The Linear Model (LM)

T. Florian Jaeger

September 1, 2024

Readings and assignments in *preparation* of this class

In preparation for class, please **read *James et al. (2013)*, Chapter 3 up to—but not including—3.2 (12 pages)**. This text is quite intuitive and assumes comparatively little background. But if you find yourself getting lost, please stop and first read or watch one of the many more informal introductions to linear models (e.g., this one). Then try to read James et al. (2013).[1] **You should plan for 1-2h of reading and summarizing your questions in preparation for this week of the class.**

[1] James et al. (2013) *An introduction to statistical learning* is the an attempt—a wonderful attempt, full of effective visualization, I think—to make the content of one of the most influential books in machine learning ("Statistical learning" by Tibshirani and colleagues, the developers of bootstrap and other important non-parametric approaches) accessible to a non-expert audience.

Optionally, those of you have interests beyond the basics of the LM, and find it intriguing to ponder about why we use the models we use, might enjoy reading p. 73-78 of McElreath's (2019) *Statistical Rethinking*[2]. It's an amazing book, and these few pages accessibly and thought-provokingly take you through the question of why we often feel comfortable assuming normality in our analyses (as is done in the linear model).

Also optionally, you might find it helpful to read and *work through* this document. **Please note that this document is providing R code only.** Some of the steps described here might have less direct solutions in Matlab or Python, so it is recommended that you start early.

About this document

This PDF is generated from an R markdown document (using the `tufte` package for styling). If you're interested in understanding the code that generated the examples presented below, you can clone the repo. Both the PDF and the R markdown that the PDF is generated from are in the scripts/ folder. You can go through the document line by line, or make changes and 'knit' a new PDF with the press of a button.

The linear model (LM)

LMs (aka linear regression) can be thought of as an example of a much larger family of models that describe an *outcome* variable y (aka the *dependent* variable) as the function g of one or more *predictor* variables $x_1, \ldots, x_k$ (aka *independent* variables), $y = g(x_1, ..., x_k)$. In this sense, the LM is a—very constrained—*predictive model*. Specifically, the LM assumes that y is a draw from a Normal distribution $\mathcal{N}$ with constant standard deviation σ and a mean μ that is a linear combination of all the predictors included in the model:

$$y \sim \mathcal{N}(\mu, \sigma) \tag{1}$$
$$\mu = \beta_1 x_1 + ... \beta_k x_k \tag{2}$$
$$= X\beta \tag{3}$$

Here, y is a column vector of outcome observations of length n (e.g., a column in your data table), X is is called the n x k *model (or design) matrix* (e.g., in the simplest case, a number of k columns in your data table) and β is a vector of weight values of length k (aka coefficients)—one for each of the k predictors in the model.

In the literature, LMs are often written in a number of different ways. For instance, we can say that our expectation for y, $\mathrm{E}[y] = X\beta$, or that y is a linear combination plus some normally distributed residual (prediction) error ϵ, $y = X\beta + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$. You will likely encounter all of these different formulations. You might also encounter formulations that describe each individual observation of y rather than the entire vector of observations, e.g., $\mathrm{E}[y_i] = \beta_1 x_{1,i} + ... + \beta_k x_{k,i}$.

Finally, a common—but arbitrary—convention is to use a different symbol for one particular predictor, the *intercept* α:

$$y \sim \mathcal{N}(\mu, \sigma) \tag{4}$$
$$\mu = \alpha + \beta_1 x_1 + ... \beta_k x_k \tag{5}$$
$$= \alpha x_0 + \beta_1 x_1 + ... \beta_k x_k \tag{6}$$

Here, x_0 is assumed to be the constant 1, i.e., $x_0 = 1$. This convention singles out α, which helps to realize that it has an intuitive interpretation: it is the predicted value of the outcome y when all $x_1 = ... = x_k = 0$.

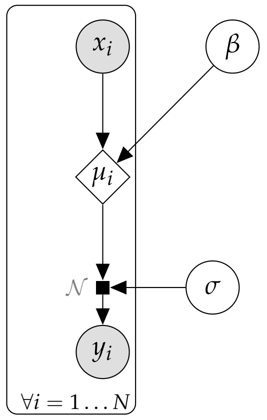


Figure 1: A graphical model visualization of the linear model. Shaded circles are observable variables. White circles are latent variables. These are the variables that constitute the degrees of freedom of the model, the paramters that are fit to the data. Diamonds shapes indicate variables that are determined by other variables. The index $i = 1, ..., N$ refers identifies the observations (usually, the rows) in your data. Note that σ, unlike μ_i is assuemd to be constant across observations. This is the assumption of the homogeneity of variance.

Use of the LM in our field(s)

While the LM can be used as a predictive model—i.e., to make predictions about unseen data—this is rarely done in our fields. Rather, we tend to fit an LM to our data, and then conduct statistical inferences about the *effects* of one or more of the model's predictors based on the best-fitting parameterization of that model. Best-fitting parameterization refers to the choice of βs (and the residual standard deviation σ, though we don't tend to be interested in this parameter) that make the LM deliver the best predictions about the observed values of y.

We'll learn later what "best" means here, and how those best-fitting parameters and their standard errors are obtained. For now, we'll assume that those are known, and focus on what they tell us about the effects of the different predictors that are included in the LM.

Assumptions of the LM

The formulation of the LM in (1)-(3) above entails several assumptions:

1. **Independence**. Each observation y is assumed to be drawn independent of all other observations.[2]
2. **Normality**. y is drawn from a Normal distribution $\mathcal{N}$, and thus is assumed to be able to take any real value from $-\infty$ to $+\infty$.
3. **Homogeneity of variance**. While E[y] $= \bar{}$ is assumed to vary with the predictors X, the residual standard deviation around that mean σ is assumed to be independent of X, and thus constant and identical across all observations.
4. **Linearity**. y is linear in the βs. That is, the *effects* of predictors are assumed to be linear.
5. **Additivity**. The *effects* of predictors are assumed to be additive: $E[y]$ is assumed to be the weighted *sum* of the predictors.[3]

[2] This is a very common assumption, shared e.g., by t-tests, ANOVA, χ^2-tests, and many non-parametric tests.

[3] Assumptions 4 and 5 often confuse learners. To understand them it is important to distinguish between **input variables** and the **predictors** we derive from them. For example, in the case study presented below, one of the input variables is VOT. But we could include both VOT and VOT^2 as predictors in the model. This would model both linear and quadratic effects of VOT, each of which is assumed linearly add to the predicted outcome y. Similarly, we can include interactions of the input variable VOT and some other predictor in the model, which will add their product as a predictor to the model. That is, we can very much use an LM to ask whether two variables have super- or sub-additive interactions. The additivity assumption just means that the effects of the interaction predictor are assumed to be added to the effects of the other predictors in the model.

In practice, many of these assumptions are violated by the data we analyze. Critically, not all of these violations are equally detrimental to the reliability of our conclusions. For example, in simple 2x2 factorial experiments in the cognitive and neurosciences, the linearity assumption is trivially given. Even for experiments with continuous predictors (aka covariates), the linearity assumption *might* not do much harm if the relation between the predictors and the outcome is monotonic and the effects are sufficiently large.

Other assumptions, however, do tend to matter in practice. For example, violations of the assumption of independence—e.g., for repeated-measures data—tend to result in inflated Type I errors (an increased rate of spurious significances). Similarly, violations of the assumption of homogeneity (or its weaker forms like the assumption of homoscedasticity) can substantially inflate Type I error rates. We will return to these issues in later lectures.

Geometric intuitions

I won't repeat the geometric interpretation of LMs here, as the James et al. readings does a great job setting that up. Generally speaking, an LM with k predictors (plus the intercept) can be thought of as defining a hyperplane in a $k+1$ dimensional space, where the additional dimension is the outcome. For instance, for a single predictor, the LM defines a line that describes the predicted outcome as a function of the single predictor. For two predictors, the LM defines a plane in the 3D space defined by the two predictors and the outcome, etc.

Often it is more intuitive, however, to plot the data in fewer dimensions. For example, an LM with a continuous predictor, a categorical predictor (factor), and their interaction, describes multiple lines (that might differ in slope). An LM with a quadratic polynomial of a single continuous input variable will have two continuous predictors derived from that input variable. But rather than to plot the predictions in 3D space, we can plot them in 2D space, showing the effects of both the linear and the quadratic transforms of the input variable along a single axis.

General workflow

Here's the general workflow I'd recommend for an LM analysis, following adequate data checks, variable typing (e.g., making sure that variables that are factors indeed are factors,, etc.) and filtering (e.g., removal of practice and/or catch trials, outlier exclusion, etc.):

- Get data into table format, with each observation of your data corresponding to one row of the table and separate columns for the outcome and each of the input variables.[4]

- Decide how to code the factors in your data, so that each of the factor input variables can be translated into one ore more numerical predictors.[5] Decide whether to transform your continuous input variables, e.g., by centering and/or standardizing them.

- Define the regression formula $y \sim 1 + x_1 + ... + x_k$, where 1 stands for the constant 1. Together with the data table and information about the factor coding (which is stored as an attribute to the factor column in the data.frame in R and in many similar programs), this formula contains all the information that is required for a function intended to fit the model to the data.

  A call to the LM-fitting function typically first implicitly creates the model matrix X. **Looking at the model matrix corresponding to an LM or other model can be a very effective way to learn to understand formula syntax in whatever language you're using.** By default, this model matrix will contain the constant 1 as the first column, followed by one column per *predictor*. Note that this can be more columns than the number of input variables. For

[4] What counts as an observation will depend on both your hypothesis (e.g., whether it is meant to test a hypothesis within a participant or across participants) and the type of model you're using (e.g., an LM vs. a linear mixed-effects model).

[5] In R, the `contrasts()` function and the family of `contr.*` functions are very helpful in setting up the contrasts for your factors, without having the manually change the column in the data frame. This approach means that the factors in the data table remain factors (allowing character values), avoiding the confusion that can result when factors are manually transformed into numeric predictors that are stored in the data table.

example, factor input variables with k distinct levels will result in $k-1$ distinct predictors, and thus $k-1$ distinct columns in the model matrix. Similarly, interactions in the formula add additional columns to the model matrix (more on that in a later lecture). And there are a number of other common formula abbreviations that will expand into multiple columns in the design matrix. If you're not sure what a formula does, I recommend simply looking at the model matrix it creates. E.g., in R:

```
model.matrix(lm(data = d, formula = Response.RT ~ 1 + Condition * VOT)) %>%
  # Select 20 random rows from the model matrix for illustration
  as_tibble() %>%
  slice_sample(n = 20)
```

```
# A tibble: 20 x 6
   `(Intercept)` ConditionShift10 ConditionShift40   VOT `ConditionShift10:VOT` `ConditionShift40:VOT`
           <dbl>            <dbl>            <dbl> <dbl>                  <dbl>                  <dbl>
 1             1                1                0    60                     60                      0
 2             1                0                0    40                      0                      0
 3             1                0                1    -5                      0                     -5
 4             1                0                1    60                      0                     60
 5             1                1                0    25                     25                      0
 6             1                0                1    90                      0                     90
 7             1                0                0    45                      0                      0
 8             1                0                1   100                      0                    100
 9             1                1                0    80                     80                      0
10             1                0                1    70                      0                     70
11             1                1                0    65                     65                      0
12             1                1                0    70                     70                      0
13             1                0                1    90                      0                     90
14             1                0                1    15                      0                     15
15             1                0                0     0                      0                      0
16             1                0                1    40                      0                     40
17             1                0                0     0                      0                      0
18             1                0                1    90                      0                     90
19             1                0                1    35                      0                     35
20             1                0                0    20                      0                      0
```

Note that the formula that we use to define our LM contains only all the *known* or *observable* variables, both the outcome variable and the input variables. The *unknown latent* variables—the βs and σ—are not mentioned in the formula. That's because those are the variables we're asking to be *estimated*, e.g., one β for each predictor. The estimates that we get from fitting the LM are what is reported in the output of the LM if you print or summarize it.

Working through an example

Next, we'll go through an example. We'll use the data from Tan & Jaeger (2024). *Learning to understand an unfamiliar talker: Testing models of adaptive speech perception*. The data come from an experiment on adaptive speech perception, and contain both responses and reaction times from a two-alternative forced-choice task (2AFC). From the abstract of Tan & Jaeger:

> [...] a few minutes of exposure can significantly reduce the processing difficulty listeners experience during initial encounters with an unfamiliar accent. How such adaptation unfolds incrementally, however, remains largely unknown, leaving basic predictions by theories of adaptive speech perception untested. [...] We begin to address these knowledge gaps in a novel incremental exposure-test paradigm. We expose US English listeners to shifted phonetic distributions of word-initial stops (e.g., "dill" vs. "till"), while incrementally assessing cumulative changes in listeners' perception. [...]

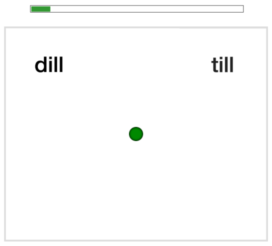
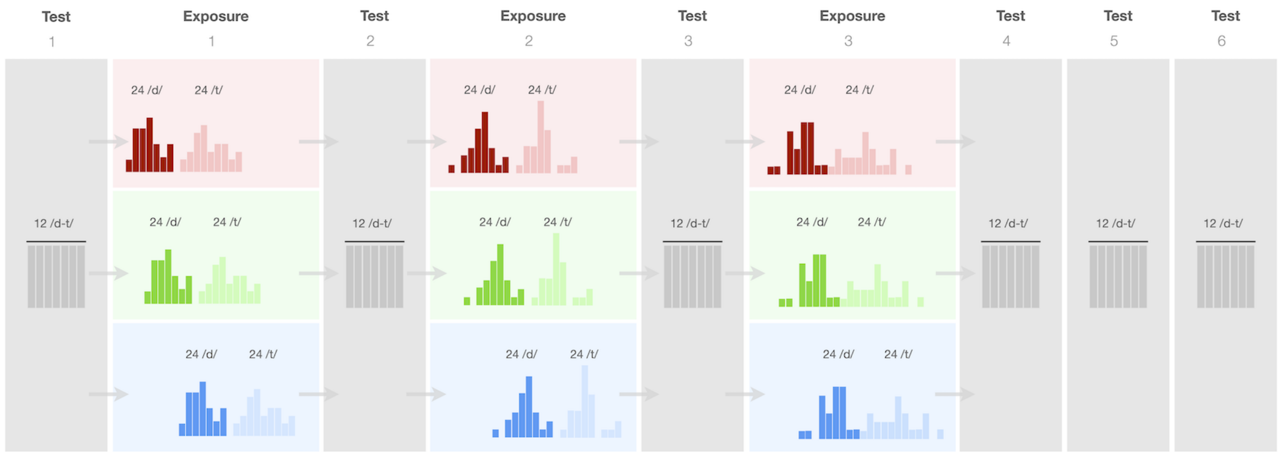


Figure 2: The same simple 2AFC task on all trials. Participants started each trial by clicking the green button. A recording played, and **participants had to answer which of the two words they heard**.

The following figure illustrates the design of the study, consisting of three between-participant exposure conditions, crossed with nine within-participant blocks, each eliciting responses along the phonetic VOT continuum (measured at 12 points). VOT refers to voice onset timing, and is the primary cue to the contrasts like *dip* and *tip* in US English. It is the time between the opening of a speaker's lips (release the air burst) and the start of vocal fold vibrations. As little as 5ms difference in that time can make the difference between hearing the word *dip* (lower VOT) and the word *tip* (higher VOT). The three between-participant condition differed only in the distribution of VOT during exposure blocks.



Specifically, the green condition has the exact same distribution as the red condition, but shifted 10ms to the right; the blue condition is shifted +40ms to the right. This design makes it possible to assess how participants' interpretation of the same physical VOT input during the gray test blocks changes as a function of the differently shifted VOT distributions during the exposure blocks. **This makes it possible to test whether listeners change their categorization functions over VOT as a function of recent input from a talker**.

Figure 3: Incremental exposure-test design. Histograms summarize distribution of voice onset time (VOT) in each block (dark bars: "d"-words; light: "t"-words). The three *between-participant* conditions (rows) differed only in the distribution of VOT during exposure blocks. VOTs in test blocks were identical within & across conditions.

121 participants were included for analysis, each providing 144 exposure trials (3 blocks) and 72 test trials (6 blocks). Participants were unaware of the exposure-test structure of the experiment, and the procedure for exposure and test trials was identical: participants saw two words on the screen (e.g., "dip" and "tip"). Participants then heard a recording of a single word and then had to click the word they heard. All recordings were spoken by the same voice. Here **we use only the data from the six gray test blocks**, leaving 26352 observations. ## The data Since we want to avoid violations of the assumption of independence, we use the data from a single participant from the Shift40 condition, living us with $6 * 12 = 72$ observations. Here's some of the data of that participant. If you downloaded this R markdown document, feel free to change the participant, and see how that changes the results.

```
   ParticipantID Condition Phase Block Trial VOT Response.isT Response.RT
1            124   Shift40  test     1     1  50         TRUE        2325
2            124   Shift40  test     1     2  55         TRUE        1791
3            124   Shift40  test     1     3  25        FALSE        1445
4            124   Shift40  test     1     4  30        FALSE        1999
5            124   Shift40  test     1     5  15        FALSE        1453
6            124   Shift40  test     1     6  65         TRUE        1641
7            124   Shift40  test     1     7  40         TRUE        4900
8            124   Shift40  test     1     8  35         TRUE        1489
9            124   Shift40  test     1     9  -5        FALSE        3543
10           124   Shift40  test     1    10   5        FALSE        1587
11           124   Shift40  test     1    11  45        FALSE        5549
12           124   Shift40  test     1    12  70         TRUE        1211
13           124   Shift40  test     3     1  30        FALSE        1337
14           124   Shift40  test     3     2  45        FALSE        1788
15           124   Shift40  test     3     3  50        FALSE        3213
```

Example 1: Analyzing the effects of VOT on response times

We'll begin by analyzing the effects of VOT on response RTs. This makes response RTs our outcome variable, and VOTs the input variable. The first decision we need to make is how we'd like to code VOT. For example, we could treat VOT as a continuous predictor or as a factor with 12 distinct levels (the number of measurement locations along VOT). And, if we treat VOT as a continuous predictor, we could include only linear VOT or also quadratic VOT or higher order polynomials, provided we have enough data to avoid overfitting.[6] Similarly, we could decide to include log-transformed VOT instead of VOT, or other transforms. As described above, the linearity assumption of LMs does not prevent us from applying non-linear transform to any of the input variables.

[6] As a rule of thumb, the number of coefficients in the LM should be less than 1/20th of the number of data points, $k < n/20$. But that's just a rule of thumb, and details depend on how balanced the data is with regard to the distribution of predictor values.

For the present example, we will treat VOT as a continuous predictor and only include its linear component.[7]

[7] When we treat a predictor x as continuous, we making the assumption that a change in its value of 1 unit has the same effect on the outcome y, regardless of where along x that change is added. That follows from the linearity assumption mentioned above.

```
m <- lm(formula = Response.RT ~ 1 + VOT, data = dd)
summary(m)
```

```
Call:
lm(formula = Response.RT ~ 1 + VOT, data = dd)

Residuals:
   Min     1Q Median     3Q    Max 
-749.9 -451.7 -287.1    2.4 3685.0 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 2205.812    191.953  11.491   <2e-16 ***
VOT           -7.595      4.556  -1.667      0.1 .  
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 856.6 on 70 degrees of freedom
Multiple R-squared:  0.03818,	Adjusted R-squared:  0.02444 
F-statistic: 2.779 on 1 and 70 DF,  p-value: 0.1
```

Write-up

For this particular example, there isn’t much to report. Note that we wouldn’t usually case about the intercept, and rarely report it. So we might write:

> To test our hypothesis, we used linear regression to regress the participant’s reaction time against VOT. The effect of VOT was not significant ($p \geq .1$).

A few considerations:

- Try to resist language that suggests significance even when it was not found, as in “… did not reach significance”.
- Do not take p-values to indicate effect strength. It is ok to write things like “highly significant effect” but that is not the same as a “large effect”. Similarly, a “large effect”—in whatever way that is determined—is not necessarily the same a “theoretically particularly important” effect.

Questions to ponder in class

- How would you interpret the above output?
- How would we expect the output to change if we centered VOT by subtracting the mean VOT from each VOT value before entering it into the model? Which estimates of the model would change? Which ones would stay the same?
- How about if we also standardized VOT (by dividing it by its standard deviation)?
- How could we test whether the slope of VOT changes across Blocks?

Bonus question:

- How could we test a somewhat more interesting hypothesis that more ambiguous VOTs—VOTs that are more likely to elicit an equal proportion of "d"- and "t"-responses—are responded to more slowly?

Fitting

Consider the following fitted LM:

```
Call:
lm(formula = Response.RT ~ 1 + nBlock, data = dd)

Residuals:
   Min     1Q Median     3Q    Max
-897.5 -496.5 -289.7  168.8 3440.5

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2147.32     224.19   9.578 2.33e-14 ***
nBlock        -38.85      36.29  -1.070    0.288
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 866.4 on 70 degrees of freedom
Multiple R-squared:  0.01611,   Adjusted R-squared:  0.00205
F-statistic: 1.146 on 1 and 70 DF,  p-value: 0.2881
```

How did the model fitting algorithm arrive at the estimates and what do they present? What do the standard errors (SEs) of the estimates represent? Intuitively, we know that the estimates are the "best-fitting" estimates for the coefficients. What this means depends on how the goodness of fit is calculated.

One way to find the best-fitting estimates for the coefficients of an LM is to minimize the total prediction error. For each observation in our data, the prediction error or *residual* of the model m is defined as the difference between the predicted value of y, $\mathrm{E_m}[y]$, and the actually observed value of y, i.e., $residual = y - \mathrm{E_m}[y]$. Since we don't want negative and positive prediction errors (resulting from over- and under-predicting y, respectively) to cancel each other out, we cannot simply sum those prediction errors to get an estimate of how well the model fits our data. Instead, we sum the squared residuals, giving us the residual sum of squares (RSS).[8] Finding the best-fitting estimates then becomes the identical to finding the estimates that minimize the RSS.

[8] One could instead use any other function that would result in only positive prediction errors (e.g., taking the *absolute* prediction error). However, using the squared residuals has both appealing properties, and it turns out to be identical—for LMs—to maximizing the likelihood of the data.

To better understand what happens when we find the best-fitting estimates by minimizing the RSS, let's define a few function that help use deliver predictions just like the final LM presented above does, but for arbitrary intercept values a and arbitrary slope values b for the effect of Block. We also define a function that gets the prediction error for each observation in our data, and a function that sums up the squared prediction error:

```
get_prediction <- function(a, b, data = dd) {
  a + b * as.numeric(dd$nBlock)
}

get_prediction_error <- function(a, b, data = dd) {
  dd$Response.RT - get_prediction(a, b, data = data)
}

get_RSS <- function(a, b, data = dd) {
  sum(get_prediction_error(a, b, data = data)^2)
}
```

Now we can use these functions to visualize the predictions and calculate the RSSs for different hypotheses about *a* and *b*. You can use Acrobat Viewer to animate through the following examples (PDF system preview typically won't work):

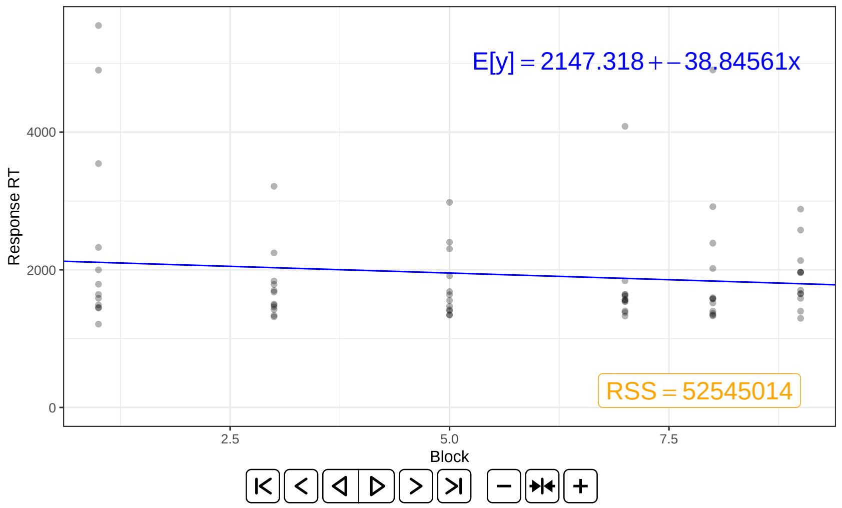


Figure 4: Raw RTs for one participants across the six test blocks (points), along with prediction line for different intercepts *a* and slopes*b* for the model `RT ~ 1 + nBlock`. Compare how the RSS changes, depening on *a* and *b*.

We can also visualize the RSS—i.e., the function we're trying to minimize—depending on the a wide range of values for *a* and *b*. This illustrates two things. First, the estimates provided by the LM at the start of this section indeed minimize the RSS of the model. Second, RSS are a convex function of the parameters in the model (here *a* and *b*), with a single optimum. This very appealing property of LMs allows for efficient fitting algorithms that are *guaranteed* to converge within an arbitrary finite number of steps against the optimal parameters. Basically, it means that going downhill is bound to get you closer to the optimum, rather than stuck in a local minimum (as can be the case for more complex optimization problems, e.g,. for many types of non-linear models).

Here's an example of how one can use a general optimizer to find the best-fitting estimates for an LM. The example uses R's `optim()` function but you can try to implement it in your preferred pro-

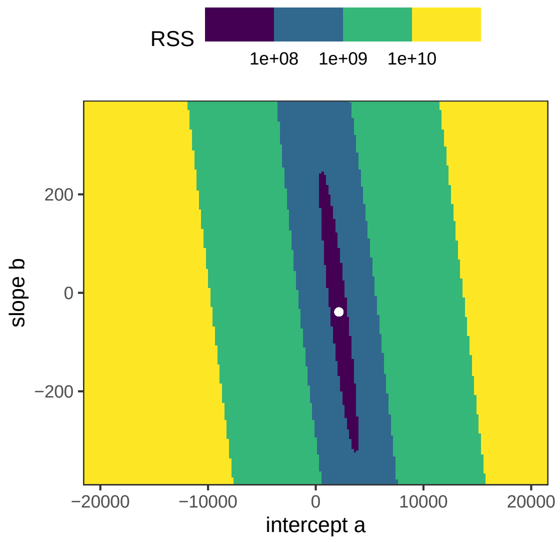


Figure 5: RSS for the model `RT ~ 1 + nBlock`, depending on its intercept a and slope b. The white point shows that location of the estimates given in the LM output at the beginning of this section.

gramming language.[9] You can compare its output to the estimates reported at the start of the section:

[9] By default, `optim()` uses Quasi-Newtonian optimization. This works just fine for the present purpose.

```
# let's store the optimizer's steps in a data.frame
d.trace <- tibble(.rows = 0)

o <-
  optim(
    # Starting values for a and b
    par = c(0, 0),
    fn = function(x) {
      rss <- get_RSS(x[1], x[2]);
      # Uncomment this line to see how optim() searches the parameter space
      # message(paste0("a = ", x[1], ", b = ", x[2], "; RSS = ", rss, "\n"))
      # <<- assigns values to *global* variables (outside the function)
      d.trace <<- d.trace %>% bind_rows(tibble(a = x[1], b = x[2], RSS = rss))
      return(rss)
    })

# print optimal parameters found by optim()
print(o$par)
```

```
[1] 2147.47665  -38.87942
```

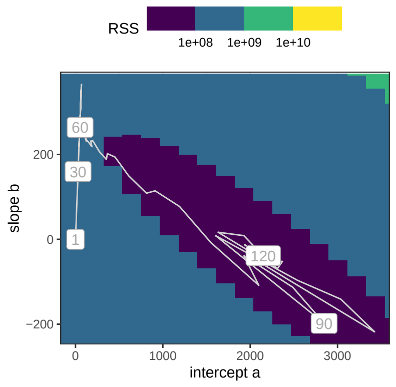


Figure 6: Same as above but zoomed in and showing the path of the optimizer superimposed. Labels show step of optimizer

Apply what you learned

While minimization of RSS works well for LMs, there is a more generally applicable approach that yields the same results for LMs but also can be used for the fitting of many others types of models, *likelihood maximization*. Under this approach, we find the best-fitting estimates that maximize the likelihood of the observed outcomes y. Recall that the LM assumes y to be normally distributed around the predicted $E[y]$. This means we can calculate the *likelihood* of each observation under the hypothesized normal distribution. This normal distribution is a function of the hypothesized values for all the model's coefficients (a and b in the example above) *and* the value hypothesized for *sigma*. For any hypothesis about a, b, and σ, we can get the density of an observation y under $\mathcal{N}(a + bx, \sigma)$. By multiplying the densities of all observations, we can get the data likelihood under the hypothesized values for a, b, and σ. In practice, we instead sum the log-transformed likelihoods. This is numerically more stable, and the x that maximizes $\log(f(x))$ is the same that maximizes $f(x)$.

- Define a function that calculates the sum of log-densities for the model `RT ~ 1 + nBlock` given a set of arguments a, b, and σ.[10]
- Make a figure like the one shown above for RSS. For this figure set σ to the σ from the LM at the start of this section.[11] We don't expect the likelihood values to be identical to the RSS values—they are on completely different scales. But do the surfaces have similar shapes (curvatures)?

[10] In R, `dnorm()` will give you the density of an observation or vector of observations y under a Normal with the specified mean and standard deviations. Use the optional argument `log = TRUE` to get log densities.

[11] In R, use `sigma()` to extract σ from an LM.

Session info

```
devtools::session_info()
```

```
- Session info ------------------------------------------------------------------------------------------------------------
 setting  value
 version  R version 4.3.2 (2023-10-31)
 os       macOS Sonoma 14.5
 system   aarch64, darwin20
 ui       X11
 language (EN)
 collate  en_US.UTF-8
 ctype    en_US.UTF-8
 tz       America/New_York
 date     2024-09-01
 pandoc   3.1.11 @ /Applications/RStudio.app/Contents/Resources/app/quarto/bin/tools/aarch64/ (via rmarkdown)

- Packages ----------------------------------------------------------------------------------------------------------------
 package     * version  date (UTC) lib source
 backports     1.5.0    2024-05-23 [1] CRAN (R 4.3.3)
 broom       * 1.0.6    2024-05-17 [1] CRAN (R 4.3.3)
 cachem        1.1.0    2024-05-16 [1] CRAN (R 4.3.3)
 cli           3.6.2    2023-12-11 [1] CRAN (R 4.3.1)
 colorspace    2.1-0    2023-01-23 [1] CRAN (R 4.3.0)
 crayon        1.5.2    2022-09-29 [1] CRAN (R 4.3.0)
 devtools      2.4.5    2022-10-11 [1] CRAN (R 4.3.0)
 digest        0.6.35   2024-03-11 [1] CRAN (R 4.3.1)
 dplyr       * 1.1.4    2023-11-17 [1] CRAN (R 4.3.1)
 ellipsis      0.3.2    2021-04-29 [1] CRAN (R 4.3.0)
 evaluate      0.23     2023-11-01 [1] CRAN (R 4.3.1)
 fansi         1.0.6    2023-12-08 [1] CRAN (R 4.3.1)
 farver        2.1.2    2024-05-13 [1] CRAN (R 4.3.3)
 fastmap       1.2.0    2024-05-15 [1] CRAN (R 4.3.3)
 forcats     * 1.0.0    2023-01-29 [1] CRAN (R 4.3.0)
 fs            1.6.4    2024-04-25 [1] CRAN (R 4.3.1)
 generics      0.1.3    2022-07-05 [1] CRAN (R 4.3.0)
 gganimate   * 1.0.9    2024-02-27 [1] CRAN (R 4.3.1)
 ggplot2     * 3.5.1    2024-04-23 [1] CRAN (R 4.3.1)
```

```
ggthemes    * 5.1.0    2024-02-10 [1] CRAN (R 4.3.1)
gifski        1.12.0-2 2023-08-12 [1] CRAN (R 4.3.0)
glue          1.7.0    2024-01-09 [1] CRAN (R 4.3.1)
gtable        0.3.5    2024-04-22 [1] CRAN (R 4.3.1)
hms           1.1.3    2023-03-21 [1] CRAN (R 4.3.0)
htmltools     0.5.8.1  2024-04-04 [1] CRAN (R 4.3.1)
htmlwidgets   1.6.4    2023-12-06 [1] CRAN (R 4.3.1)
httpuv        1.6.15   2024-03-26 [1] CRAN (R 4.3.1)
knitr       * 1.47     2024-05-29 [1] CRAN (R 4.3.3)
labeling      0.4.3    2023-08-29 [1] CRAN (R 4.3.0)
later         1.3.2    2023-12-06 [1] CRAN (R 4.3.1)
lifecycle     1.0.4    2023-11-07 [1] CRAN (R 4.3.1)
lubridate   * 1.9.3    2023-09-27 [1] CRAN (R 4.3.1)
magrittr    * 2.0.3    2022-03-30 [1] CRAN (R 4.3.0)
memoise       2.0.1    2021-11-26 [1] CRAN (R 4.3.0)
mime          0.12     2021-09-28 [1] CRAN (R 4.3.0)
miniUI        0.1.1.1  2018-05-18 [1] CRAN (R 4.3.0)
munsell       0.5.1    2024-04-01 [1] CRAN (R 4.3.1)
pillar        1.9.0    2023-03-22 [1] CRAN (R 4.3.0)
pkgbuild      1.4.4    2024-03-17 [1] CRAN (R 4.3.1)
pkgconfig     2.0.3    2019-09-22 [1] CRAN (R 4.3.0)
pkgload       1.3.4    2024-01-16 [1] CRAN (R 4.3.1)
prettyunits   1.2.0    2023-09-24 [1] CRAN (R 4.3.1)
profvis       0.3.8    2023-05-02 [1] CRAN (R 4.3.0)
progress      1.2.3    2023-12-06 [1] CRAN (R 4.3.1)
progressr     0.14.0   2023-08-10 [1] CRAN (R 4.3.0)
promises      1.3.0    2024-04-05 [1] CRAN (R 4.3.1)
purrr       * 1.0.2    2023-08-10 [1] CRAN (R 4.3.0)
R6            2.5.1    2021-08-19 [1] CRAN (R 4.3.0)
Rcpp          1.0.12   2024-01-09 [1] CRAN (R 4.3.1)
readr       * 2.1.5    2024-01-10 [1] CRAN (R 4.3.1)
remotes       2.5.0    2024-03-17 [1] CRAN (R 4.3.1)
rlang         1.1.4    2024-06-04 [1] CRAN (R 4.3.3)
rmarkdown     2.27     2024-05-17 [1] CRAN (R 4.3.3)
rstudioapi    0.16.0   2024-03-24 [1] CRAN (R 4.3.1)
scales        1.3.0    2023-11-28 [1] CRAN (R 4.3.1)
sessioninfo   1.2.2    2021-12-06 [1] CRAN (R 4.3.0)
shiny         1.8.1.1  2024-04-02 [1] CRAN (R 4.3.1)
stringi       1.8.4    2024-05-06 [1] CRAN (R 4.3.1)
stringr     * 1.5.1    2023-11-14 [1] CRAN (R 4.3.1)
tibble      * 3.2.1    2023-03-20 [1] CRAN (R 4.3.0)
tidyr       * 1.3.1    2024-01-24 [1] CRAN (R 4.3.1)
tidyselect    1.2.1    2024-03-11 [1] CRAN (R 4.3.1)
tidyverse   * 2.0.0    2023-02-22 [1] CRAN (R 4.3.0)
timechange    0.3.0    2024-01-18 [1] CRAN (R 4.3.1)
tinytex       0.51     2024-05-06 [1] CRAN (R 4.3.1)
tufte       * 0.13     2023-06-22 [1] CRAN (R 4.3.0)
tweenr        2.0.3    2024-02-26 [1] CRAN (R 4.3.1)
tzdb          0.4.0    2023-05-12 [1] CRAN (R 4.3.0)
urlchecker    1.0.1    2021-11-30 [1] CRAN (R 4.3.0)
usethis       2.2.3    2024-02-19 [1] CRAN (R 4.3.1)
utf8          1.2.4    2023-10-22 [1] CRAN (R 4.3.1)
vctrs         0.6.5    2023-12-01 [1] CRAN (R 4.3.1)
viridisLite   0.4.2    2023-05-02 [1] CRAN (R 4.3.0)
withr         3.0.0    2024-01-16 [1] CRAN (R 4.3.1)
xfun          0.44     2024-05-15 [1] CRAN (R 4.3.3)
xtable        1.8-4    2019-04-21 [1] CRAN (R 4.3.0)
yaml          2.3.8    2023-12-11 [1] CRAN (R 4.3.1)

[1] /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library
```