Fundamental plots for electrophysiological data

Vikram B. Baliga

2023-02-28

Contents

Chapter 1

About

This site is a work in progress. Ultimately, it will provide a walkthrough on how to produce fundamental plots from electrophysiological data. The content was initialized using a bookdown 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://creativecommons.org/licenses/by/4.0/
[2] 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",
              "readxl",
              "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         readxl_1.4.2       lubridate_1.9.2    forcats_1.0.0
## [21] stringr_1.5.0      dplyr_1.1.0        purrr_1.0.1        readr_2.1.4
## [25] tidyr_1.3.0        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     cellranger_1.1.0   yaml_2.3.7
##  [5] pillar_1.8.1       backports_1.4.1    lattice_0.20-45    glue_1.6.2
##  [9] digest_0.6.31      polyclip_1.10-4    colorspace_2.1-0   htmltools_0.5.4
## [13] Matrix_1.5-1       R.oo_1.25.0        pkgconfig_2.0.3    labelled_2.10.0
## [17] broom_1.0.3        haven_2.5.1        bookdown_0.32      xtable_1.8-4
## [21] mvtnorm_1.1-3      scales_1.2.1       tweenr_2.0.2       ggforce_0.4.1
## [25] tzdb_0.3.0         timechange_0.2.0   emmeans_1.8.4-1    farver_2.1.1
## [29] generics_0.1.3     ellipsis_0.3.2     withr_2.5.0        geepack_1.3.9
```

```
## [33] cli_3.6.0          magrittr_2.0.3     estimability_1.4.1 evaluate_0.20
## [37] R.methodsS3_1.8.2  fansi_1.0.4        MASS_7.3-58.2      geeM_0.10.1
## [41] tools_4.2.2        hms_1.1.2          lifecycle_1.0.3    munsell_0.5.0
## [45] compiler_4.2.2     rlang_1.0.6        ggridges_0.5.4     grid_4.2.2
## [49] rstudioapi_0.14    mosaicCore_0.9.2.1 rmarkdown_2.20     boot_1.3-28
## [53] gtable_0.3.1       R6_2.5.1           knitr_1.42         fastmap_1.1.1
## [57] utf8_1.2.3         ggformula_0.10.2   stringi_1.7.12     Rcpp_1.0.10
## [61] vctrs_0.5.2        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 (written by Tony Lapsanksy), and 2) Neuralynx (written by Eric Press)

3.1 Spike2

Written by Tony Lapsansky, February 24, 2023

1. Save the Spike2 file
    1. Use the name structure `YEARMODA_sequence_investigator`
    2. Save data in the corresponding directory `"C:\InvestigatorName\ephys\YEAR-MO-DA"`
2. Open `Spike2` and open the file
3. Apply a digital high pass filter if needed. Note that if the data were collected with the high pass filter set at greater than 100 Hz (no LFP signal) then skip to step 4.
    1. Right click on channel and select `FIR Digital Filters…` (see Spike 2 help → index `Digital Filter` for explanation)
    2. Under the pull down menu for `Filter` change from `Example low pass filter` to `Example high pass filter`
    3. Select the `Show Details` button in the bottom right
    4. Adjust blue slider to shift the colour dots above the slider from red to yellow to green, but use the minimum level to achieve green. Fine adjustments can be made just under the slider.
    5. Hit `Apply`
    6. Set `Destination` to the next available channel (often channel 4)
    7. Click Okay

8. Close the filtering window. You are given the option to save the filter. Do not do this. It is important to set the filter each time. (?)

4. Set thresholds for spikes
    1. Right click on the filtered channel and select `New WaveMark`
    2. Clear previous templates if any are present. To do so, select the trash can icon within each template.
    3. Locate the dashed vertical line, which can be found at time 0 in the main window. This line indicates your cursor position.
    4. Move the dashed line through the trace to observe potential spike as determined by the default upper and lower thresholds.
    5. Right click the upper bound marker (the upper horizontal dashed line in the `WaveMark` window) and select `move away`
    6. Identify spikes based on the lower bound. It is usually helpful to zoom in on the x-axis (time) to do this. Set the lower bound so that obvious spikes are included and ambiguous spikes are excluded.
5. Choose template setting
    1. Move the cursor to a typical spike. The upper window is a base template. Click and hold on the upper trace and drag it to the first available template window.
    2. 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.
    3. For the line `Maximum amplitude change for a match` enter `20`. This will allow a spike that fits a template to vary in maximum amplitude by up to 20%.
    4. For the line `Remove the DC offset before template matching`, confirm that the box is checked.
    5. Nothing else should need to be changed. Click `OK`.
6. Spike sorting
    1. Back in the `WaveMark` window, make sure that the box `Circular replay` is unchecked, and that the box `Make Templates` is checked.
    2. Ensure that the vertical cursor on the main window is at time zero (or the first spike).
    3. Hit the play button , which is called "run forward". This will take several minutes.
7. Use PCA to delete and merge spike templates
    1. Select `New Channel` on the `WaveMark` window to place the spike data in the next available channel (typically, Channel 5)
    2. Close the `WaveMark` window.
    3. Right click on the sorted channel and select `Edit WaveMark`
    4. Within the `WaveMark` window, go the pull down menu `Analyse` and select `Principal components`. Select `OK`. This opens a window of all spikes colored by template.

5. Rotate around all three axes to determine if there is one, two, or more clusters.
6. Identify templates that should be deleted and those that should be merged. Delete templates that sparse and peripheral.
7. Delete the template(s) in the `WaveMark` window by selecting that template's trash can icon.
8. Merge templates by dragging them into the same window
9. Hit the `reclassify` button in the `WaveMark` window.

8. Export the spike-sorted data

    1. `File → Export As`
    2. Select `.mat` (Matlab data)
    3. Use the same filename and location but with the .mat extension.
    4. Hit `Save`
    5. Select `Add` for `All Channels`
    6. Click `Export`
    7. Click `OK` (this will take several minutes)

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.

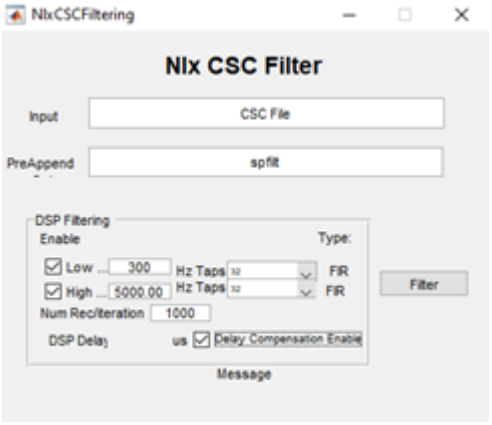


Figure 3.1: Nix csc filter

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 putative neurons are accounted for a minimal noise is detected.**

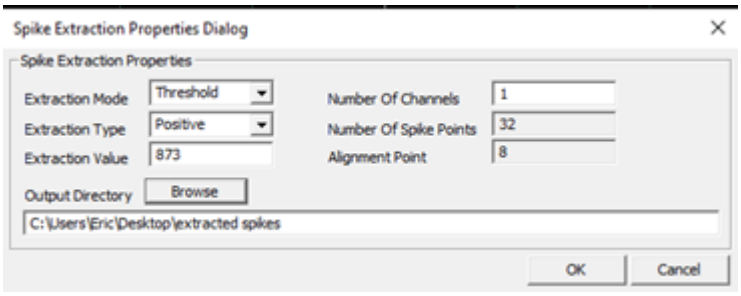


Figure 3.2: Spike extraction properties

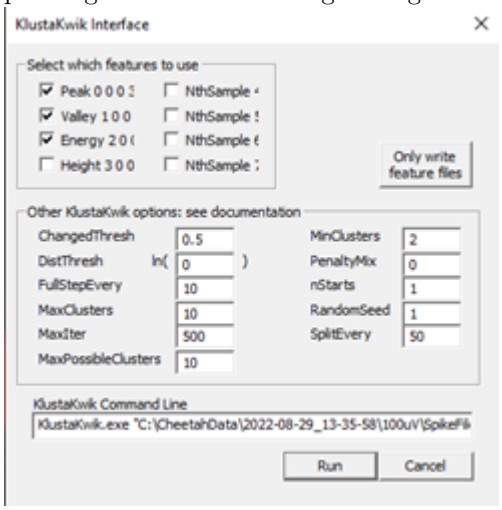
5. Spike sorting:

    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

Raw data and spike sorted traces

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

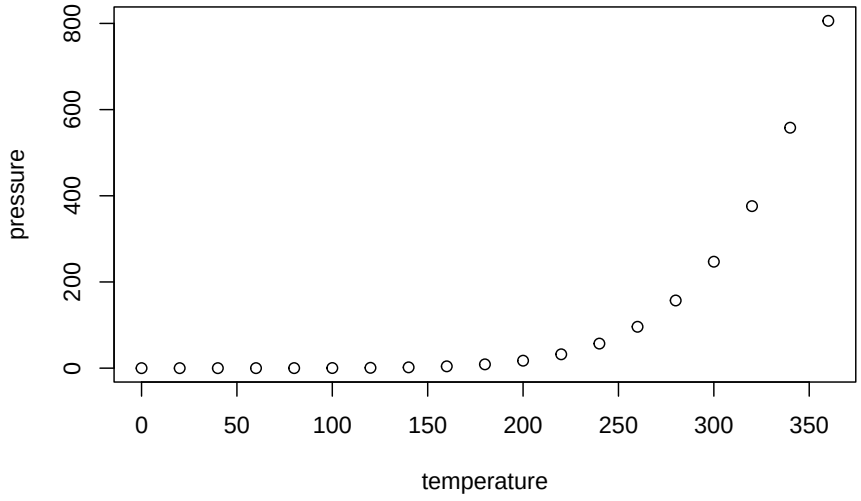
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

4.1 Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

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., `04132022_009m.mat` and `04132022_009m` have the same basename, which is `04132022_009m`).

```
## 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/04132022_009m.csv 04132022_009m 04132022_009 ./data/04132022_0~ 041320~
## # ... with abbreviated variable name 1: basedate.y
```

```
## and:
base_names
```

```
## [1] "04132022_009m"
```

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

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

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2437148 130.2    4173244 222.9  4173244 222.9
## Vcells 4211242  32.2   10146329  77.5  8020311  61.2
```

```
## 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

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`
  )
## Set up a `SF_cpd` column that tranlates SFs to cycles per degree
csv_data_sets[[i]]$SF_cpd[csv_data_sets[[i]]$Spatial_Frequency == 0.000668] <-
  2^-6
csv_data_sets[[i]]$SF_cpd[csv_data_sets[[i]]$Spatial_Frequency == 0.001336] <-
  2^-5
csv_data_sets[[i]]$SF_cpd[csv_data_sets[[i]]$Spatial_Frequency == 0.00267]  <-
  2^-4
csv_data_sets[[i]]$SF_cpd[csv_data_sets[[i]]$Spatial_Frequency == 0.0053]   <-
  2^-3
csv_data_sets[[i]]$SF_cpd[csv_data_sets[[i]]$Spatial_Frequency == 0.0106]   <-
  2^-2
csv_data_sets[[i]]$SF_cpd[csv_data_sets[[i]]$Spatial_Frequency == 0.0212]   <-
  2^-1

## 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]]$Spatial_Frequency[1],
  SF_cpd = csv_data_sets[[i]]$SF_cpd[1],
  Temporal_Frequency = csv_data_sets[[i]]$Temporal_Frequency[1],
  Direction  = csv_data_sets[[i]]$Direction[1],
  Time = 0.000
)

## Determine when the onset of motion occurred according to matlab
## NOTE: IF THIS INFO IS NOT IN CHANNEL 3, PLEASE CHANGE ACCCORDINGLY
first_moving_mat <-
```

```
  mat_import[[stringr::str_which(names(mat_import), "Ch3")[1]]][[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)
  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)]

## 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")

## Extract photodiode data
options(scipen = 999)
photod_full <-
  tibble(
    Photod =
      photod_default_channel[9][[1]][, 1])
photod_full$Time <-
  seq(
    from = 0,
    by = 1 / 25000,
    length.out = nrow(photod_full)
  )
photod_full <-
  photod_full %>%
  mutate(Time_char = as.character(round(Time, 5)))

## Extract all spike data
```

```
all_spike_dat <-
  tibble(
    Time =
      spikes_default_channel[times_location][[1]][, 1],
    code =
      spikes_default_channel[codes_location][[1]][, 1]) %>%
  ## Get rid of empty rows
  filter(code > 0) %>%
  ## Characterize time, for purposes of joining later
  mutate(Time_char = as.character(round(Time, 5))) #%>%
  #mutate(code = as.character(code))

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

if(length(n_cells) > 1) {
  all_spike_dat_tmp <-
    all_spike_dat %>%
    ## Group by identity of spiking neuron
    group_by(code) %>%
    ## Split into separate dfs, one per neuron
    #split(factor(all_spike_dat$code))
    group_split()

  ## Keep the name of the first cell's spike column as simply "Spikes"
  first_cell <-
    all_spike_dat_tmp[[1]] %>%
    rename(Spikes = code)

  ## Additional cells are labeled as "Spike_n"
  ad_cells <- n_cells[n_cells > 1]
  all_cells <- NULL
  for (j in ad_cells) {
    #print(i)
    new_name = paste0("Spikes_", j)
    all_cells[[j]] <-
      all_spike_dat_tmp[[j]]
    names(all_cells[[j]])[match("code", names(all_cells[[j]]))] <-
      new_name
  }

  ## Add first cell back in
  all_cells[[1]] <- first_cell

  ## Consolidate
```

```
    all_spike_dat <-
      all_cells %>%
      ## Tack on additional spike columns
      reduce(left_join, by = "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, everything())

  } else {
    all_spike_dat <-
      all_spike_dat %>%
      rename(Spikes = code) %>%
      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) %>%
    ## Join in the spike data
    left_join(all_spike_dat, by = "Time_char") %>%
    select(-Time.y) %>%
    rename(Time = Time.x) %>%
    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 = as.numeric(Time_char)) %>%
  ## Carry metadata forward
  mutate(
    Trial = zoo::na.locf(Trial, type = "locf"),
    Spatial_Frequency = zoo::na.locf(Spatial_Frequency, type = "locf"),
    SF_cpd = zoo::na.locf(SF_cpd, 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 = Temporal_Frequency/SF_cpd,
    Log2_Speed = log2(Speed)
  )

## Add info to metadata
metadata_one_full <-
  first_csv %>%
  mutate(
    Speed = Temporal_Frequency/SF_cpd,
    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, first_cell, 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 2.194635 mins
```

```
## Tidy up
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  4074544 217.7   70130373 3745.4  40623809 2169.6
## Vcells 44126215 336.7  460296310 3511.8 719130736 5486.6
```

```
## 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(SF_cpd, Temporal_Frequency, Speed, Direction) %>%
    arrange(Direction) %>%
    ## Sort by SF_cpd (smallest to largest)
    arrange(desc(SF_cpd)) %>%
    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
if (bin_size == 10){
  slice_size <- 501
  slicemin <- 202
  slicemax <- 498
} else if (bin_size == 100){
  slice_size <- 51
  slicemin <- 21
  slicemax <- 49
} else if (bin_size == 1){
  slice_size <- NULL
  slicemin <- NULL
  slicemax <- NULL
} 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  4077439 217.8   56104299 2996.3  40623809 2169.6
## Vcells 44125380 336.7  368237048 2809.5 719130736 5486.6
```

```
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, SF_cpd, Temporal_Frequency, Direction,
             everything()) %>%
      ## Just in case there some hang over
      filter(Time_stand >= 0)
  } else { ## IF YOU ARE BINNING, RUN THIS:
  replicates_ordered[[k]] <-
    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)
    ) %>%
    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),
      SF_cpd = m$SF_cpd[1],
      Temporal_Frequency = m$Temporal_Frequency[1],
      Speed = m$Speed[1],
      Direction = m$Direction[1],
      Replicate = k
    ) %>%
    ## Bring stim info to first few columns
```

```
            select(Speed, SF_cpd, Temporal_Frequency, Direction,
                   everything()) %>%
            ## Just in case there some hang over
            filter(Time_stand >= 0) %>%
            filter(bin < slice_size + 1)
        }
      }
      }

    ## 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 object and clean up
    rm(replicates_ordered)
    gc()
  }
}
```

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

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

```
## Time difference of 1.147151 mins
```

```
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  4083807 218.1   11765926  628.4  40623809 2169.6
## Vcells 46315999 353.4  150829896 1150.8 719130736 5486.6
```

```
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
```

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. Choose whichever one
## corresponds to the bin_size you used above
condition <-  "_binsize10"
#condition <-  "_binsize100"
#condition <-  "_unbinned"

## 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

Chapter 6

Raster and mean spike rate plots

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

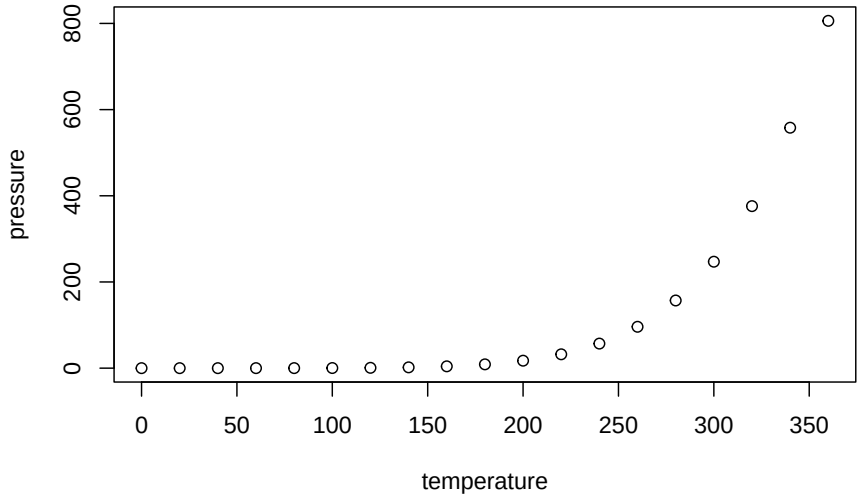
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

6.1 Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Chapter 7

Spatiotemporal tuning

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

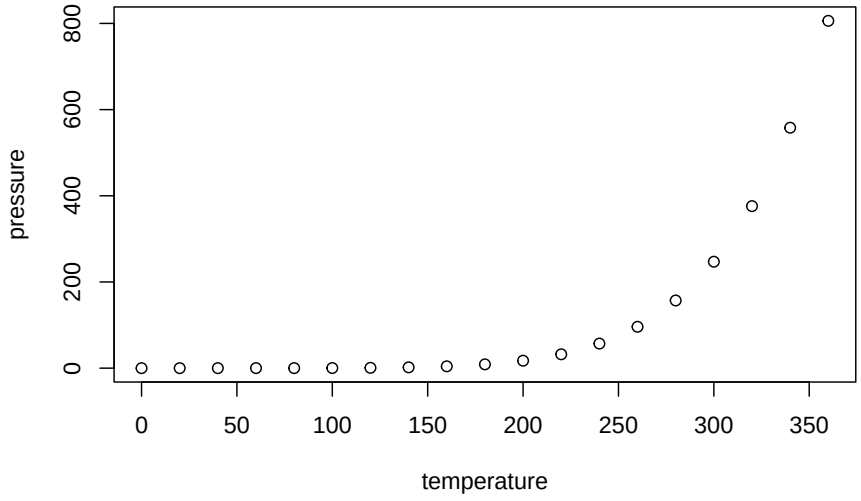
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

7.1 Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Chapter 8

Histological verification

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

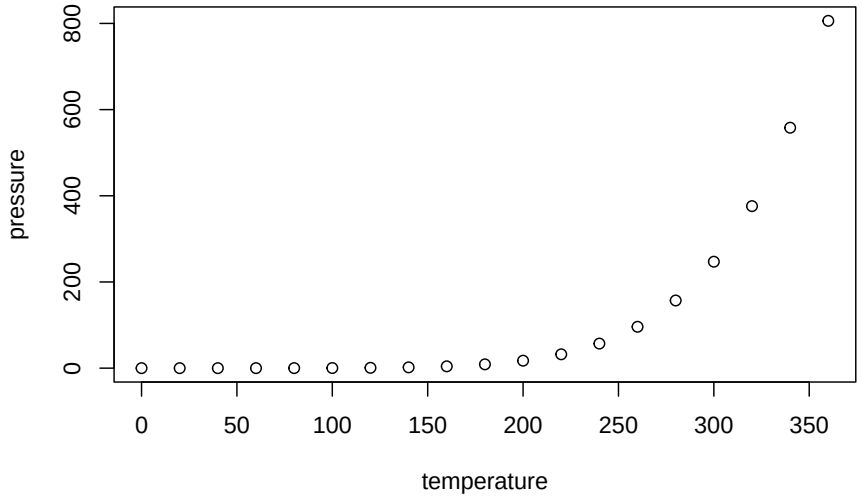
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

8.1 Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.