Computational Cognitive Science 2024-2025

Tutorial 2: Parameter Estimation

Files in this tutorial's materials:

```
manifest.txt
data.json
load_data.r
tutorial2.Rmd
```

General Information

The exercises in this tutorial are designed so that you can do them by directly editing the R-markdown (.Rmd) file. This file contains some code you need to complete so it will not "knit" to an HTML file or PDF until it is complete; you should run the cells individually while working on it.

```
2+3
```

```
## [1] 5
```

You can run cells in R-markdown using the little green "play" arrow in the cell, the "run" button at the top of the editor window, or keyboard shortcuts visible from the run button's menu. Running a cell will show its output directly below the cell. Alternatively, you can generate pdf or html documents by executing `Knit` (in the toolbar in R-studio), which is a great way to communicate your results.

You can also copy your code into a script file and execute those or simply perform one-off calculations in the R console. In this case, be sure you're running R commands from the **console** window, rather than the **terminal** window.

Package: GGplot2 for Data Visualization

This tutorial will use ggplot2, a powerful library for data visualization. It doesn't come with base R, so we'll have to install it – although if you're using a laboratory computer it may already be there.

This command will load the library if it's available:

```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
# Some additional libraries
require(stringr)
```

```
## Loading required package: stringr
```

If that command gave a warning message, you need to install the package. To do that, run the command below, then try `require(ggplot2)` again.

```
install.packages("ggplot2", dependencies = TRUE)
```

GGplot2 follows a specific approach to data visualization known as the 'grammar of graphics.' We will use

examples from the library here without going in-depth into the overall principles, but you may find it helpful to refer to the package's documentation.

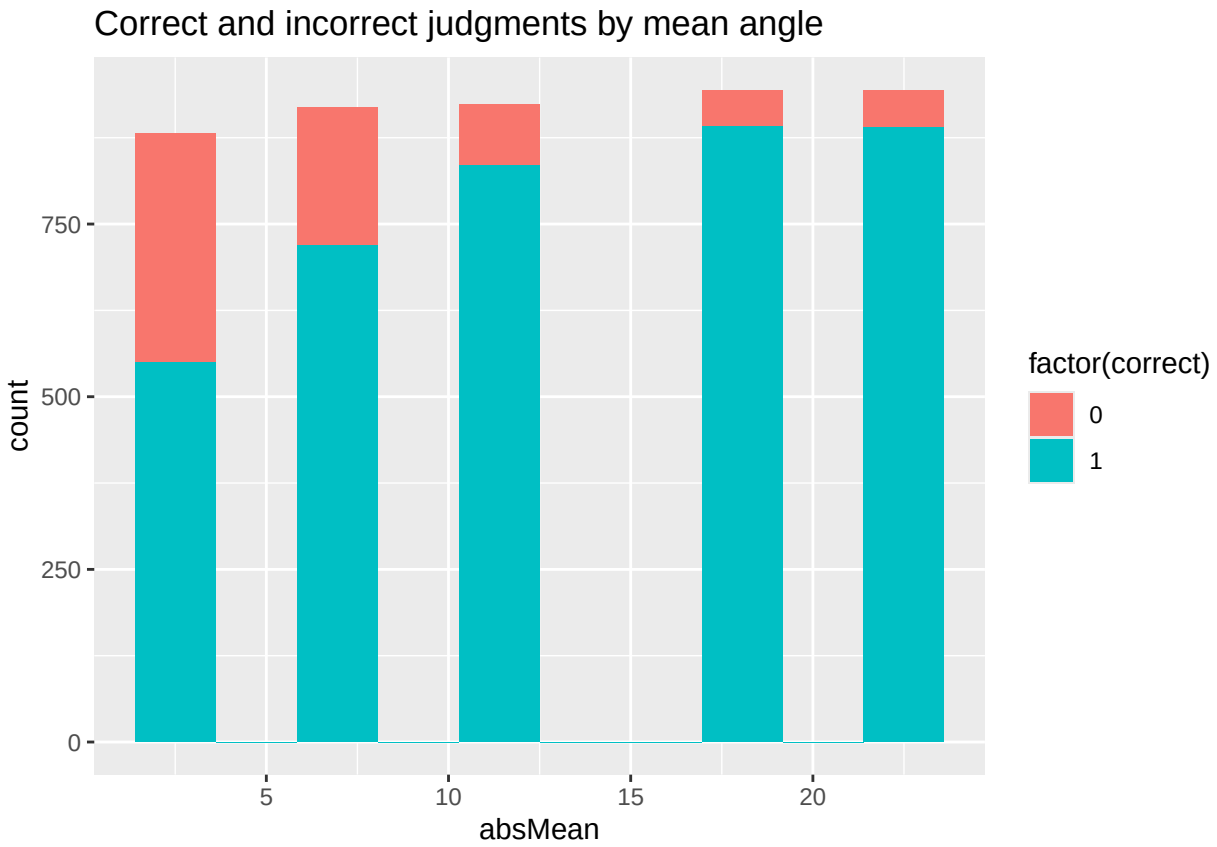
Fitting Reaction Times

We will continue working with our replication of the Smith and Vickers (1988) data. To simplify our task we will only look at data for the mean angle of 22.5 degrees. We will also ignore whether the participants' responses were correct or not. This is a debatable decision, especially if one wants to provide a general model of the cognitive process underlying the task. However, it should be less of an issue for this particular condition, as most responses for this angle should be correct.

```
source('load_data.r')
data <- loadData("data.json")
```

To verify our hunch that most of the 22.5 angle judgments were correct, we can visualize the distribution of correct and incorrect judgments for each angle. The code below produces a histogram of judgments and steps through some data visualizations with ggplot2 along the way.

```
ggplot(data) + # the core 'ggplot' function takes the data as input
  aes( # 'aesthetics' mapping: which aspects of the data will we plot?
    x=absMean, # the values of the 'absMean' column will be plotted on the x axis
    fill=factor(correct)) + # the 'correct' column will set the color fill of the bars
      # ^ the 'factor' function converts from numbers (0,1) to strings ("0","1")
  geom_histogram( # 'geometry' layer: what kind of plot is this?
    bins=10) + # specify a sensible number of bins for this histogram
  ggtitle("Correct and incorrect judgments by mean angle") # add title
```



We now store all rows for which the mean angle was 22.5 degrees (both positive and negative angles), and simplify our task by removing extremely long reaction times (>10 seconds). This is the data set we will be

working with in this tutorial.

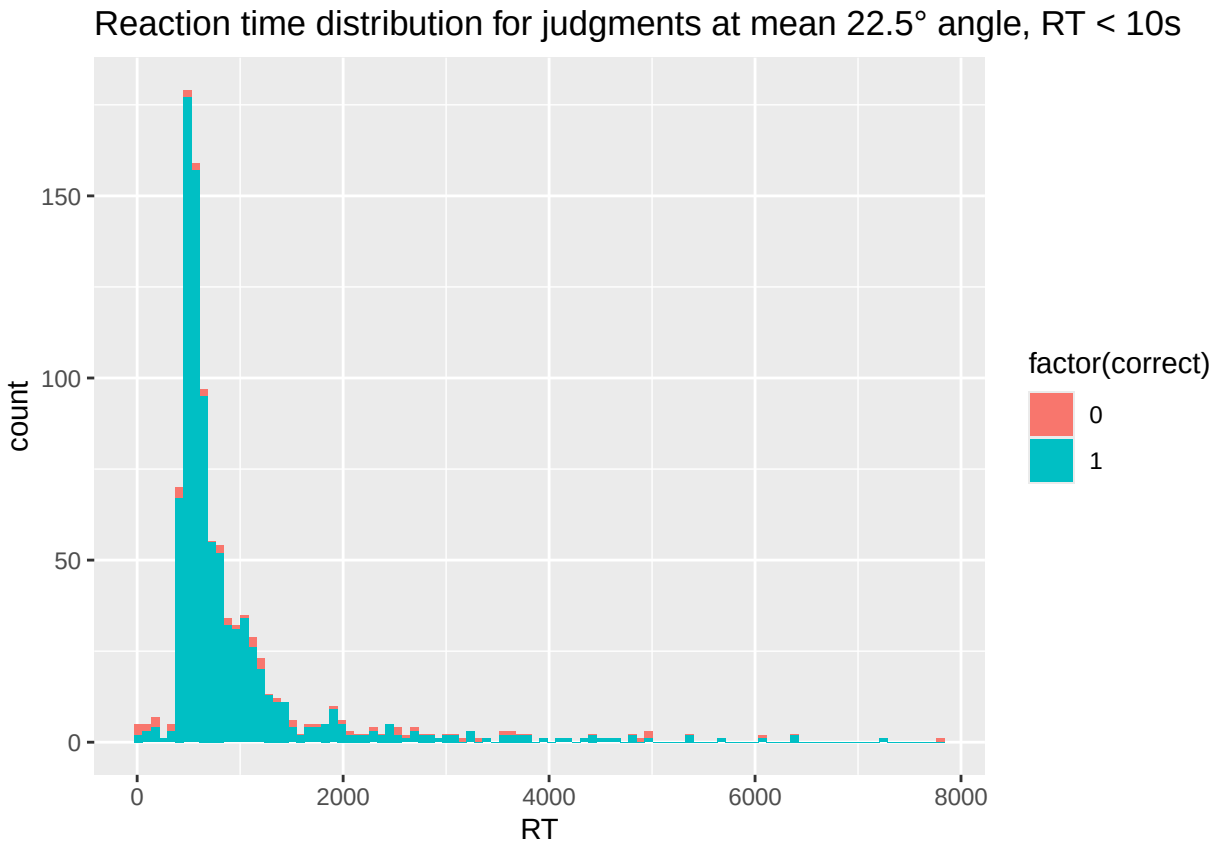
```
data.22 <- data[data$absMean == 22.5,] # filter data to specific trial
noextremes <- data.22[data.22$RT < 10000,] # exlude extreme response data
```

In this tutorial we will focus exclusively on reaction times (RTs). Before conducting analyses, it is a good idea to look at the data.

Exercise: Plot the distribution of decision times in `noextremes`. You can modify the plotting code above.

Solution:

```
ggplot(noextremes) + # mandatory: change data source
  aes(x=RT, # mandatory: change variable plotted on x axis. Could also look at log RTs
    fill=factor(correct)) + # optional: keep color fill, although it's not needed here
  geom_histogram(
    bins=100) + # optional: guess a good # of bins for this variable
  ggtitle("Reaction time distribution for judgments at mean 22.5° angle, RT < 10s")
```



Question: What are some important qualities that a probabilistic model of reaction times should have?

Solution: *It should produce distributions that are compatible with real response times, e.g., it should be real-valued, positive, continuous, and asymmetric. We might also want it to accommodate our theoretical commitments about reaction times, e.g., that there's a floor on reaction times imposed by nerve transmission times (via, say, a shift). It might also be good to account for extreme outliers, but this could be left to a mixture model.*

We discussed the random walk model in tutorial 1, which allowed us to account for both the choice (left or right) and the decision times in speeded-decision tasks. In this tutorial we will use a different approach and model the decision times with a Weibull distribution.

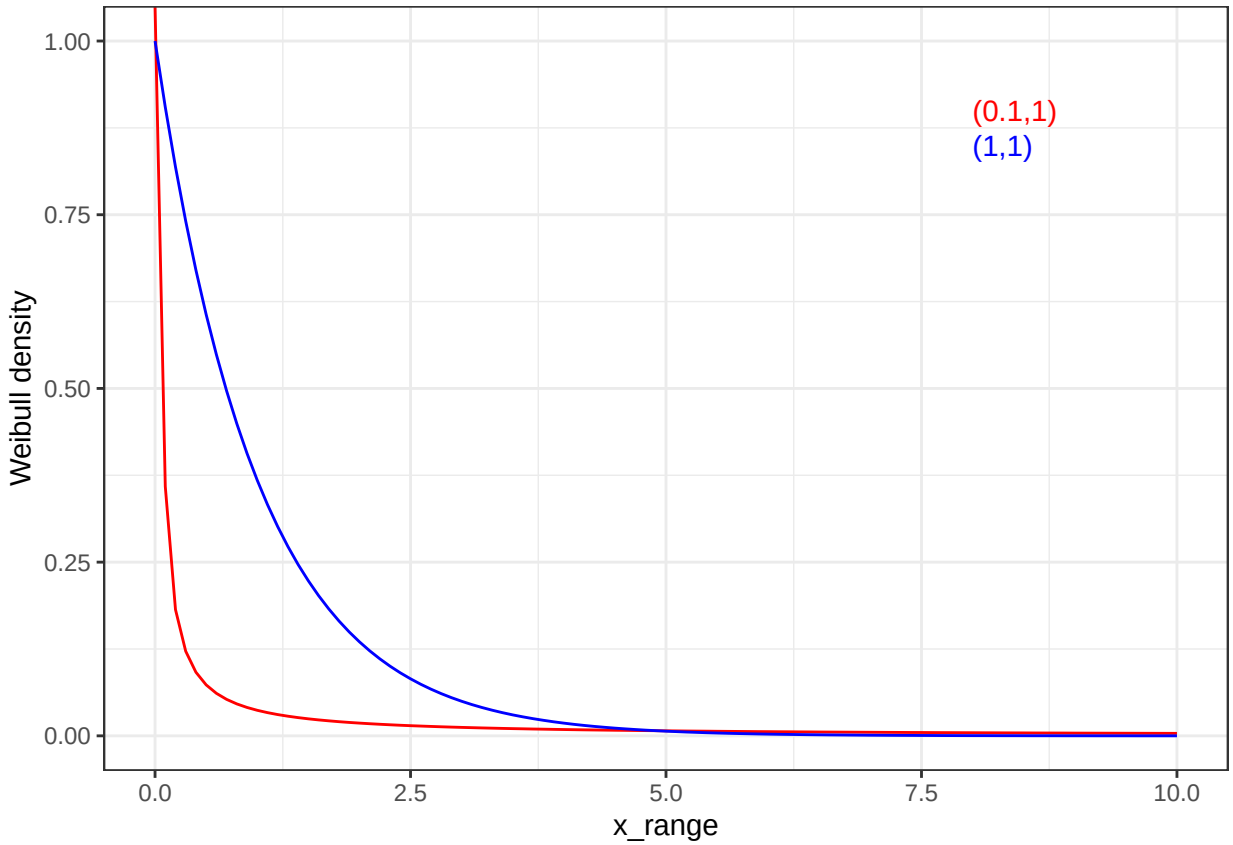
The Weibull distribution has a few advantages over the previous random walk model. First, the random walk model assumes discrete time steps and therefore cannot be directly applied to our data. Second, using a standard statistical distribution allows us to conveniently and directly compute quantities like likelihoods.

Let's get an intuition for what the Weibull distribution looks like. The density function of the Weibull, as well as a random number generators and quantiles are available in R via `dweibull()`, `rweibull()` and `qweibull()`. As you can see from the documentation of these functions, two parameters determine the Weibull density: its scale and shape. We can visualize the effect of varying these parameters over a range of inputs.

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0, \\ 0, & x < 0, \end{cases}$$

The Weibull distribution:

```
x_range <- seq(0,10, by=0.1)

ggplot() +
  aes(x = x_range) + # set same x range for all lines
  geom_line(aes(y = dweibull(x_range, shape=0.1, scale=1)),
            color="red") + # plot *red* line for weibull with shape & scale (0.1, 1)
  annotate("text", x=8, hjust=0,
           y=.9, col="red", label="(0.1,1)") + # red annotation for (0.1, 1)
  geom_line(aes(y = dweibull(x_range, shape=1, scale=1)),
            color="blue") + # plot *blue* line for weibull with shape & scale (1, 1)
  annotate("text", x=8, hjust=0,
           y=.85, col="blue", label="(1,1)") +  # lower, blue annotation for (1, 1)
  ylab("Weibull density") + # label y axis
  theme_bw() # optional: give plot a nicer theme
```

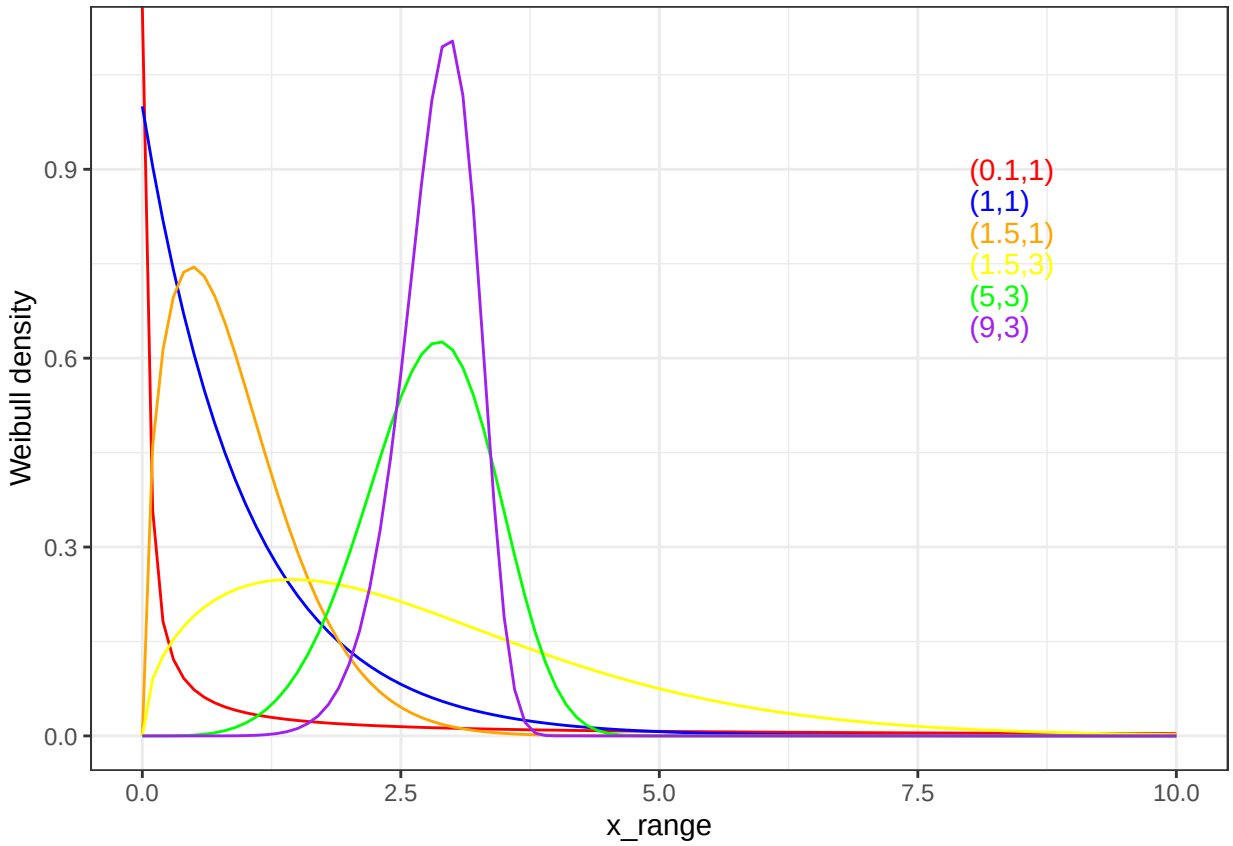


Exercise: Extend the plot above by adding the density of the Weibull distribution for the following shape and scale parameters: [(shape = 1.5, scale = 1), (1.5, 3), (5,3), (9, 3)], using `dweibull(x_range, shape, scale)`. How do the parameters influence the shape of the distribution?

Solution: *A small value for the shape parameter gives a more skewed distribution (when the shape parameter equals 1, we obtain an exponential), and increasing the shape parameter produces a more normal (bell-shaped) distribution. The scale stretches out the distribution (from F&L).*

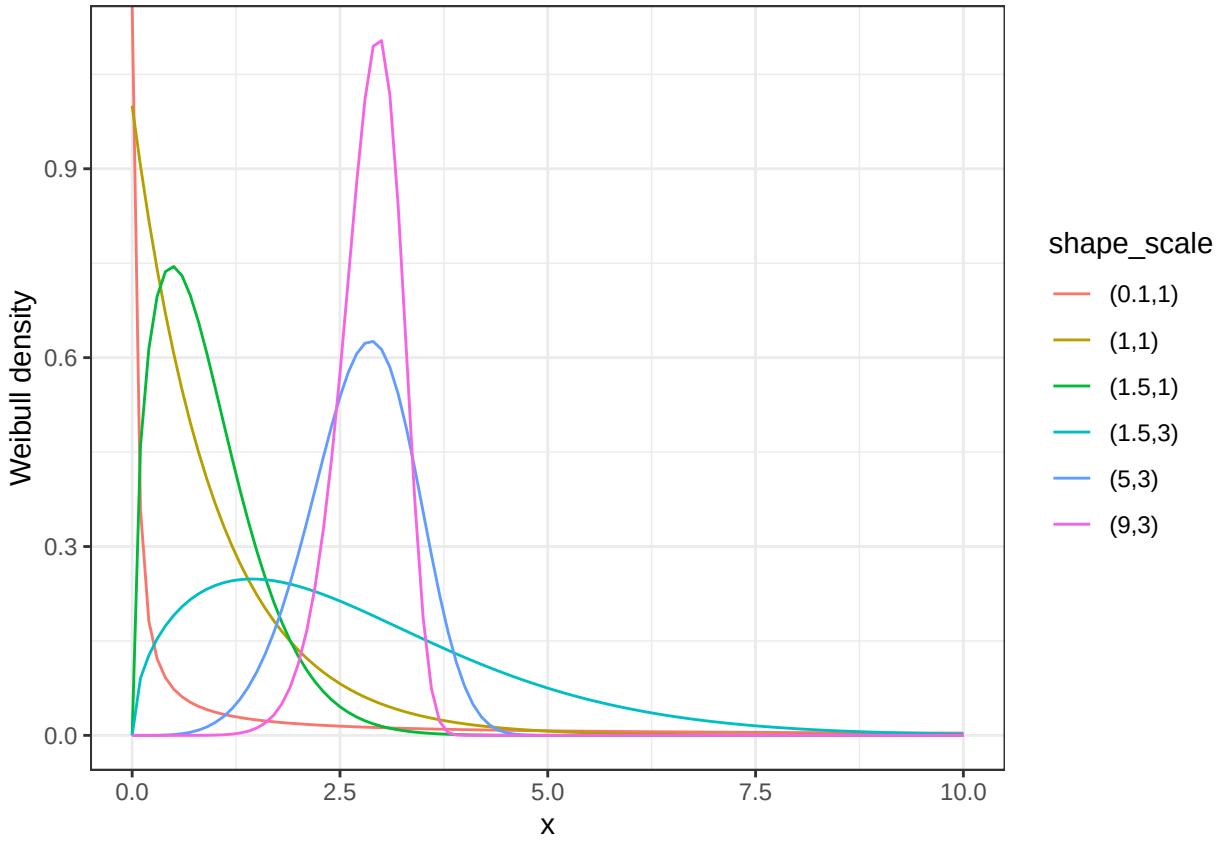
```
# solution A: write it all out as one big plot
# adding layers incrementally like this can be useful during data exploration,
#  although with this many you probably want to do some more code streamlining
#   (see next solution for a better approach overall)

ggplot() +
  aes(x = x_range) + # set same x range for all lines
  geom_line(aes(y = dweibull(x_range, shape=0.1, scale=1)),
            color="red") + # plot *red* line for weibull with shape & scale (0.1, 1)
  annotate("text", x=8, hjust=0,
           y=.9, col="red", label="(0.1,1)") + # red annotation for (0.1, 1)
  geom_line(aes(y = dweibull(x_range, shape=1, scale=1)),
            color="blue") + # plot *blue* line for weibull with shape & scale (1, 1)
  annotate("text", x=8, hjust=0,
           y=.85, col="blue", label="(1,1)") +  # lower, blue annotation for (1, 1)
  geom_line(aes(y = dweibull(x_range, shape=1.5, scale=1)),
            color="orange") +
  annotate("text", x=8, hjust=0,
           y=.8, col="orange", label="(1.5,1)") +
  geom_line(aes(y = dweibull(x_range, shape=1.5, scale=3)),
```

```
            color="yellow") +
  annotate("text", x=8, hjust=0,
           y=.75, col="yellow", label="(1.5,3)") +
  geom_line(aes(y = dweibull(x_range, shape=5, scale=3)),
            color="green") +
  annotate("text", x=8, hjust=0,
           y=.7, col="green", label="(5,3)") +
  geom_line(aes(y = dweibull(x_range, shape=9, scale=3)),
            color="purple") +
  annotate("text", x=8, hjust=0,
           y=.65, col="purple", label="(9,3)") +
  ylab("Weibull density") + # label y axis
  theme_bw()  # optional: give plot a nicer theme
```




```
# Solution B: build desired mappings directly into a data frame,
#   and let ggplot do everything else.
# Coloring and legend are handled automatically, with no repeated code!

# build data frame with specified inputs
n_lines = 6 # number of distributions we're plotting
len_x = length(x_range) # number of steps in the range
wb_df <- data.frame(x = rep(x_range, n_lines),
                    shape = rep(c(0.1, 1, 1.5, 1.5, 5, 9), each=len_x),
                    scale = rep(c(1, 1, 1, 3, 3, 3), each=len_x))
# add derived values
wb_df$shape_scale = with(wb_df, paste0("(", shape, ",", scale, ")"))
wb_df$y = with(wb_df, dweibull(x, shape, scale)) # calculate density for each set of inputs
```

```
ggplot(wb_df) +
  aes(x=x, y=y,
      color=shape_scale) + # setting 'color' aesthetic tells ggplot to group on this
  geom_line() + # so geom layer automatically plots separate lines grouped by color input
  ylab("Weibull density") +
  theme_bw()
```



Fitting Decision Times with the Weibull Distribution

We will augment the Weibull distribution with a *shift* parameter. The shift parameter shifts the entire distribution, changing its mean but leaving the overall shape (after the point it is shifted to) unchanged. The shift parameter has been used in previous research and it allows to represent an initial interval that might correspond to the time required to encode the stimulus.

With this additional parameter the mean of the shifted Weibull is:

```
weibull.mean <- function(shape, scale, shift=0) {
  scale * gamma(1+1/shape) + shift
}
```

We will also define a density function for our shifted Weibull, and a function that returns false if any of our parameters are invalid. We will use these later.

```
dweibull_s <- function(x,shape,scale,shift,log=FALSE) {
  dweibull(x-shift,shape,scale,log)
}
```

```
invalidParams <- function(par) {par[1] <= 0 || par[2] <= 0 || par[3] < 0}
```

We will start our parameter estimation by fitting the mean of the shifted Weibull to the mean decision time in our data set using the root mean squared error (RMSE). We will use the default optimization procedure in R `optim()`. Also note that we are using the parameter order `[shape, scale, shift]` and not `[shift, scale, shape]` as in Farrell and Lewandowsky (2018).

```
wb.rmse <- function(rtData, par) {
  if (invalidParams(par)){
    # for negative shape/scale parameters the Weibull is undefined
    # so we return a high error
    1E100
  }
  else {
    sqrt((weibull.mean(par[1], par[2], par[3]) - mean(rtData))^2)
  }
}
```

Exercise: Fit the Weibull distribution using the `optim` function for the RMSE function above on the decision time (`RT`) data (`noextremes`). The `optim` function requires you to pass starting values to the optimization algorithm; set these to (1, 1, 1), i.e., `c(1,1,1)`.

Solution:

```
rmseOpt <- optim(par = c(1, 1, 1), fn = wb.rmse, rtData = noextremes$RT)
```

Question: What are the parameters resulting from the optimization?

Solution:

```
print(str_interp("The fitted parameters are ${rmseOpt$par}"))
```

```
## [1] "The fitted parameters are c(\"0.1677720570276\", \"1.43242560705477\", \"1.57424899008948\")"
```

Question: What is the corresponding mean of the Weibull for those parameters? What is the error for the optimized parameters?

```
wm <- weibull.mean(rmseOpt$par[1],rmseOpt$par[2],rmseOpt$par[3])
wmrmse <- wb.rmse(noextremes$RT, rmseOpt$par)
print(str_interp("Mean=${wm}; RMSE=${wmrmse}"))
```

```
## [1] "Mean=959.43844767667; RMSE=1.88019355391589e-05"
```

```
# Check the mean of our data
noextremes$RT %>% mean
```

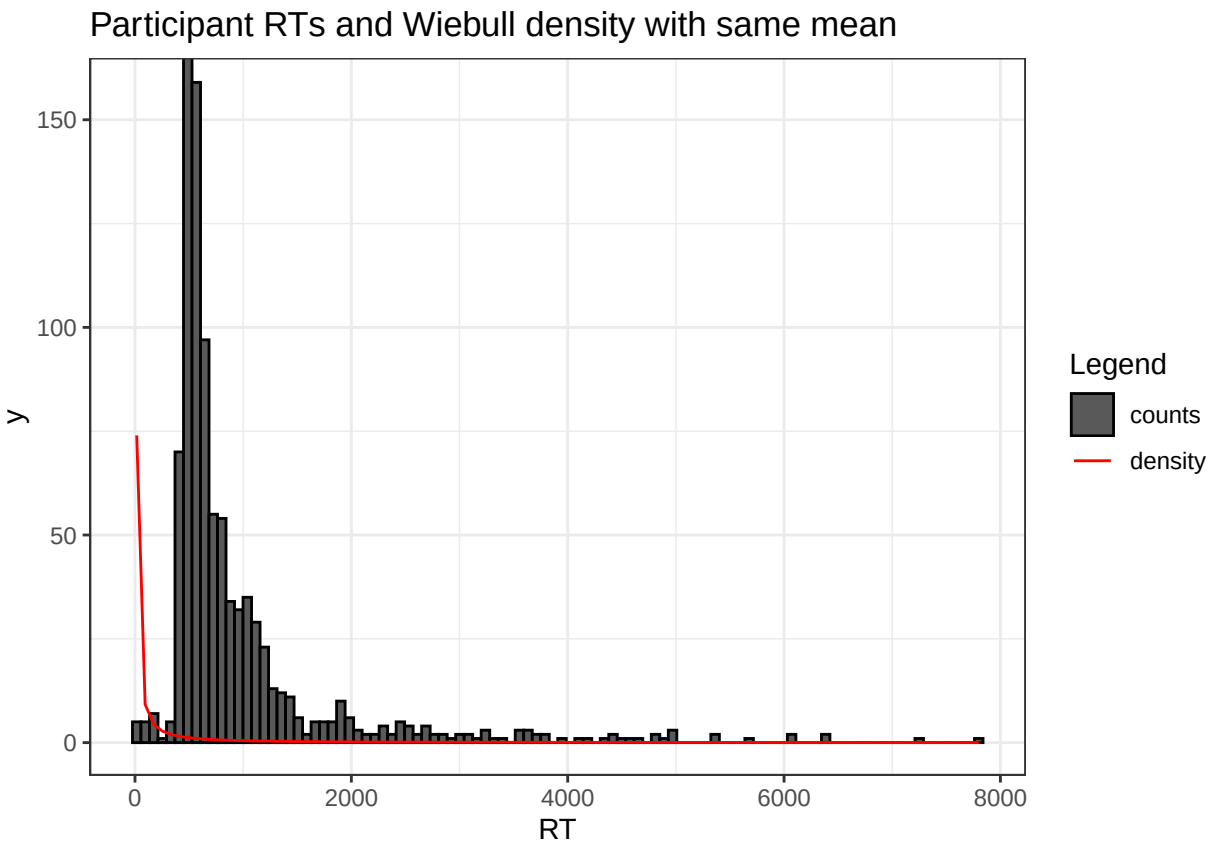
```
## [1] 959.4384
```

Exercise: How well does the mean match the experiment data? Plot the fitted density and the experiment data. Is this a satisfying fit? Do you have any intuitions as to how it might be improved?

Solution:

```
# Let's create a helper function.
# The "stat_function" part of the plot multiplies the density by a number >>1
# to make the density visible against the counts. This is a bit of a hack and
# there are more principled approaches, but they're more involved/fiddly.
rt_vs_weibull <- function(rtDf,wParams) {
ggplot(rtDf, aes(RT,color="counts"))  +
  geom_histogram(bins = 100) +
  stat_function(aes(color="density"),fun = function(x)
```

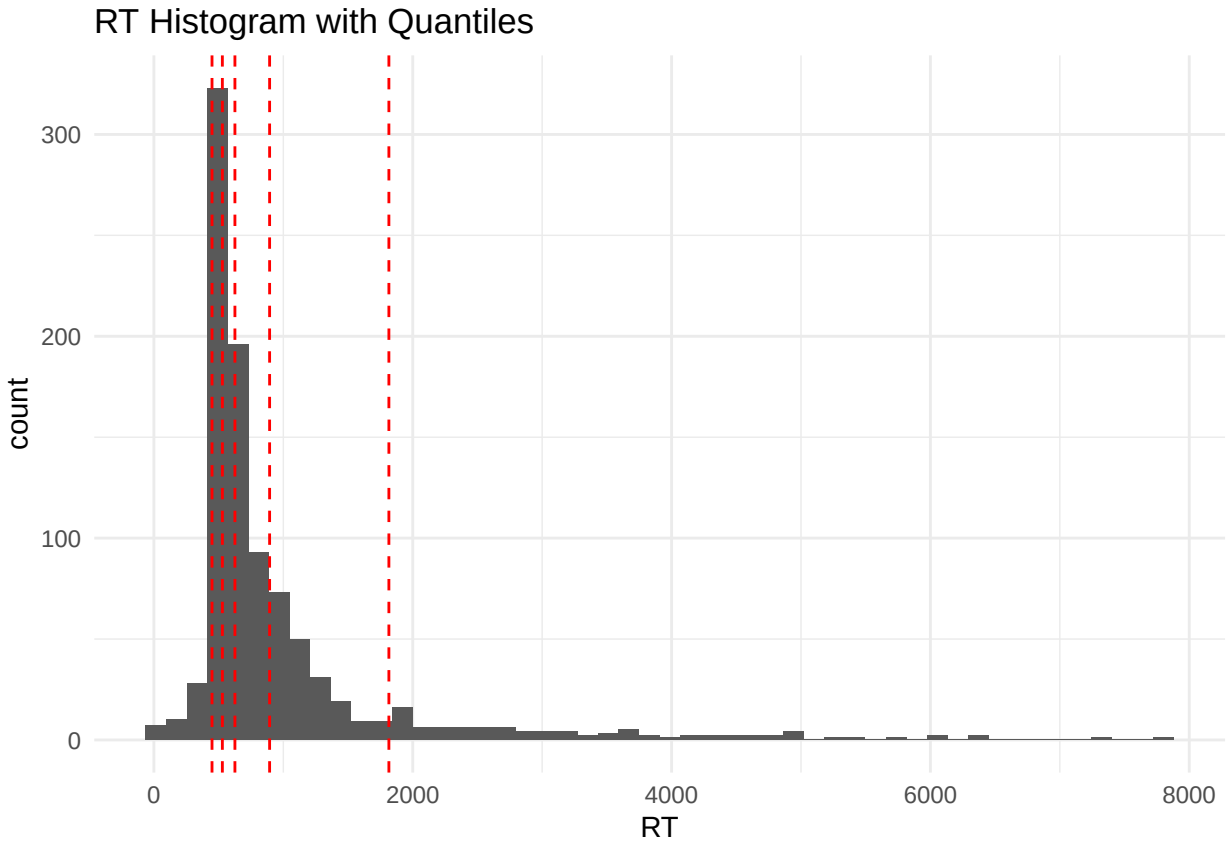
```
    dweibull_s(x, wParams[1],wParams[2],wParams[3],log=FALSE)*20*nrow(rtDf)) +
    theme_bw() +
    coord_cartesian(ylim = c(0, max(15,nrow(rtDf)/6))) +
    scale_colour_manual("Legend", values = c("black", "red"))
}

rt_vs_weibull(noextremes,rmseOpt$par) +
  ggtitle("Participant RTs and Wiebull density with same mean")
```



The fit to the mean is excellent, but the overall correspondence between the distributions may be poor; there is an infinite number of ways the distribution can fit any particular mean. Look at the properties of the Weibull distribution (e.g., on Wikipedia) if it isn't clear why this is.

The textbook notes that we can fit a distribution to the general shape of our empirical distribution by fitting quantiles rather than just the mean:

```
# Show the quantiles of RT in noextremes for c(.1, .3, .5, .7, .9)

quantiles <- quantile(noextremes$RT, c(.1, .3, .5, .7, .9))


# Show the rt histogram with quantiles overlaid
ggplot(noextremes, aes(RT)) +
  geom_histogram(bins = 50) +
  geom_vline(xintercept = quantiles, color = "red", linetype = "dashed") +
  ggtitle("RT Histogram with Quantiles") +
  theme_minimal()
```




```
wb.qrmse <- function(rtData, par) {
  q_p <- c(.1,.3,.5,.7,.9)
  if (invalidParams(par)) {
    1E100 # Invalid params -> bad fit
  }

  empiricalQuantiles <- quantile(rtData, q_p)
  q_pred <- qweibull(q_p, shape=par[1], scale=par[2]) + par[3]
  sqrt(mean((q_pred-empiricalQuantiles)^2))
  sqrt(mean((q_pred-quantile(rtData, q_p))^2))
}
```

Exercise: Fit the shifted Weibull to the decision-time data.

Solution:

```
rmseQuantOpt <- optim(par = c(1, 1, 1), fn = function(x) {
  wb.qrmse(noextremes$RT, x)
})
```

```
## Warning in qweibull(q_p, shape = par[1], scale = par[2]): NaNs produced
## Warning in qweibull(q_p, shape = par[1], scale = par[2]): NaNs produced
## Warning in qweibull(q_p, shape = par[1], scale = par[2]): NaNs produced
```

Question: What parameters did the optimization find? Compare them to the parameters we found using only the mean for the RMSE.

Solution:

```
cat(str_interp("Quantile-fitted params: ${rmseQuantOpt$par};\nmean-fitted: ${rmseOpt$par}"))
```

```
## Quantile-fitted params: c("1.15445639222105", "788.980305815452", "121.269744882442");
## mean-fitted: c("0.1677720570276", "1.43242560705477", "1.57424899008948")
```

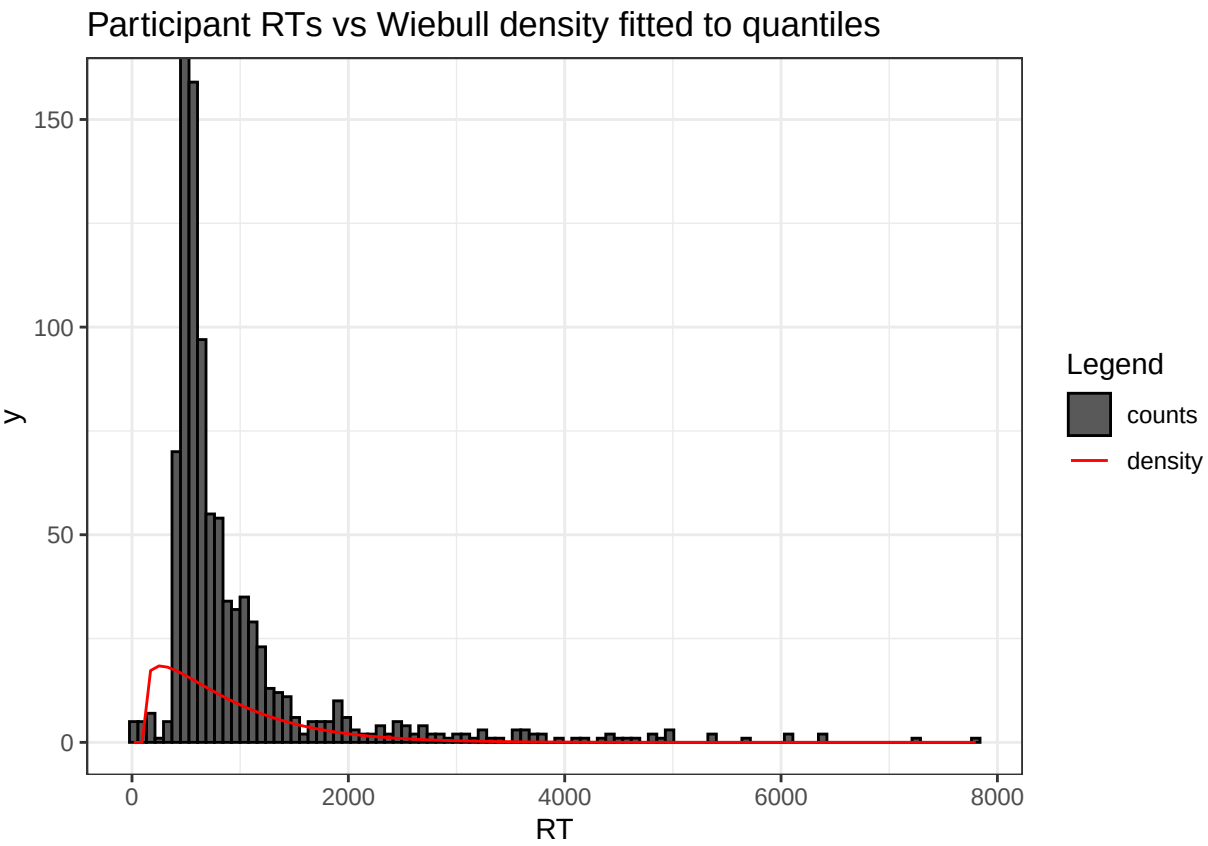
Exercise: Plot the results of the optimization and the participants' decision times – how well does the mean match the experiment data?

Solution:

```
qpars <- rmseQuantOpt$par
cat(str_interp("Quantile-fitted mean: ${weibull.mean(qpars[1],qpars[2],qpars[3])};\nmean-fitted mean: $
```

```
## Quantile-fitted mean: 871.302493012526;
## mean-fitted mean: 959.43844767667
```

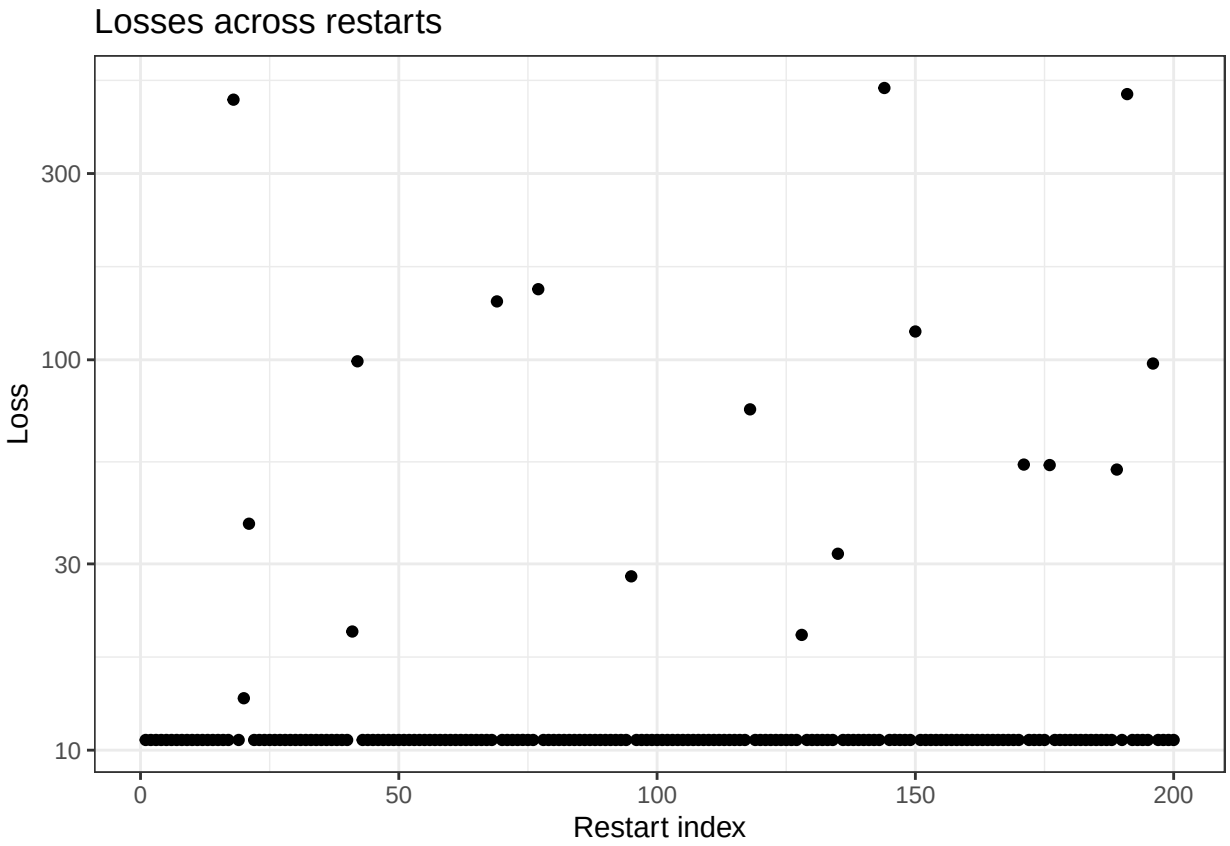
```
rt_vs_weibull(noextremes,rmseQuantOpt$par) +
  ggtitle("Participant RTs vs Wiebull density fitted to quantiles")
```



Exercise: By default `optim` uses the Nelder-Mead optimization procedure. It requires us to provide starting guesses for the parameters. Evaluate the results of the optimization (for the quantile optimization) for a range of starting values. Edit the code below to generate 200 random uniform starting values from 0.1-50 and 0.1-500 (`runif()`) for the scale and shape parameters, respectively. For the shift generate 200 random uniform values between 0 and 16. After updated the code below, run it and explain what it is doing as well as what the output tells us.

```
n_restarts = 200

### Begin solution
scaleStart <- runif(n_restarts, min=0.1, max=50)
```

```r
shapeStart <- runif(n_restarts, min=0.1, max=500)
shiftStart <- runif(n_restarts, min=0.1, max=16)
### End solution

params <- cbind(scaleStart, shapeStart, shiftStart) # make a matrix of start parameters

restartOpt <- function(params,optfn,plotVals=FALSE) {
  # N.B. this procedure involves iterating the optimization process
  #      over our randomly chosen start parameters
  #
  # R functions are often optimized to run efficiently over vectors,
  #   the preference is to *apply* a function over a vector or matrix,
  #   then another function to the resulting output,
  #   rather than looping over the matrix and
  #   performing several steps in each iteration.
  # The following code reflects this stylistic preference.
  # Documentation for the 'apply' family of functions
  #     (i.e. apply, sapply, lapply, tapply) can be a useful reference.
  #
  # Run optimization for all starting parameter values
  opt <- apply(params, 1, # this tells apply to iterate over rows
               optim, fn = optfn)
  ## Extracts the value of the optimized function for each start
  opt_val <- sapply(opt, function(x) x[["value"]])
  ## Extracts optimized parameter values from results
  opt <- opt[order(opt_val)]
  ## Visualizes different runs, if we ask it to
  if(plotVals) {print(ggplot() +
    geom_point(aes(x=1:length(opt_val), y=opt_val)) +
    labs(title="Losses across restarts",
      x = "Restart index", y="Loss") +
    scale_y_log10() +
    theme_bw())
  }
  ## Returns (one) best fit
  opt[[1]]
}

ofn <- function(x) wb.qrmse(noextremes$RT, x)
print(suppressWarnings(restartOpt(params,ofn,plotVals=TRUE)))
```




```
## $par
##  scaleStart  shapeStart  shiftStart
##   0.5783251 321.1379331 456.4019934
##
## $value
## [1] 10.62063
##
## $counts
## function gradient
##      366       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Solution:

The output tells us that the optimization procedures we have used can lead to different results for different starting values, reminding us it is a good idea to run optimization procedures for many starting values.

When we fit the mean of the Weibull to the data we saw that achieving a low error does not guarantee that our model parameters capture the underlying data. In general, it is a good idea to check how model predictions look and how closely they match the actual data, as well as subgroups in our data.

Question: What is the goal of maximum likelihood estimation? On a high level, how does it differ from our approach up to this point? What are some reasons for and against using MLE?

Solution: *The goal of MLE is to estimate model parameters by maximizing the likelihood of the data given*

those parameters. MLE provides an answer to the question "which model instance best fits the data?" when "best fits" is defined as "gives the most evidence for". If θ is a particular assignment of values to the model's parameters and y represents the observed data then the MLE answer to this question is given by:

$$\hat{\theta} = argmax_{\theta} L(\theta|y) = argmax_{\theta} P(y|\theta)$$

Until now, our modeling approach has been to find the parameters which minimized the root mean squared error (RMSE) between the model's predictions and the observed values. For MLE, we define a probability distribution and estimate parameters via likelihood maximization.

We may prefer to use MLE over error minimization because MLE fits nicely into a broader probabilistic approach; F&L note it has attractive properties such as parametrization invariance, consistency, efficiency, and asymptotic normality. RMSE might be preferable in cases where it's difficult to define a probability distribution.

Let's define a negative log likelihood function, relying on the shifted Weibull density we defined earlier.

```
wb.lldev <- function(data,par)  {
  # We are adopting F&W's practice of assigning a very high (but finite)
  # NLL when parameters are invalid or the data will have probability zero.
  # Consider some of the pros and cons of their approach.
  if (invalidParams(par) || any(data < par[3])) {
    return (1E100)
  }
  ll <- dweibull_s(data, shape=par[1], scale=par[2],shift=par[3],log=TRUE)
  # Sometimes people will compute the "deviance", or twice the NLL. This term is
  # relevant in some model selection settings, but the difference isn't so
  # important here, and the MLE is unchanged by this transformation.
  -2*sum(ll)
}
```

Exercise: Find MLEs for the parameters, and compare their log likelihoods to those of the previous estimates. Discuss.

Solution:

```
quantileFitLLD <- wb.lldev(noextremes$RT,rmseQuantOpt$par)
meanFitLLD <- wb.lldev(noextremes$RT,rmseOpt$par)
print(str_interp("LL for the mean-matching parameters: ${meanFitLLD}"))
```

```
## [1] "LL for the mean-matching parameters: 19260.1710459306"
```

```
print(str_interp("LL for the quantile-matching parameters: ${quantileFitLLD}"))
```

```
## [1] "LL for the quantile-matching parameters: 1e+100"
```

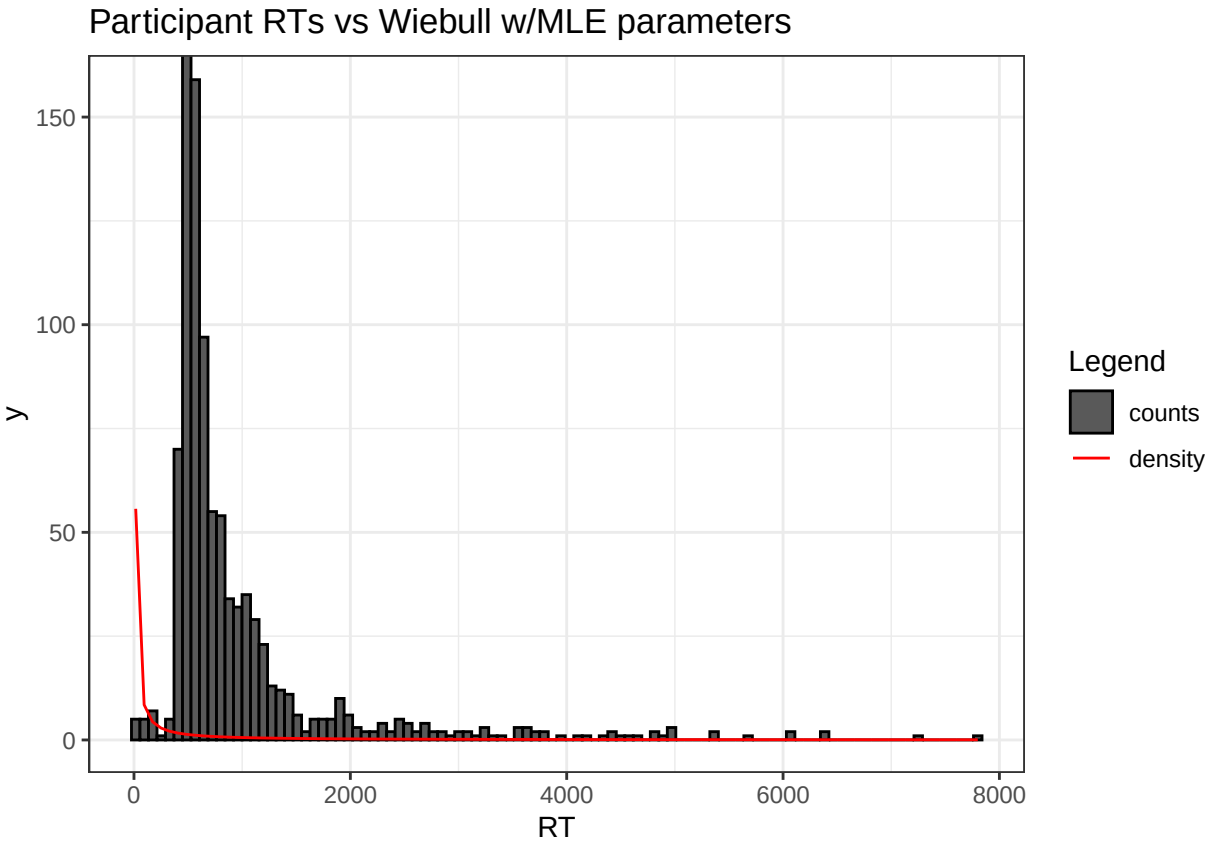
```
optP <- optim(c(1,1,1), fn = function(x) wb.lldev(noextremes$RT,x))
print(optP$value)
```

```
## [1] 18927.94
```

```
print(str_interp("LL for the maximum-likehood parameters: ${optP$value}"))
```

```
## [1] "LL for the maximum-likehood parameters: 18927.944421241"
```

```
rt_vs_weibull(noextremes,optP$par) +
  ggtitle("Participant RTs vs Wiebull w/MLE parameters")
```



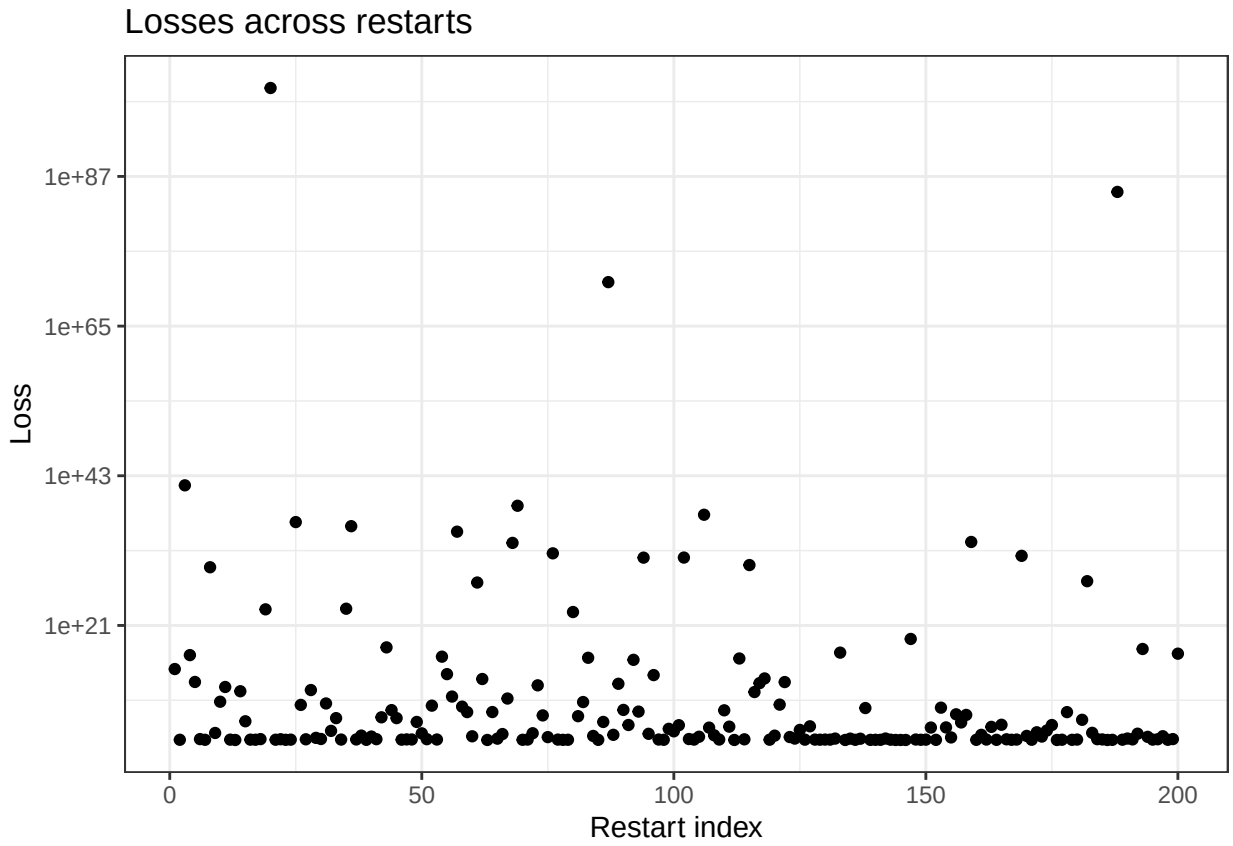
Question: What would happen if we did not use negative log likelihoods in this case?

Solution: *Using positive likelihoods would lead to minimizing the likelihoods, which we don't want. The* `optim` *function minimizes the error, which isn't a big deal but overriding that behavior is more work than adding a negative sign. If we used raw likelihoods, they would round to zero and it would be impossible to compute or optimize our loss function without doing a lot of work to use high-precision arithmetic.*

Exercise: Are we confident that our MLEs are actually maximizing the likelihood? Try the optimization with different starting values. Discuss what you find.

Solution:

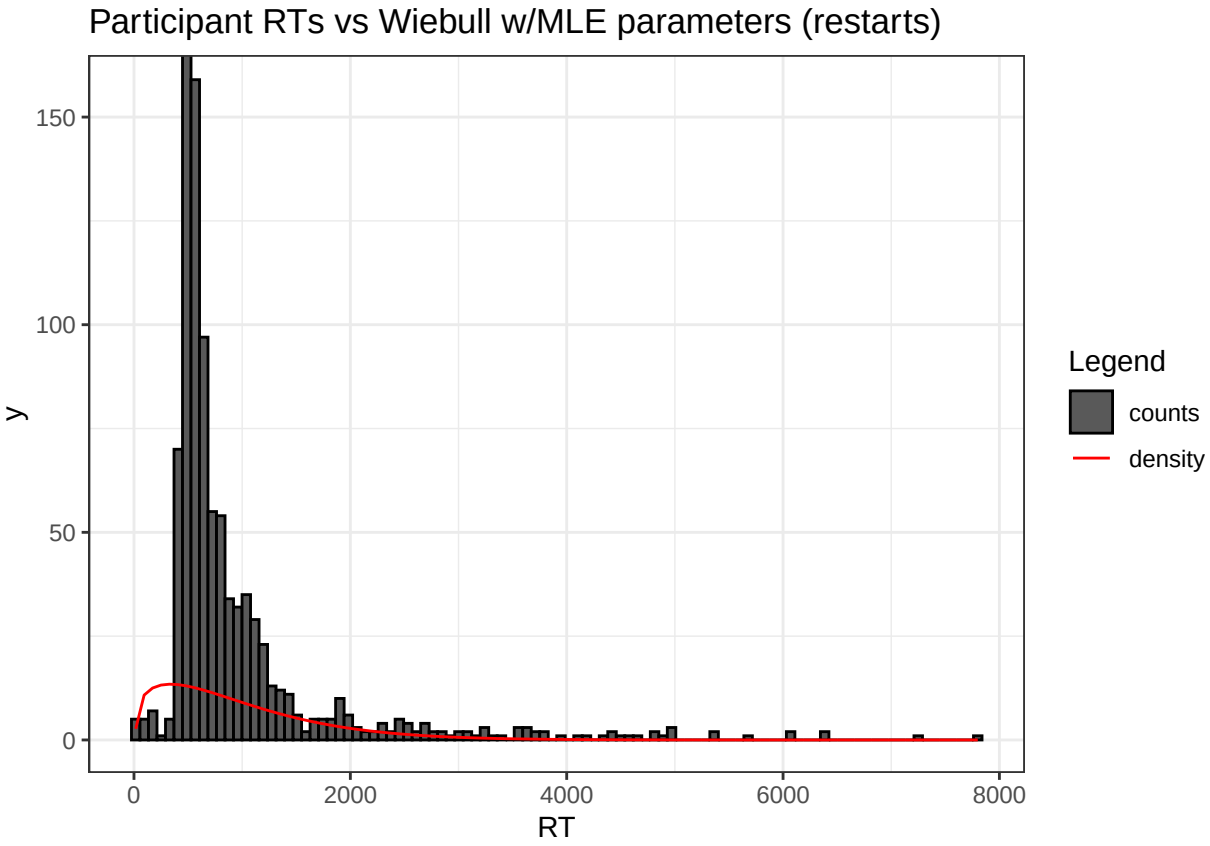
```
opt_ll <- suppressWarnings(restartOpt(params,function(x)  {
  v <- wb.lldev(noextremes$RT,x);if(is.nan(v) || is.infinite(v)) 1E100 else v},plotVals=TRUE))
```




```
cat(str_interp("MLE LL after restarts: ${opt_ll$value};\nparams: ${opt_ll$par}"))
```

```
## MLE LL after restarts: 14677.8756807754;
## params: c("1.28606190671148", "1033.55812453718", "15.4737168346847")
```

```
rt_vs_weibull(noextremes,opt_ll$par) +
  ggtitle("Participant RTs vs Wiebull w/MLE parameters (restarts)")
```



If we fit our data across many restarts, we find that many of the starting places lead to high losses.

Question: If you plot the MLE density against the empirical distribution, does this support an argument for having a mixture model that captures different types of responses? Why or why not? If you were going to fit a finite mixture model, how many components would you use, and why? What distribution(s) would the components have?

Solution: *There appear to be at least two modes in the overall distribution of responses. The Weilbull distribution is unimodal, and the best fitting distribution has a high density in a range of reaction times that are rare for human participants. It might make sense to use 2-3 components: one for the fast responses where maybe people are initiating a response before they even process the scene, one for the slow ones, and one to capture the "long tail" of participants who are perhaps multitasking. We could use normal or log-normal components to start, but more elaborate options, including a Weibull alongside a "no judgment" fast exponentially-distributed component and a heavy-tailed component, might fit the data better.*

References

Farrell, Simon, and Stephan Lewandowsky. 2018. *Computational Modeling of Cognition and Behavior*. Cambridge University Press.

Smith, Philip L., and Douglas Vickers. 1988. "The Accumulator Model of Two-Choice Discrimination." *Journal of Mathematical Psychology* 32 (2): 135–68. https://doi.org/10.1016/0022-2496(88)90043-0.