounter.

```
## R version 4.2.2 (2022-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19045)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] splines   stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] MESS_0.5.9         circular_0.4-95    see_0.7.4          report_0.5.6
##  [5] parameters_0.20.2  performance_0.10.2 modelbased_0.8.6   insight_0.19.0
##  [9] effectsize_0.8.3   datawizard_0.6.5   correlation_0.8.3  bayestestR_0.13.0
## [13] easystats_0.6.0    cowplot_1.1.1      R.matlab_3.7.0     gridExtra_2.3
## [17] zoo_1.8-11         lubridate_1.9.2    forcats_1.0.0      stringr_1.5.0
## [21] dplyr_1.1.0        purrr_1.0.1        readr_2.1.4        tidyr_1.3.0
## [25] tibble_3.1.8       ggplot2_3.4.1      tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] R.utils_2.12.2     ggstance_0.3.6     yaml_2.3.7         pillar_1.8.1
##  [5] backports_1.4.1    lattice_0.20-45    glue_1.6.2         digest_0.6.31
##  [9] polyclip_1.10-4    colorspace_2.1-0   htmltools_0.5.4    Matrix_1.5-1
## [13] R.oo_1.25.0        pkgconfig_2.0.3    labelled_2.10.0    broom_1.0.3
## [17] haven_2.5.1        bookdown_0.32      xtable_1.8-4       mvtnorm_1.1-3
## [21] scales_1.2.1       tweenr_2.0.2       ggforce_0.4.1      tzdb_0.3.0
## [25] timechange_0.2.0   emmeans_1.8.4-1    farver_2.1.1       generics_0.1.3
## [29] ellipsis_0.3.2     withr_2.5.0        geepack_1.3.9      cli_3.6.0
## [33] magrittr_2.0.3     estimability_1.4.1 evaluate_0.20      R.methodsS3_1.8.2
```

```
## [37] fansi_1.0.4        MASS_7.3-58.2      geeM_0.10.1        tools_4.2.2
## [41] hms_1.1.2          lifecycle_1.0.3    munsell_0.5.0      compiler_4.2.2
## [45] rlang_1.0.6        grid_4.2.2         ggridges_0.5.4     rstudioapi_0.14
## [49] mosaicCore_0.9.2.1 rmarkdown_2.20     boot_1.3-28        gtable_0.3.1
## [53] R6_2.5.1           knitr_1.42         fastmap_1.1.1      utf8_1.2.3
## [57] ggformula_0.10.2   stringi_1.7.12     Rcpp_1.0.10        vctrs_0.5.2
## [61] tidyselect_1.2.0   xfun_0.37          coda_0.19-4
```

2.2 %not_in%

This guide will also rely on this handy function, which you should add to your code:

```
`%not_in%` <- Negate(`%in%`)
```

This simple operator allows you to determine if an element does not appear in a target object.

Chapter 3

Spike sorting

We'll cover how to spike sort using two programs: 1) Spike2[1] (written by Tony Lapsansky) and 2) Neuralynx[2] (written by Eric Press).

The function of spike sorting is to isolate action potentials from the background voltage signal. These methods use the shape of the waveform to detect and distinguish the spiking activity of each neuron recorded by an electrode.

3.1 Spike2

Written by Tony Lapsansky, February 24, 2023

These instructions assume that you have been given a Spike2 recording file (extension `.smrx`) and asked to spike sort.

Spike2 includes a detailed description of the program, accessible by clicking `Help` → `Index`

3.1.1 File naming conventions:

- Use the name structure `YEARMODA_sequence_investigator`
- Save data in the corresponding directory `"C:\InvestigatorName\ephys\YEAR-MO-DA"`

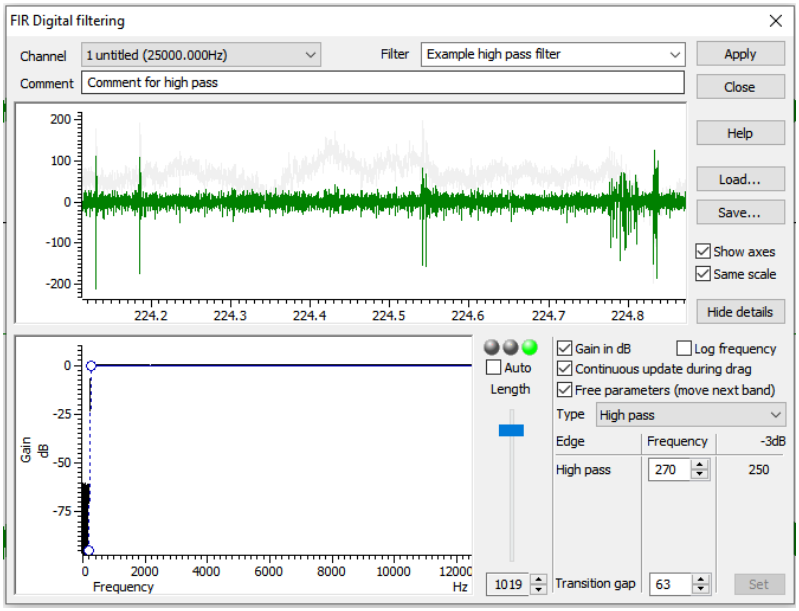
[1] https://ced.co.uk/products/spkovin
[2] https://neuralynx.com/

Figure 3.1: Spike2

3.1.2 Spike sorting with Spike2

1. **Open the main `Spike2` file** for the recording. This file should have the extension `.smrx`.
2. **Apply a digital high pass filter**, if needed. Note: if the data were collected with the high pass filter set at greater than 100 Hz (no LFP signal) then proceed to step 3.

   - Right click on the raw data channel (typically Ch1) and select `FIR Digital Filters…`. We want to use an FIR filter rather than an IIR filter as the latter can introduce a variable time lag in the resulting data (see Spike 2 `Help` → `Index` → `Digital Filter` for full explanation).
   - Under the pull down menu for `Filter`, change the filter from `Example low pass filter` to `Example high pass filter`.
   - Select the `Show Details` button in the bottom right.
   - Adjust blue slider change the filter length. Shift the slider until the coloured dots above the slider from red to yellow to green. This removes wobbles in the data. Use the minimum level (~1019) to achieve green. Fine adjustments can be made just under the slider.



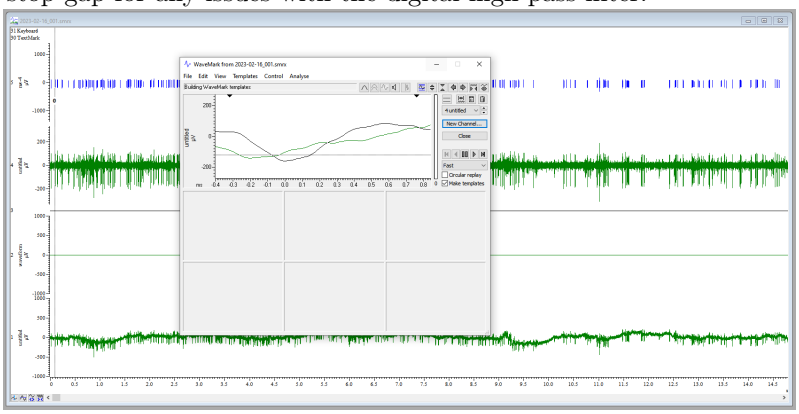
- Hit `Apply`
- Set `Destination` to the next available channel (typically Channel 4)
- Click `Okay`
- Close the filtering window. You are given the option to save the filter. This is unnecessary.

3. **Setting the threshold for spike identification**

- Right click on the filtered channel and select `New WaveMark`
- Clear previous templates if any are present. To do so, select the trash can icon within each template. These may be present from a previous session.
- Locate your cursor position, indicated by the vertical dashed line in the main window (typically found at time 0)
- Slide the dashed line horizontally through the trace to observe potential spikes as determined by the default upper and lower thresholds.
- Right click the upper bound marker (the upper horizontal dashed line in the `WaveMark` window) and select `Move Away`. We will rely on the lower bound to identify spikes for sorting, as the activity above baseline is typically closer in magnitude to the background.
- Slide the dashed line horizontally through the trace to observe potential spikes as determined by the lower threshold alone.
- Adjust the lower threshold to catch spikes of interest. This threshold will vary based based on the distance between the electrode and the neuron, the quality of the isolate, and the level of background noise. Values between 50 mV and 200 mV are typical.Set the lower bound so that spikes of interest are included and ambiguous spikes are excluded.

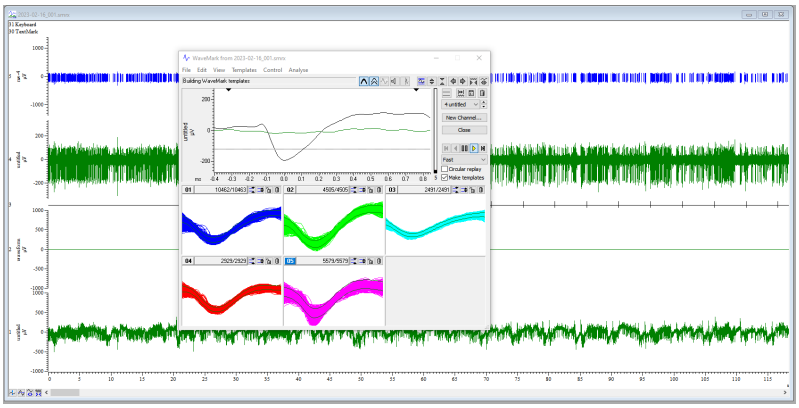
4. **Designing the spike template**

   - Move the cursor to a characteristic spike. In the upper window, you will see the provisional template. Click and hold on the trace in the upper window and drag it to the first available spot in the lower, template window.
   - To set parameters for spike sorting, click on the button just to the left of the trash can icon (on the top half, upper right of the `WaveMark` window). This is the "parameters dialog" button. This opens a template settings window.
   - For the line `Maximum amplitude change for a match` enter a value between `10` and `20`. This will allow a spike that fits a template to vary in maximum amplitude by up to 10-20%.
   - For the line `Remove the DC offset before template matching`, confirm that the box is checked. This means that Spike2 will account for abrupt shifts in the signal baseline before template matching. This is a stop-gap for any issues with the digital high pass filter.
   - Click `OK`.



5. **Spike sorting**

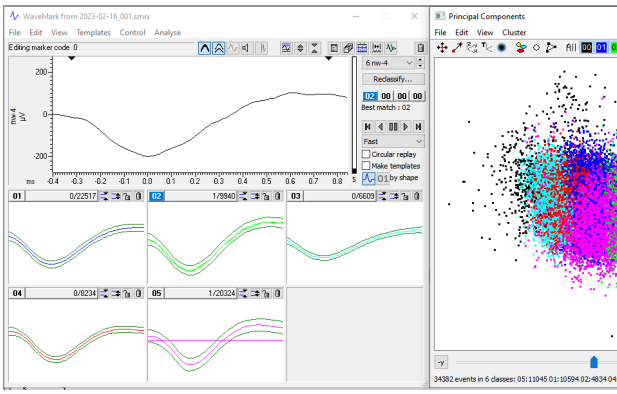
   - Back in the `WaveMark` window, make sure that the box `Circular replay` is **unchecked**. If checked, spike sorting will loop indefinitely.
   - Ensure that the vertical cursor on the main window is at time zero (or the first spike) so that no spikes are missed.
   - Back in the `WaveMark` window, make sure that the box `Make templates` is **checked**. If unchecked, only spikes corresponding to the provisional template will be identified. We want to let spike2 help us to identify potential multi-unit activity.
   - Hit the play button , which is called "run forward". Spike sorting will proceed for several minutes. Each identified spike will appear briefly in the `WaveMark` window and will be assigned to a template.



In this image, I have selected options for `Overdraw` *and* `Show template limits`

6. **Merge, delete, and save templates**

- After spike sorting has completed, select `New Channel` on the `WaveMark` window to place the spike sorted data in the next available channel (typically, Channel 5)
- Close the existing `WaveMark` window.
- Right click on the **spike sorted channel** and select `Edit WaveMark`.
- Within the `WaveMark` window, go the pull down menu `Analyse` and select `Principal components`. Select `OK`. This opens a window containing a principal component analysis of all spikes colored by their assigned template.
- Rotate around all three axes to determine if there is one, two, or more clusters. In theory, each cluster corresponds to a single neuron. Often, spikes are categorized into multiple templates, but realistically correspond to the activity of a single neuron.
- Identify templates that should be deleted and those that should be merged. We will delete spikes corresponding to templates that are sparse and peripheral.
- Delete the template(s) in the `WaveMark` window by selecting that template's trash can icon.
- Merge templates by dragging them into the same window
- Hit the `reclassify` button in the `WaveMark` window to commit these



changes to the data in the main window. *In this example, we have good evidence from the PCA to merge these five templates.*

7. **Export the spike-sorted data**
    - `File → Export As`
    - Select `.mat` (`matlab` data)
    - Use the same filename and location but with the `.mat` extension.
    - Hit `Save`
    - Select `Add` for `All Channels`
    - Click `Export`
    - Click `OK` (this will take several minutes)

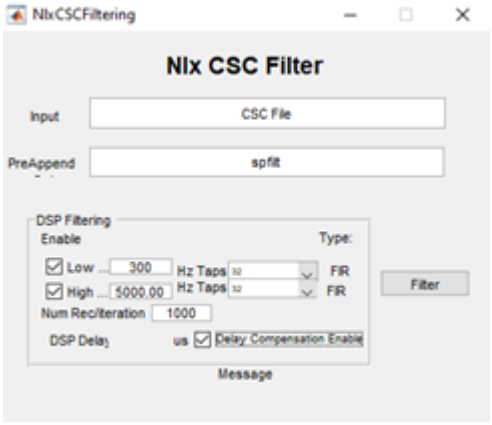
Note: May need to select an earlier MATLAB file convention to work with R.

3.2 Neuralynx

Written by Eric Press, November 11, 2022

1. Spike sorting database:
    1. Check the column labelled `Sorting status` to find days of recording that are `cued` meaning they are ready to be sorted. Recordings are cued for spike sorting once information about the recording has been added to the database. This includes observations from the day's recording, whether the electrode position was moved from the previous recording, and the stimulus condition for each recording. The recordings are stored at the following location and are named/organized by date and time of recording:
    `Computer/LaCie (D:)/Eric's data/nlx_recordings`
2. Filtering the raw traces (CSCs):
    1. Use the `NlxCSCFiltering` tool on any Windows machine to run a band-pass filter on input `CSC` files.

2. Choose all the `CSC` files for a given recording, change the `PreAppend` field to `spfilt`, which stands for spike-filtered and adjust the `DSP` filtering fields to match the image to the right. This selects for frequencies in the raw traces where spikes will be found, but removes low frequency (LFP) and high frequency components of the traces.



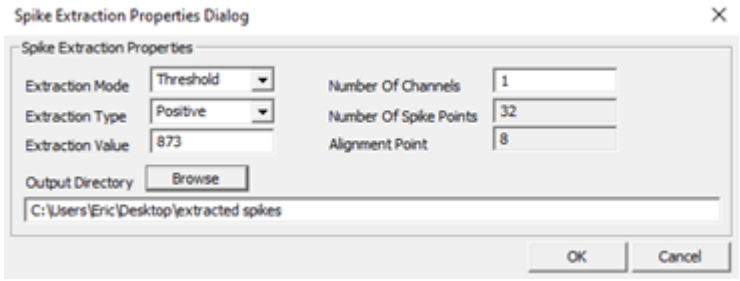
3. Examine the filtered traces:

    1. Take a closer look at the filtered traces (Open in `Neuraview` on any Windows machine) and determine which channels are likely to have isolatable spikes and how many distinct spikes there might be. It helps to keep `Neuraview` open when setting thresholds in the next step.

4. Spike detection from filtered traces:

    1. Use the `CSCSpikeExtractor` tool on any Windows machine to detect spikes above or below a given µV) threshold. The units displayed in the program will be AdBitVolts which are simply 10.92x from the µV value.
    2. Based on the filtered traces, within `CSCSpikeExtractor`, set the spike extraction properties (`Spike Extraction -> Properties` OR `Ctrl+P`) as shown above. The `Extraction Value` is set to 10.92x the µV you chose by viewing the filtered traces.
    3. Press `Ctrl+S` to extract spikes from the selected file at the desired settings. The resulting file will be placed in the `extracted spikes` filter on the `Desktop`.
    4. Create subfolders in the recording folder for each threshold and move the extracted spikes at each threshold into the appropriate folder. These spike-detected files will be used for spike sorting in the next step.
    5. **If it helps with detecting real spike waveforms while eliminating noise, run recordings through spike detection at multiple threshold (positive or negative) such that only all puta-**

tive neurons are accounted for a minimal noise is detected.



5. Spike sorting:

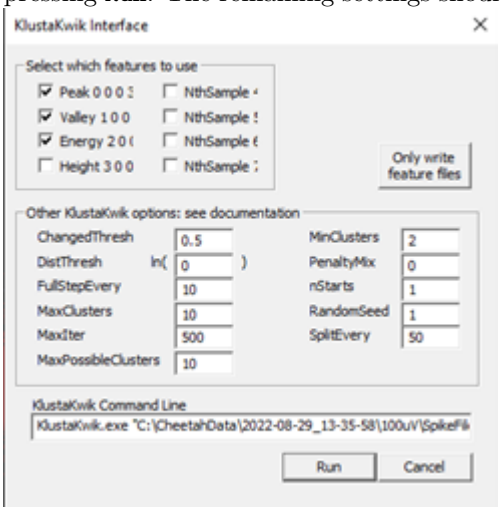
    1. Open the extracted spikes in `Spikesort3D` on either the Neuralynx machine or another Windows machine that has an active `SpikeSort3D` licence. You can also use `TeamViewer` to control the Neuralynx machine but this works much better with another Windows machine.
    2. Press OK when the feature selection window appears. If you want to select alternate features to display, select them from the list provided. Sometimes it can be helpful to use PCA1 – 3 in isolating neurons but often it makes things more challenging.
    3. Using the 3D Plot, examine the clustering of spikes. Follow the image below to aid in interacting with the 3D plot (MB = the scroll wheel button i.e. middle mouse button). You can change the features displayed on each axis with `Q/W`, `A/S`, and `Z/X` respectively. Also, `Ctrl+P` brings up a window that allows you to change the size and opacity of points on the plot (I find `size = 2`, `alpha = 0.5` works well to improve visual definition of the clusters). If distinct clusters are difficult to see, find the combination of 3 features that produces the most noticeable clustering or the greatest spread of points in the space. The features displayed in the 3D plot are shown at the top left of the plot (i.e. X(3) Height # # # #). Use those features for the next step.

    3D Plot Movement and interaction

MB + X or Y axis mouse movement	Pan the display
MB + Shift + X/Y movement	Pan the display locked to one axis
MB + Alt + X/Y movement	Rotate the display
MB + Shift + Alt + X/Y movement	Rotate the display locked to one axis
MB + Ctrl + Alt + Y movement	Zoom display in or out
Mouse scroll wheel	Zoom display in or out
Ctrl + Left Click	Insert convex hull point
Right Click	Cancel current convex hull input
Shift + Right Mouse Button	Show spike nearest cursor

    4. Run `KlustaKwik` (`Cluster → Autocluster using KlustaKwik`) and select the 3 features that generate the most clearly separable

clusters on the 3D view – often, the first 3 (`Peak`, `Valley`, `Energy`) do a decent job. Change the `MaxPossibleClusters` to `10` before pressing `Run`. The remaining settings should match the image below.



5. Following calculations, use the `Waveform` window and the 3D plot to group the distinct clusters into what you believe are waveforms produced by distinct neurons. Use the number keys to highlight distinct clusters and `Ctrl+M` to merge clusters together. `Ctrl+C` copies the selected cluster and can be used to split a cluster into 2 if you believe portions of the cluster belong to distinct putative neurons. This step takes some practice. You can use `Ctrl+Z` to undo only one move. Otherwise, you may need to exit without saving and start again at step 4. Save with `Ctrl+S` often and click OK to overwrite the file.
6. Once you are satisfied with the waveforms left, note how many there are, and whether it seems possible that some of the groups belong to the same neuron. Consider what you know about excitable membranes to make these decisions. Fill out the `Spike Sorting Database` with the information used to reach this point. This includes, the threshold(s), # of clusters, # of putative neurons (often 1 less than the # of clusters because it would be a stretch to include the smallest amplitude waveform as a distinct, separable neuron), and any else to note from performing sorting.
7. Save each cluster to its own spike file (`File → Save Multiple Spike Files`)
8. Open the separate spike files you just created, along with the original filtered trace in `Neuraview`. Scroll along the recording and examine if the sorting you performed seems believable. Do the spikes in different rows really seem like they're different in the filtered trace? Do some spikes not seem like real spikes? If anything seems amiss, make the appropriate merges in `SpikeSort3D` before proceding.
9. Export the relevant data from the sorting. Perform the following:

1. `File → Save ASCII Timestamp Files`
2. `File → Save Multiple Spike Files`
3. `File → Save ASCII Avg Waveforms`
4. Also, save the file itself with `Ctrl+S`

10. Lastly, bring up all the waveforms together on the waveform plot. Take a screenshot and save it to the folder where the extracted spikes (and now timestamps files) are stored.

6. Moving sorted files to other locations:

    1. Once a chunk of recordings have been sorted, copy/paste the entire recording file to Eric's orange 1TB storage drive (Lacie). Place them in the following folder: `Eric's data/sorted_recordings`

Chapter 4

Quick version - condensed plots

Coming soon

Chapter 5

Data wrangling

Once data have been spike sorted, we are ready to begin working in `R`. To get to a point where meaningful preliminary plots can be produced, a few things need to be addressed:

1) Labeling the time series of spike & photodiode data based on the stimulus that appears on screen (i.e., matching the log file to the data file). This includes labeling phases (like "blank", "stationary", and "moving") along with experimental metadata such as SF, TF, and stimulus orientation (direction).

2) Re-organizing the spike & photodiode so that separate replicates of a stimulus run are uniquely labelled and then arranged together.

3) Binning the data into 10- and 100-ms data sets, and then exporting CSVs of the unbinned, 10-ms-binned, and 100-ms-binned data. This will be highly useful for situations where you are handling multiple data files (e.g., from different recording days), in which case it is likely that your machine will not be able to store all objects in RAM.

   Before proceeding any further, please ensure you have installed and loaded all the necessary `R` packages as detailed in the Preface section of this guide.

5.1 Import example file and metadata

We will use a recently-collected data file and its corresponding metadata file to showcase the fundamentals of wrangling ephys data into a more easily plot-able format.

5.1.1 Identify files to import

The following code is based on the assumptions that:

1) Your files are stored in a directory entitled `/data`
2) The basename of each file (i.e., the name of the file, excluding the file extension) is identical for each set of spike sorted data and corresponding metadata log file (e.g., `2023-02-16_001.mat` and `2023-02-16_001.csv` have the same basename, which is `2023-02-16_001`).

```
## List all files of each file type
csv_files <-
  list.files("./data", pattern = ".csv",
             full.names = TRUE)
mat_files <-
  list.files("./data", pattern = ".mat",
             full.names = TRUE)

## Generate metadata tibbles for each file type
csv_file_info <-
  tibble(
    csv_files = csv_files,
    ## Extract the basename by removing the file extension
    basename =  basename(csv_files) %>% str_remove(".csv"),
    ## NOTE: PLEASE ADJUST THE FOLLOWING LINE TO BE ABLE TO EXCTRACT OUT THE
    ## DATE BASED ON YOUR NAMING CONVENTION
    basedate =  basename(csv_files) %>% str_sub(start = 1, end = 12)
  )
mat_file_info <-
  tibble(
    mat_files = mat_files,
    ## Extract the basename by removing the file extension
    basename =  basename(mat_files) %>% str_remove(".mat"),
    ## NOTE: AGAIN, PLEASE ADJUST THE FOLLOWING LINE TO BE ABLE TO EXCTRACT OUT
    ## THE DATE BASED ON YOUR NAMING CONVENTION
    basedate =  basename(mat_files) %>% str_sub(start = 1, end = 12)
  )

## Matchmake between .MAT data and .CSV log files
csv_mat_filejoin <-
  inner_join(csv_file_info, mat_file_info, by = "basename") %>%
  ## OPTIONAL STEP: remove any rows where either the .MAT or .CSV is missing
  drop_na()

## Store a vector of basenames in the environment. This will become useful later
```

```
base_names <- csv_mat_filejoin$basename

## Your end products from this code block should look something like:
csv_mat_filejoin
```

```
## # A tibble: 1 x 5
##   csv_files                 basename       basedate.x   mat_files        based~1
##   <chr>                     <chr>          <chr>        <chr>            <chr>
## 1 ./data/2023-02-16_001.csv 2023-02-16_001 2023-02-16_0 ./data/2023-02-~ 2023-0~
## # ... with abbreviated variable name 1: basedate.y
```

```
## and:
base_names
```

```
## [1] "2023-02-16_001"
```

5.1.2 Data import and preliminary labeling

We will now use the `R.matlab` package to import the `.mat` file into R and then label the spike and photodiode time series based on the information in the `.csv` log file

Because `.mat` files can be large, data import can take several minutes.

Please see in-line comments for further guidance

```
## Set up empty vectors that will collect sets of replicates that we will be
## splitting up
metadata_sets <- NULL
meta_splits <- NULL
data_splits <- NULL
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2426248 129.6    4179092 223.2  4179092 223.2
## Vcells 4198206  32.1   10146329  77.5  8185131  62.5
```

```
starttime <- Sys.time() ## Optional, will help you assess run time
for (i in 1:nrow(csv_mat_filejoin)) {

  ## Which file # are we working on?
  print(i)
```

```
## Set up temporary objects in which to eventually write data
csv_data_sets <- NULL
mat_data_sets <- NULL
joined_data_sets <- NULL

## Import the matlab file. This may take some time
mat_import <-
  R.matlab::readMat(csv_mat_filejoin[i,"mat_files"])

## Read in the corresponding csv log file
csv_data_sets[[i]] <-
  read_csv(as.character(csv_mat_filejoin[i,"csv_files"]),
           show_col_types = FALSE) %>%
  ## Rename columns for convenience
  rename(
    Spatial_Frequency = `Spatial Frequency`,
    Temporal_Frequency = `Temporal Frequency`,
    Cycles_Per_Pixel = `Cycles Per Pixel`
  )

## The log file does not have time = 0, so set up a separate tibble to
## add this info in later. Some of the metadata will just be filler for now.
initial <- tibble(
  Trial = "initialization",
  Spatial_Frequency =  csv_data_sets[[i]]$Spatial_Frequency[1],
  Cycles_Per_Pixel  =  csv_data_sets[[i]]$Cycles_Per_Pixel[1],
  Temporal_Frequency = csv_data_sets[[i]]$Temporal_Frequency[1],
  Direction  = csv_data_sets[[i]]$Direction[1],
  Time = 0.000
)

## Find photodiode
## It is almost always in channel 2, but we should be sure to check before
## extracting automatically
photod_default_channel <-
  mat_import[[stringr::str_which(names(mat_import), "Ch2")[1]]]
if (!photod_default_channel[1][[1]][1] == "waveform") {
  warning("File ", i,": Photodiode channel identity uncertain")
}

## Find spikes
## Similarly, spikes are almost always in channel 5, but we should check
spikes_default_channel <-
  mat_import[[stringr::str_which(names(mat_import), "Ch5")[1]]]
if("codes" %not_in% attributes(spikes_default_channel)$dimnames[[1]]) {
```

```
  warning("File ", i,": Sorted spikes channel identity uncertain")
}
## If that worked, see if we can automatically determine the "times" and
## "codes" slot numbers
times_location <-
  which(attributes(spikes_default_channel)$dimnames[[1]] == "times")
codes_location <-
  which(attributes(spikes_default_channel)$dimnames[[1]] == "codes")

## Find matlab's stimulus change log
stim_change_channel <-
  mat_import[[stringr::str_which(names(mat_import), "Ch3")[1]]]
## Each sweep should be 5 secs. We'll check that the median is 5
## If this results in an error, then the channel identity could be wrong, or
## there may have been an issue with sweep duration during the recording
## process
if(!median(round(diff(stim_change_channel[[5]][,1]),0)) == 5) {
  warning("File ", i,": stim change channel identity uncertain")
}

## Determine when the onset of motion occurred according to matlab
first_moving_mat <-
  stim_change_channel[[5]][,1][1]
## Find the first "moving" phase in the log file
first_moving_csv <-
  csv_data_sets[[i]] %>%
  filter(Trial == "moving") %>%
  select(Time) %>%
  slice(1) %>%
  as.numeric()
## Find the first "blank" phase in the log file
first_blank <-
  csv_data_sets[[i]] %>%
  filter(Trial == "blank") %>%
  select(Time) %>%
  slice(1) %>%
  as.numeric()
## Compute the difference between these two
first_mvbl_diff <- first_moving_csv - first_blank

## Check to see if the final row of the metadata is "moving" and truncate
## if not
## This can effectively be done by truncating after the final "moving" phase
max_moving_sweep <-
  max(which(csv_data_sets[[i]]$Trial == "moving"))
```

```
first_csv_tmp <-
  bind_rows(initial, csv_data_sets[[i]]) %>%
  ## Add 1 to max moving sweep since we tacked on "initial" in previous step
  ## Then slice to restrict any extraneous partial sweeps
  slice_head(n = (max_moving_sweep + 1)) %>%
  ## Add the first event time to "Time" and subtract first_mvbl_diff (~2 secs)
  ## What this does is shift the log csv's time stamping to match the matlab
  ## file's stim change channel's time stamping
  mutate(Time = Time + first_moving_mat - first_mvbl_diff - first_blank) %>%
  ## Make character version of Time for joining later
  ## This will be crucial for _join functions
  mutate(Time_char = as.character(round(Time,3)))

## Duplicate the initialization for ease of setting T0
inception <-
  initial %>%
  mutate(Time_char = as.character(round(Time,3)))
inception$Trial[1] <- "inception"

## Bind the initialization rows
first_csv <-
  bind_rows(inception, first_csv_tmp)
## Compute stimulus end times
first_csv$Stim_end <- c(first_csv$Time[-1], max(first_csv$Time) + 3)

## Get final time
final_time <- first_csv$Stim_end[nrow(first_csv)]

## Extract photodiode data
## First generate a time sequence to match to the photodiode trace
Time_vec <- seq(
  from = 0.0000,
  by = 1 / 25000,
  length.out = length(photod_default_channel[9][[1]][, 1])
)
## The key thing is to get a character version of time from this
Time_char_vec <- as.character(round(Time_vec, 3))

## Grab the photodiode data
photod_full <-
  tibble(Photod =
           photod_default_channel[9][[1]][, 1])
## Add numeric time
photod_full$Time <-
  seq(
```

```
    from = 0.0000,
    by = 1 / 25000,
    length.out = nrow(photod_full)
  )
options(scipen = 999)
photod_full <-
  photod_full %>%
  ## Add the character time
  add_column(Time_char = Time_char_vec) %>%
  ## Use the charcter time to define a group
  group_by(Time_char) %>%
  ## Then average the photodiode within
  summarise(Photod = mean(Photod)) %>%
  ungroup() %>%
  mutate(Time = round(as.numeric(Time_char), 3)) %>%
  arrange(Time) %>%
  filter(Time <= final_time)

## Extract all spike data
all_spike_dat <-
  tibble(
    Time =
      spikes_default_channel[times_location][[1]][, 1],
    code =
      spikes_default_channel[codes_location][[1]][, 1]) %>%
  ## Characterize time, for purposes of joining later
  mutate(Time_char = as.character(round(Time, 3)))

## How many distinct neurons are there?
cell_ids <- sort(unique(all_spike_dat$code))
n_cells <- 1:length(cell_ids)

if(length(n_cells) > 1) { ## if there's more than one distinct neuron
  all_spike_dat_tmp <-
    all_spike_dat %>%
    ## Group by identity of spiking neuron
    group_by(code) %>%
    ## Split into separate dfs, one per neuron
    group_split()

  ## Additional cells are labeled as "Spike_n"
  all_cells <- NULL
  for (j in n_cells) {
    #print(j)
```

```
    new_name = paste0("Spikes_", cell_ids[j])
    all_cells[[j]] <-
      all_spike_dat_tmp[[j]]

    ## Consolidate to 3 decimal places
    all_cells[[j]] <-
      all_cells[[j]] %>%
      group_by(Time_char) %>%
      summarise(code = mean(code)) %>%
      mutate(code = ceiling(code)) %>%
      ungroup() %>%
      mutate(Time = round(as.numeric(Time_char), 3)) %>%
      arrange(Time) %>%
      filter(Time <= final_time)

    names(all_cells[[j]])[match("code", names(all_cells[[j]]))] <-
      new_name
    ## Replace "j" with 1 to indicate presence/absence of spike rather than
    ## cell identity
    all_cells[[j]][new_name] <- 1

    ## If the identity is 1, replace "Spikes_1" with just "Spikes"
    if (new_name == "Spikes_1") {
      names(all_cells[[j]])[match(new_name, names(all_cells[[j]]))] <-
        "Spikes"
    }
  }

  ## Consolidate
  all_spike_dat <-
    all_cells %>%
    ## Tack on additional spike columns
    reduce(full_join, by = "Time_char") %>%
    arrange(Time_char) %>%
    ## Remove time.n columns but
    ## Do not remove Time_char
    select(-contains("Time.")) %>%
    ## Regenerate numeric time
    mutate(
      Time = as.numeric(Time_char)
    ) %>%
    select(Time, Time_char, Spikes, everything()) %>%
    filter(Time <= final_time)

} else { ## If there's only 1 neuron
```

```
    all_spike_dat <-
      all_spike_dat %>%
      group_by(Time_char) %>%
      summarise(code = mean(code)) %>%
      mutate(code = ceiling(code)) %>%
      ungroup() %>%
      rename(Spikes = code) %>%
      mutate(Time = round(as.numeric(Time_char), 3)) %>%
      arrange(Time) %>%
      filter(Time <= final_time) %>%
      select(Time, Time_char, everything())
  }

  options(scipen = 999)
  mat_data_sets[[i]] <-
    ## Generate a time sequence from 0 to final_time
    tibble(
      Time = seq(from = 0, to = final_time, by = 0.001)
    ) %>%
    ## Character-ize it
    mutate(Time_char = as.character(round(Time, 5))) %>%
    ## Join in the photodiode data
    left_join(photod_full, by = "Time_char") %>%
    select(-Time.y) %>%
    rename(Time = Time.x) %>%
    arrange(Time) %>%
    ## Join in the spike data
    left_join(all_spike_dat, by = "Time_char") %>%
    select(-Time.y) %>%
    rename(Time = Time.x) %>%
    arrange(Time) %>%
    filter(Time <= final_time) %>%
    ## Replace NAs with 0 in the Spike columns only
    mutate(
      across(starts_with("Spikes"), ~replace_na(.x, 0))
    )

  ## Merge the matlab data with the metadata
  joined_one_full <-
    mat_data_sets[[i]] %>%
    ## Join by the character version of time NOT the numerical!!
    full_join(first_csv, by = "Time_char") %>%
    ## Rename columns for clarity of reference
    rename(Time_mat = Time.x,
           Time_csv = Time.y) %>%
```

```
    ## Convert character time to numeric time
    mutate(Time = round(as.numeric(Time_char), 3)) %>%
    ## Carry metadata forward
    mutate(
      Trial = zoo::na.locf(Trial, type = "locf"),
      Spatial_Frequency = zoo::na.locf(Spatial_Frequency, type = "locf"),
      Cycles_Per_Pixel  = zoo::na.locf(Cycles_Per_Pixel, type = "locf"),
      Temporal_Frequency = zoo::na.locf(Temporal_Frequency, type = "locf"),
      Direction = zoo::na.locf(Direction, type = "locf"),
      Time_csv = zoo::na.locf(Time_csv, type = "locf"),
      Stim_end = zoo::na.locf(Stim_end, type = "locf")
    ) %>%
    ## Calculate velocity
    mutate(
      Speed = round(Temporal_Frequency/Spatial_Frequency, 0),
      Log2_Speed = log2(Speed)
    )

  ## Add info to metadata
  metadata_one_full <-
    first_csv %>%
    mutate(
      Speed = round(Temporal_Frequency/Spatial_Frequency, 0),
      Log2_Speed = log2(Speed),
      Stim_end_diff = c(0, diff(Stim_end))
    )

  ## Some quality control checks
  ## What are the stim time differences?
  stimtime_diffs <- round(metadata_one_full$Stim_end_diff)[-c(1:2)]
  ## How many total reps were recorded?
  stimtime_reps <- length(stimtime_diffs)/3
  ## What do we expect the overall structure to look like?
  stimtime_expectation <- rep(c(1,1,3), stimtime_reps)
  ## Does reality match our expectations?
  if (!all(stimtime_diffs == stimtime_expectation)) {
    ## If you get this, investigate the file further and determine what went
    ## wrong
    print("stimtime issue; investigate")
  }

  ## Sometimes the final sweep gets carried for an indefinite amount of time
  ## before the investigator manually shuts things off. The following block
  ## truncates accordingly
  mark_for_removal <-
```

```
    which(round(metadata_one_full$Stim_end_diff) %not_in% c(1, 3))
  if (any(mark_for_removal == 1 | mark_for_removal == 2)) {
    mark_for_removal <- mark_for_removal[mark_for_removal > 2]
  }
  if (length(mark_for_removal) > 0) {
    metadata_sets[[i]] <-
      metadata_one_full %>%
      filter(Time < metadata_one_full[mark_for_removal[1],]$Time)
    joined_data_sets[[i]] <-
      joined_one_full %>%
      filter(Time_mat < metadata_one_full[mark_for_removal[1],]$Time)
  } else {
    metadata_sets[[i]] <-
      metadata_one_full
    joined_data_sets[[i]] <-
      joined_one_full
  }

  ## Organize the metadata for export in the R environment
  meta_splits[[i]] <-
    metadata_sets[[i]] %>%
    ## Get rid of the non-sweep info
    filter(!Trial == "inception") %>%
    filter(!Trial == "initialization") %>%
    ## Group by trial
    #group_by(Spatial_Frequency, Temporal_Frequency, Direction) %>%
    ## Split by trial group
    group_split(Spatial_Frequency, Temporal_Frequency, Direction)

  data_splits[[i]] <-
    joined_data_sets[[i]] %>%
    ## Get rid of the non-sweep info
    filter(!Trial == "inception") %>%
    filter(!Trial == "initialization") %>%
    ## Group by trial
    group_by(Spatial_Frequency, Temporal_Frequency, Direction) %>%
    ## Split by trial group
    group_split()

  ## Do some cleanup so large objects don't linger in memory
  rm(
    first_csv, inception, initial, mat_import, first_csv_tmp,
    photod_default_channel, spikes_default_channel, photod_full,
    all_spike_dat, all_spike_dat_tmp, all_cells,
    metadata_one_full, joined_one_full, joined_data_sets,
```

```
    csv_data_sets, mat_data_sets
  )
  message("File ", i, ": ", csv_mat_filejoin[i,"basename"], " imported")
  gc()
}
```

```
## [1] 1
```

```
endtime <- Sys.time()
endtime - starttime ## Total elapsed time
```

```
## Time difference of 2.325644 mins
```

```
## Tidy up how R has been using RAM by running garbage collection
gc()
```

```
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   5567281  297.4   16192312  864.8   25300487  1351.2
## Vcells 204486851 1560.2  915533579 6985.0 1430521216 10914.1
```

```
## Name each data set according to the basename of the file
names(metadata_sets) <- csv_mat_filejoin$basename #base_names
names(meta_splits)   <- csv_mat_filejoin$basename #base_names
names(data_splits)   <- csv_mat_filejoin$basename #base_names

## Get organized lists of stimuli that were used
## This will ultimately be used for arranging data by stimuli in a sensible
## order
metadata_combos <- NULL
for (i in 1:length(metadata_sets)) {
  metadata_combos[[i]] <-
    metadata_sets[[i]] %>%
    ## Get unique stimulus parameters
    distinct(Spatial_Frequency, Temporal_Frequency, Speed, Direction) %>%
    arrange(Direction) %>%
    ## Sort by SF (smallest to largest)
    arrange(desc(Spatial_Frequency)) %>%
    mutate(
      ## Set a "plot_order" which provides a running order of stimuli
      plot_order = 1:length(meta_splits[[i]]),
      name = paste(Direction, "Deg,",  Speed, "Deg/s")
    )
}
names(metadata_combos) <- csv_mat_filejoin$basename #base_names
```

What we now have is the following:

- `metadata_sets`: contains a `list` of `tibble`s (one per imported file), each of which comprises the stimulus log
- `metadata_combos`: contains a `list` of `tibble`s (one per imported file), each of which comprises the stimulus log re-written in a preferred plotting order
- `meta_splits`: a `list` of `list`s. The first level of the `list` corresponds to the file identities. Within each file's `list`, there is an additional set of `list`s, each of which contains a `tibble` with the log info for a specific stimulus combination. Essentially the same as `metadata_sets` but split up on a per-stimulus basis.
- `data_splits`: another `list` of `list`s, arranged in the same hierarchical order as `meta_splits`. Each tibble herein contains the spike and photodiode data from the `matlab` file on a per-stimulus basis.

Note that since we are only using one example file, each of these lists should only be 1 element long (with the latter two having additional elements within the first element)

5.2 Organizing replicates (required) and binning (optional)

It is common to record several different sweeps of a stimulus to collect (somewhat) independent replicates of neural responses. The primary task of this section will be to use the lists from the previous section to gather & label replicates of the same stimulus.

A secondary task is to deal with binning, if desired (highly recommended). Depending on the goals of plotting and/or analyses, it may be wise to work with either unbinned spike data or to bin the data at a convenient and sensible interval. I generally choose to work at the following levels:

1. Unbinned
2. Bin size = 10 ms
3. Bin size = 100 ms

Important note: Rather than provide separate code for each of these 3 scenarios, I will provide one example. Bin size **must** be declared beforehand – please pay attention.

```
## Set bin size here
## Units are in ms (e.g. 10 = 10ms)
bin_size = 10 ## 10 or 100 or 1 (1 = "unbinned")

slice_size = NULL
slicemin = NULL
slicemax = NULL
condition = NULL
if (bin_size == 10){
  slice_size <- 501
  slicemin <- 202
  slicemax <- 498
  condition <- "_binsize10"
} else if (bin_size == 100){
  slice_size <- 51
  slicemin <- 21
  slicemax <- 49
  condition <- "_binsize100"
} else if (bin_size == 1){
  slice_size <- NULL
  slicemin <- NULL
  slicemax <- NULL
  condition <- "_unbinned"
} else {
  stop("bin_size is non-standard")
}


all_replicate_data_reorganized <-
  vector(mode = "list", length = length(meta_splits))
name_sets <-
  vector(mode = "list", length = length(meta_splits))
gc()
```

```
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   5567387  297.4   16192312  864.8   25300487  1351.2
## Vcells 204479268 1560.1  732426864 5588.0 1430521216 10914.1
```

```
starttime <- Sys.time()
for (i in 1:length(meta_splits)){

  ## i = file number
  print(i)

  ## We'll need to collect data at a per-stimulus level and on a per-replicate
```

```
## level within the per-stimulus level
## "j" will be used to designate a unique stimulus
## We'll first create an empty object in which to collect stimulus-specific
## data
replicate_data_reorganized <- NULL
## For each of j unique stimuli...
for (j in 1:length(meta_splits[[i]])) { # j = {direction,speed}
  ## Isolate the j-th data
  d <- data_splits[[i]][[j]]
  ## And the j-th log data
  m <- meta_splits[[i]][[j]] %>%
    group_by(Trial) %>%
    ## Label separate replicates
    mutate(Replicate = row_number())

  ## Extract a stimulus label to a name_set that will be used later
  name_sets[[i]][[j]] <-
    paste(m$Direction[1], "Deg,",  m$Speed[1], "Deg/s")

  ## Set up a temporary object to deal with per-replicate data
  replicates_ordered <- NULL
  ## "k" will be used to designate replicate number
  for (k in 1:max(m$Replicate)){
    tmp <-
      m %>%
      filter(Replicate == k)

    ## If you have a complete replicate (i.e., blank, stationary, moving)
    if (nrow(tmp) == 3 ) {
      ## Grab the specific data
      doot <-
        d %>%
        filter(Time >= min(tmp$Time)) %>%
        filter(Time <= max(tmp$Stim_end))
      ## Add bin information
      doot$bin <-
        rep(1:ceiling(nrow(doot)/bin_size), each = bin_size)[1:nrow(doot)]

      if (bin_size == 1) {
        ## IF YOU ARE NOT BINNING, RUN THIS:
        replicates_ordered[[k]] <-
          doot %>%
          mutate(
            ## Construct a standardized time within the sweep
            Time_stand = Time_mat - min(Time_mat),
```

```
            ## When does the sweep begin
            Time_begin = min(Time_mat),
            ## When does the sweep end
            Time_end = max(Time_mat),
            ## Delineate the end of the blank phase
            Blank_end = tmp$Stim_end[1] - min(Time_mat),
            ## Delineate the end of the stationary phase
            Static_end = tmp$Stim_end[2] - min(Time_mat),
            ## Label the replicate number
            Replicate = k
          ) %>%
          ## Bring stim info to first few columns
          select(Speed, Spatial_Frequency, Temporal_Frequency, Direction,
                 everything()) %>%
          ## Just in case there some hang over
          filter(Time_stand >= 0)
      } else { ## IF YOU ARE BINNING, RUN THIS:

        ## First grab time and meta info
        time_and_meta <-
          doot %>%
          ## WITHIN EACH BIN:
          group_by(bin) %>%
          summarise(
            ## Label the trial
            Trial = first(Trial),
            ## Midpoint of bin
            Time_bin_mid = mean(Time_mat),
            ## Bin beginning
            Time_bin_begin = min(Time_mat),
            ## Bin end
            Time_bin_end = max(Time_mat),
            ## Spike rate = sum of spikes divided by elapsed time
            Spike_rate = sum(Spikes) / (max(Time_mat) - min(Time_mat)),
            Photod_mean = mean(Photod)
          )

        ## Now deal with Spike_N columns
        hold_spike_n <-
          doot %>%
          select(starts_with("Spikes_")) %>%
          add_column(bin = doot$bin) %>%
          add_column(Time_mat = doot$Time_mat) %>%
          group_by(bin) %>%
          summarise(across(starts_with("Spikes_"),
```

```
                              ~ sum(.x) / (max(Time_mat) - min(Time_mat))))

          ## Put them together
          replicates_ordered[[k]] <-
            time_and_meta %>%
            left_join(hold_spike_n, by = "bin") %>%
            mutate(
              ## Add in metadata (following same definitions above)
              Time_stand = Time_bin_mid - min(Time_bin_mid),
              Blank_end = tmp$Stim_end[1] - min(Time_bin_mid),
              Static_end = tmp$Stim_end[2] - min(Time_bin_mid),
              Spatial_Frequency = m$Spatial_Frequency[1],
              Temporal_Frequency = m$Temporal_Frequency[1],
              Speed = m$Speed[1],
              Direction = m$Direction[1],
              Replicate = k
            ) %>%
            ## Bring stim info to first few columns
            select(Speed, Spatial_Frequency, Temporal_Frequency, Direction,
                   everything()) %>%
            ## Just in case there some hang over
            filter(Time_stand >= 0) %>%
            filter(bin < slice_size + 1)

          rm(time_and_meta, hold_spike_n)
        }
      }
      }

    ## Now insert it within the collector of per-stimulus data
    replicate_data_reorganized[[j]] <-
      replicates_ordered %>%
      bind_rows()

    ## Now insert it within the overall data collector
    all_replicate_data_reorganized[[i]][[j]] <-
      replicate_data_reorganized[[j]]

    ## Toss out temporary objects and clean up
    rm(replicates_ordered, d, m, tmp)
    gc()
  }
}
```

```
## [1] 1
```

```
endtime <- Sys.time()
endtime - starttime
```

```
## Time difference of 2.058902 mins
```

```
gc()
```

```
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   5572323  297.6   16192312  864.8   25300487  1351.2
## Vcells 208828836 1593.3  585941492 4470.4 1430521216 10914.1
```

```
for (i in 1:length(all_replicate_data_reorganized)) {
  for (j in 1:length(all_replicate_data_reorganized[[i]])) {
    names(all_replicate_data_reorganized[[i]])[[j]] <- name_sets[[i]][[j]]
  }
}
names(all_replicate_data_reorganized) <- csv_mat_filejoin$basename #base_names
```

At the end of this process, here is how `all_replicate_data_reorganized[[1]]` should look:

```
## How long is this list?
length(all_replicate_data_reorganized[[1]])
```

```
## [1] 80
```

```
## This object contains 48 individual tibbles. Here's an example of one
## pulled at random
all_replicate_data_reorganized[[1]][sample(1:48, size = 1)]
```

```
## $`180 Deg, 407 Deg/s`
## # A tibble: 3,507 x 16
##    Speed Spatial_F~1 Tempo~2 Direc~3   bin Trial Time_~4 Time_~5 Time_~6 Spike~7
##    <dbl>       <dbl>   <dbl>   <dbl> <int> <chr>   <dbl>   <dbl>   <dbl>   <dbl>
##  1   407      0.0248    10.1     180     1 blank    160.    160.    160.       0
##  2   407      0.0248    10.1     180     2 blank    160.    160.    160.       0
##  3   407      0.0248    10.1     180     3 blank    160.    160.    160.       0
##  4   407      0.0248    10.1     180     4 blank    160.    160.    160.       0
##  5   407      0.0248    10.1     180     5 blank    160.    160.    160.       0
##  6   407      0.0248    10.1     180     6 blank    160.    160.    160.     111.
##  7   407      0.0248    10.1     180     7 blank    160.    160.    160.     222.
##  8   407      0.0248    10.1     180     8 blank    160.    160.    160.       0
```

```
##  9   407      0.0248    10.1     180     9 blank    160.    160.    160.      0
## 10   407      0.0248    10.1     180    10 blank    160.    160.    160.    111.
## # ... with 3,497 more rows, 6 more variables: Photod_mean <dbl>,
## #   Spikes_0 <dbl>, Time_stand <dbl>, Blank_end <dbl>, Static_end <dbl>,
## #   Replicate <int>, and abbreviated variable names 1: Spatial_Frequency,
## #   2: Temporal_Frequency, 3: Direction, 4: Time_bin_mid, 5: Time_bin_begin,
## #   6: Time_bin_end, 7: Spike_rate
```

```
## To consolidate this, you can use bind_rows()
all_replicate_data_reorganized[[1]] %>% bind_rows
```

```
## # A tibble: 284,067 x 16
##    Speed Spatial_F~1 Tempo~2 Direc~3   bin Trial Time_~4 Time_~5 Time_~6 Spike~7
##    <dbl>       <dbl>   <dbl>   <dbl> <int> <chr>   <dbl>   <dbl>   <dbl>   <dbl>
##  1  1024      0.0156      16       0     1 blank    181.    181.    181.      0
##  2  1024      0.0156      16       0     2 blank    181.    181.    181.      0
##  3  1024      0.0156      16       0     3 blank    181.    181.    181.      0
##  4  1024      0.0156      16       0     4 blank    181.    181.    181.      0
##  5  1024      0.0156      16       0     5 blank    181.    181.    181.      0
##  6  1024      0.0156      16       0     6 blank    181.    181.    181.      0
##  7  1024      0.0156      16       0     7 blank    181.    181.    181.      0
##  8  1024      0.0156      16       0     8 blank    181.    181.    181.    222.
##  9  1024      0.0156      16       0     9 blank    181.    181.    181.    222.
## 10  1024      0.0156      16       0    10 blank    181.    181.    181.    111.
## # ... with 284,057 more rows, 6 more variables: Photod_mean <dbl>,
## #   Spikes_0 <dbl>, Time_stand <dbl>, Blank_end <dbl>, Static_end <dbl>,
## #   Replicate <int>, and abbreviated variable names 1: Spatial_Frequency,
## #   2: Temporal_Frequency, 3: Direction, 4: Time_bin_mid, 5: Time_bin_begin,
## #   6: Time_bin_end, 7: Spike_rate
```

Bear in mind that I am specifically describing `all_replicate_data_reorganized[[1]]`, NOT `all_replicate_data_reorganized`. This is because `all_replicate_data_reorganized` will itself be a `list` that is as long as the number of distinct data files you are feeding into all of the above. Since there is only 1 example file, `all_replicate_data_reorganized` is a `list` that is only 1 entry long at its top level, and within that `list` is set of 48 `lists` (one per distinct stimulus), each of which contains a stim-specific `tibble` of data.

5.3 Data export

It is highly recommended that you export `all_replicate_data_reorganized`. I generally export `all_replicate_data_reorganized` as a separate `csv` file for each of the following conditions:

1. Unbinned
2. Bin size = 10 ms
3. Bin size = 100 ms

This section will provide code to export each of the above, assuming you used those bin sizes in the previous section. Please be sure to change file naming conventions and other parameters in the event you chose a different `bin_size` in the previous section.

```
## Declare export destination
export_path <- "./data/"
## The "condition" will be appended to the file name.

## Export each tibble within all_replicate_data_reorganized
for (i in 1:length(all_replicate_data_reorganized)) {
  print(i)
  dat <-
    all_replicate_data_reorganized[[i]] %>%
    bind_rows()
  write_csv(
    dat,
    file =
      paste0(
        export_path,
        names(all_replicate_data_reorganized)[i],
        condition,
        ".csv"
      )
  )
  rm(dat)
}
```

Re-run the above chunk of code per condition you seek to export. For me, this process creates 3 files: `2023-02-16_001_unbinned.csv`, `2023-02-16_001_binsize10.csv`, and `2023-02-16_001_binsize100.csv`.

Chapter 6

Raster and mean spike rate plots

6.1 Data sets

This section will rely on some of the unbinned and binned data we exported in final steps of the previous section. We'll start by loading in information

```
## File paths and basenames of _unbinned.csv files
unbinned_filelist <-
  list.files("./data/", pattern = "_unbinned.csv",
             full.names = TRUE)
unbinned_basenames <-
  unbinned_filelist %>%
  str_remove("./data/") %>%
  str_remove("_unbinned.csv")

## File paths and basenames of _binsize10.csv files
bin10_filelist <-
  list.files("./data/", pattern = "_binsize10.csv",
             full.names = TRUE)
bin10_basenames <-
  bin10_filelist %>%
  str_remove("./data/") %>%
  str_remove("_binsize10.csv")

## File paths and basenames of _binsize100.csv files
bin100_filelist <-
  list.files("./data/", pattern = "_binsize100.csv",
```

```
             full.names = TRUE)
bin100_basenames <-
  bin100_filelist %>%
  str_remove("./data/") %>%
  str_remove("_binsize100.csv")
```

You should have something similar to the following:

```
unbinned_filelist; unbinned_basenames; bin10_filelist; bin10_basenames; bin100_filelis
```

```
## [1] "./data/2023-02-16_001_unbinned.csv"
```

```
## [1] "2023-02-16_001"
```

```
## [1] "./data/2023-02-16_001_binsize10.csv"
```

```
## [1] "2023-02-16_001"
```

```
## [1] "./data/2023-02-16_001_binsize100.csv"
```

```
## [1] "2023-02-16_001"
```

6.2 Raster plot

Here is an example of a raster plot, using the wrangled data generated in the previous section. This type of plot shows the timing of spike events within each replicate sweep, and for our purposes, we'll produce a view of this for each of the various stimulus conditions

It is important to note that we will need unbinned data for this. This is because a raster plot shows discrete spiking events through the course of a time sweep.

Here, we will use `ggplot` to create a plot with:

1. Data sub-plotted by stimulus (i.e., Speed and Direction)
2. Standardized sweep time on the x-axis, with delineation among blank, stationary, and moving phases using red, yellow, and green undershading. The y-axis will indicate replicate number
3. A black tick each time a spike is observed. Absence of black tick = no observed spike

```
## For each unbinned file, generate a raster plot
rasterplots <- NULL
for (i in 1:length(unbinned_filelist)) {
  ## Read in the data
  unbinned_data <-
    read_csv(unbinned_filelist[i]) %>%
    as_tibble()

  ## determine the max number of replicates
  max_reps <- max(unbinned_data$Replicate)

  ## Generate the code for the ggplot and save it as rasterplots[[i]]
  rasterplots[[i]] <-
    unbinned_data %>%
    ## Remove any rows where spiking does not occur in the Spikes column
    filter(Spikes == 1) %>%
    ## Convert Trial and Speed into factors and specify their level ordering
    ## This will make it easier to get the subplots in the order we want them
    mutate(
      Trial = factor(Trial,
                     levels = c("blank", "stationary", "moving")),
      Speed = factor(Speed,
                     levels = c(1024, 644, 407, 256, 128, 64, 32, 16, 8, 4))) %>%
    ggplot(aes(x = Time_stand, y = Replicate)) +
    ## The next three blocks will undershade each subplot according to stimulus
    ## phase (i.e., blank, stationary, moving)
    annotate("rect",
             xmin = 0, xmax = first(unbinned_data$Blank_end),
             ymin = 0.5, ymax = max_reps + 0.5,
             alpha = 0.075, color = NA, fill = "red") +
    annotate("rect",
             xmin = first(unbinned_data$Blank_end),
             xmax = first(unbinned_data$Static_end),
             ymin = 0.5, ymax = max_reps + 0.5,
             alpha = 0.075, color = NA, fill = "darkgoldenrod1") +
    annotate("rect",
             xmin = first(unbinned_data$Static_end), xmax = 5,
             ymin = 0.5, ymax = max_reps + 0.5,
             alpha = 0.075, color = NA, fill = "forestgreen") +
    ## Up to 10 replicates were used, so we will force the y-axis to go to 10
    scale_y_continuous(
      limits = c(0.5, max_reps + 0.5),
      expand = c(0, 0),
      breaks = c(max_reps/2, max_reps)
    ) +
```