How to fit models with categorical predictors

Inigo Urrestarazu-Porta

2025-07-09

Blablabla this is the abstract. And this is more of the abstract

Table of contents

List of Figures

List of Tables

```
Warning in file(con, "r"): cannot open file
'scripts/05-frequentist-interaction.R': No such file or directory
```

```
Error in file(con, "r"): cannot open the connection
```

```r
library(magrittr) # pipe operator
library(dplyr) # easy manipulation
library(tidyr)
```

```r
library(ggplot2) # plot
library(ggdist) # plot distributions
library(geomtextpath) # text to plots
library(patchwork) # combine plots
library(viridis)

# set basic theme
theme_set(
  theme_classic( # theme
    base_size = 16 # size of non-graph
    , base_family = 'Carlito' # font
  )
)
```

```
library(gt)
library(gtExtras)
library(kableExtra)
```

```
Error in library(kableExtra): there is no package called 'kableExtra'
```

1 Introduction

Linear regression can only be numerically applied to continuous predictors. Categorical predictors are, by definition, non-continuous and, thus, require a work-around for estimation. There are three main such work-arounds: treating categorical predictors as indicators or dummies, sum-coding them, or indexing the levels of categorical predictors.

Aim of the work.

Our examples and code will be based on R (R Core Team 2025). R is a programming language designed specifically for statistics and data analysis and, to date, it remains the most commonly used programming language to that end. Frequentist hierarchical regression models are implemented with `lme4` [REFS] package, and Bayesian hierarchical regression models with `brms` [REFS], an interface to the probabilistic programming language Stan [REFS]

2 Some basic terminology

The explanation will pivot around three main concepts: *intercept*, *slope* and *level*. In this section, those concepts are defined to ensure that the exposition on the different treatments of categorical predictors can be followed. Finally, conditional and marginal means and weighted averages are introduced.

The first two concepts, *intercept* and *slope*, are not, in fact, related to categorical variables, but to linear regression with continuous variables.

- Intercept: the value of the response or dependent variable, when all (continuous) predictors are 0. For instance, suppose in an investigation of the accuracy (ACC) of L2 speakers in a given task as a function of the years spent learning the second language (ACC ~ years). The intercept would be the average accuracy at 0 years spent learning the L2. If the minimum years spent learning the L2 in the data set are subtracted from each value and regress accuracy on the minimum-centered years, the intercept would be the average accuracy at 0 minimum-centered years spent learning the L2, i.e. the average accuracy at the minimum age in the data set.

- Slope: the change in the response variable for each 1-unit change in a continuous predictor. Retaking the previous example, the slope of years spent learning the L2 would be the change in accuracy for a 1-year difference in years spent the L2, e.g. the difference in accuracy between having spent 2 or 3 years —or any two years 1 unit apart— learning the L2.

Levels are exclusive of factors or categorical variables.

- Level: Each of the distinct units of a factor or categorical variable. For example, if investigating between-language differences, *language* is a factor or categorical variable —a variable that can only take a fixed set of values—, and each language in the data set are levels, e.g. Basque or Finish.

Finally, we introduce conditional, marginal and weighted means, since a clear understanding of thse concepts will make it easier to understand how the approaches to treating categorical predictors differ. However, these three concepts may be better explained through an example.

Suppose there are only two languages in the world, A and B, and 30% of the world's population speaks language A, and 70% speak language B. Next, imagine that, on average, syllables have a duration of 200 milliseconds in language A, whilst in language B syllables have a mean duration of 300 milliseconds. The mean duration of syllables in language A and language B are their respective conditional means, 200 and 300 ms. In other words, conditional means are the expected average outcome, given some value of another variable, in this case, given language A or language B.

As indicated by the name, weighted means are combinations of conditional means, given some weights. For instance, if language A and language B were weighted equally —technically, any weighting can be used, the syllables would have a weighted mean of $\frac{200+300}{2} = 200 \times 0.5 + 300 \times 0.5 =$ 250 ms.

However, in this particular case, it is known that there are more speakers to language B than to language A. 30% of the world's population is a speaker of language A, and the remaining 70% is a speaker of language B. Hence, the conditional means of the average syllable duration of language A and language B could be weighted by their probability distribution, and get the marginal mean duration of syllables in the world's languages. In this toy example, the marginal syllable duration would be $200 \times 0.3 + 300 \times 0.7 =$ 270 ms. Hence, a marginal mean is a particular kind of weighted mean of conditional means, where the weights come from the probability distribution of the variable that it is conditioned on.

3 The data: Salig et al. (2024)

It is often hard to reuse real data collected for scientific purposes because it requires a great familiarity with the research or even the discipline in hand. For instance, using measures used in phonetics may pose a burden on linguists that do not work in phonetics.

An alternative approach would be to simulate the data. However, simulating the data requires a steady walk through the simulation code, Yet, the goal lays in illustrating how to fit models with categorical predictors, and we do not want to shift the focus of this work towards how to simulate (realistic) data (in R).

Hence, the data in Salig et al. (2024) will be utilised. The data in Salig et al. (2024) is real data collected for scientific purposes within linguistics. Although understanding the exact goals, background and impact in the field of the study requires as much understanding as any other piece of research, the data can be simplified enough to be accessible to a wide range of readers within linguistics or, more broadly, the cognitive sciences.

The data consist of reading times in milliseconds (ms) of sentences by English-Spanish bilinguals, whose dominant language is English overall. The data were collected from a classic 2x2 *within-participant* design, where sentences could start either in English or in Spanish; then, the sentences remained in the language in which they started, or the language could be switch to the other one after some point. Thus, the reading time values come from one of the following four possibilities:

Table 1: Design grid in Salig et al. (2024)

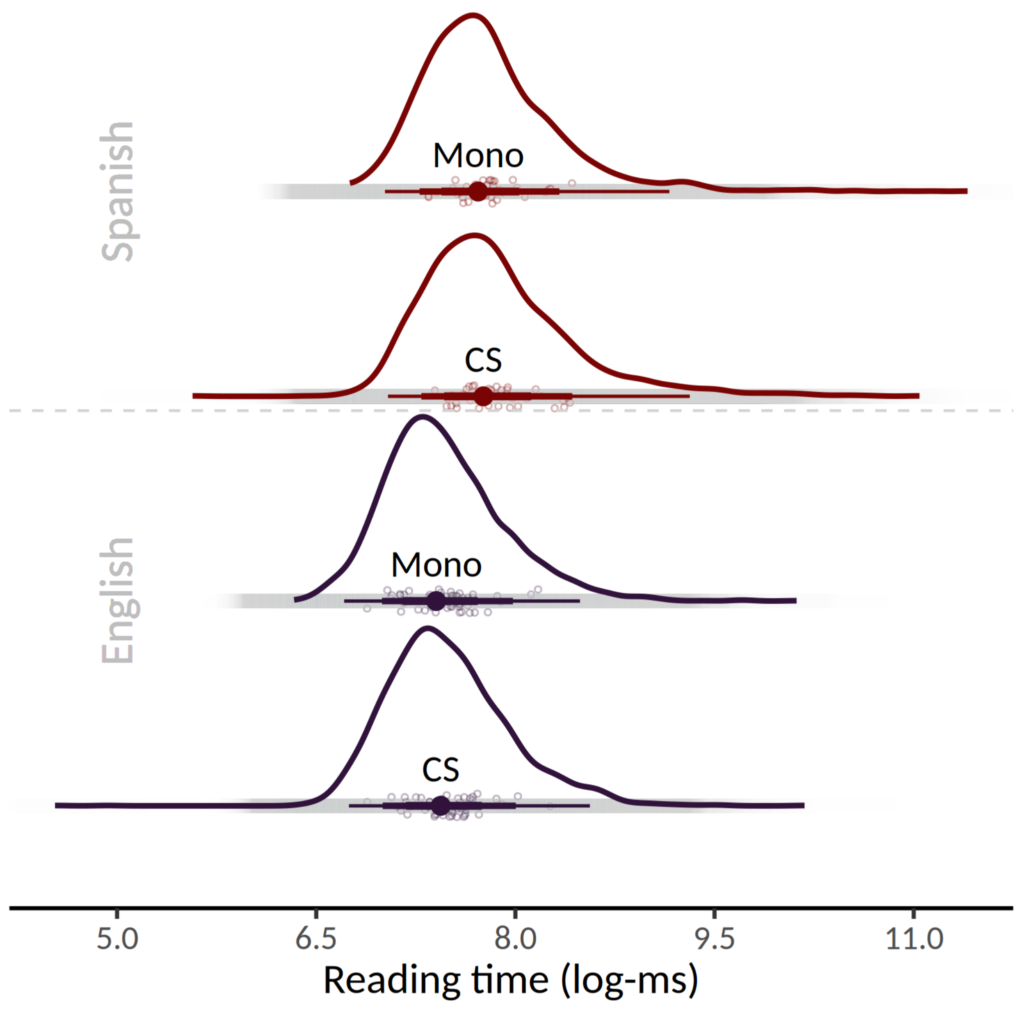
```
Error in kable_styling(., full_width = F, latex_options = "striped"): could not find functio
```

```r
salig.df <- read.csv(
  'data/salig2024.csv'
)

# variable renaming
# to type less

# subject id to S
colnames(salig.df) <-
  gsub(
    '^id$'
    , 'S'
    , colnames(salig.df)
  )

# group to G
# otherwise marginaleffects does tot work
colnames(salig.df) <-
  gsub(
    '^group$'
    , 'G'
    , colnames(salig.df)
  )

# reaction times to T
colnames(salig.df) <-
  gsub(
    '^QuestionRT$'
    , 'T'
    , colnames(salig.df)
  )

# baselang to L
colnames(salig.df) <-
  gsub(
    '^baselang$'
    , 'L'
    , colnames(salig.df)
  )

# item number to I
colnames(salig.df) <-
  gsub(
    '^itemnum$'
    , 'I'
    , colnames(salig.df)
  )
```

The data in Salig et al. (2024) are summarised in the following figure:

In addition to the four conditions, namely English monolingual, English with code-switching, Spanish monolingual and Spanish with code-switching, the data have repeated measures for participant and sentence / experimental unit.

With these data a number of frequentist generalised hierarchical regression models will be fitted. Firstly, we will illustrate the three main ways of treating categorical predictors, with a single categorical predictor. Then, a second categorical predictor will be incorporated to explain what *interaction*s mean in the context of categorical predictors. Finally, the same approaches will be applied to Bayesian regression models, since treatment given to categorical predictors impacts the parameters to be estimated directly from the model and, thus, the priors that have to be incorporated to the model.



(a) Four elements summarise the information, the point and intervals, the densities, the grey shadows, and the jittered points. The point shows the median reading time, and the intervals contain 50%, 75% and 95% of obserservations. The densities are smoothed distributions of the data set. The gray shadows are observations; hence, the darker the shadow, the more observations there are in that area. Finally, the smaller jittered points indicate the mean of each participant for that condition.

Figure 1: Summary of the empirical reading times (log-ms) in Salig et al. (2024), by experimental condition.

4 Treatments of categorical predictors

```
library(lme4)
library(broom)
library(broom.mixed)
```

```
library(emmeans)
library(marginaleffects)
```

```
Error in library(marginaleffects): there is no package called 'marginaleffects'
```

In this section the three main ways of treating categorical predictors within linear regression will be presented, namely treating categorical predictors as indicators, sum-coding them, and treating them as indexes. To that end, the data in Salig et al. (2024) will be used, as described in Section 3. Although the data have two relevant categorical predictors, language and code-switching, the three approaches will be first illustrated using language as the only categorical predictor. This will allow to focus in the differences between these approaches, before moving to combinations of (categorical) predictors.

However, we highlight that three approaches lead to the same results —as shown in Section 7, the treatment given to categorical predictors may slightly influence the exact results in Bayesian regression models—, and that the same information can be recovered regardless of the treatment given to the categorical predictor(s). Thus, the difference between treatments lays in how the results of the model are presented. Besides, in the published literature examples of all three treatments can be found. Hence, it is crucial to understand them and their differences to correctly interpret the results of studies.

First, indicator treatment is discussed, and the terms *intercept*, *conditional mean* and *grand mean* — or, sometimes, *marginal mean*— are illustrated. Then how sum-coding works is explained, and finally indexing of categorical predictors is introduced.

4.1 Indicator treatment

When treated as indicators (also called *dummy variables* or *dummies*), a level of the categorical predictor is taken as the reference, and the remaining levels are expressed as differences from that reference. Indicator treatment is the default treatment of categorical predictors in R. If the categorical predictor is coded as an vector object of class *factor*, the reference level will be the first level specified in the factor. In turn, if there is no particular order specified, i.e. the categorical predictor is a vector of class *character*, the reference will be the level that comes first in alphabetical order. The reference level is labelled as *intercept* in the output of model, even if it is not the average value of the outcome, when all (continuous) predictors are 0 —recall the definition of the term *intercept* above.

4.1.1 Fitting the model

Since R treats the categorical predictors as indicators by default, just adding the categorical predictor to the model formula returns this treatment. `+ 1` can be added to the formula, between the tilde `~` and the first predictor, to explicitly indicate that the model should be fitted with an *intercept*. Hence, both formulas are equivalent.

```
# fit-treatment-indicator
fit.lang.indicator <- lmer(
  log(T) ~ L + (1 | S) + (1 | I)
  , data = salig.df
)

# equivalent formula
fit.lang.indicator <- lmer(
  log(T) ~ 1 + L + (1 | S) + (1 | I)
  , data = salig.df
)
```

These are the population-level (or fixed) estimates of a model with a two-level categorical predictor treated as an indicator.

```
# extract population-level estimates
# from the model
indicator.lang.coef <- broom.mixed::tidy(
  fit.lang.indicator
  , conf.int = T
)

# the first two columns of the
# tidy output are irrelevant
# and I don't want std error and statistic
# obtain the colnames
# just to keep the relevant ones
cols <- colnames(indicator.lang.coef)

indicator.lang.coef <- indicator.lang.coef[
  indicator.lang.coef$effect == 'fixed'
  ,
  grepl(
    'term|^est|^conf'
    , cols
```

	estimate	95% CI
(Intercept)	7.48	[7.4, 7.55]
LSpanish	0.34	[0.24, 0.45]

```r
  )
]

indicator.lang.coef %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    -starts_with('conf')
  ) %>%
  gt(
    , rowname_col = 'term'
  )
```

The model returns two parameters, namely *(Intercept)* and *LSpanish*, one per level of the categorical predictor added to the regression formula. Yet, the parameter *(Intercept)* does not represent the average reading time when all predictors are 0. Since the sole categorical predictor added to the regression is categorical, it cannot be 0. It must be either *English* or *Spanish*. Hence, the intercept of a model fitted with categorical predictors as indicators represents the average response when all numeric predictor are 0, and the categorical predictor(s) are at their reference level(s). In other words, the *intercept* is the conditional mean of the reference level of the categorical predictor —when all numeric predictors are 0, i.e. it is the mean reaction time, conditional on being an *English* sentence —*English* comes before *Spanish* alphabetically— and every other predictor being equal to 0. Since there are no other predictors, it can be said that it is the conditional mean of *English* sentences.

Then, *LSpanish* is the difference between the average reading time of the Spanish sentences and the intercept, the average reading time of the English sentences —when all numeric predictors are 0. In other words, *LSpanish* is the *pairwise comparison* between the levels of the categorical predictor *L*, when all numerical predictors are 0. In this case, there is no other predictor than *L*, but note that these pairwise comparisons need not be performed when the rest of the predictors are 0, and that the result of these comparisons might not be the same for different values of the rest of the categorical predictors.

Although the output of models fitted with categorical predictors as indicators only returns the conditional mean of the reference level of the categorical predictor(s) —labelled as the `Intercept`—, the rest of the conditional means can be retrieved afterwards.

...

```
cmeans.emmeans <- data.frame(
  emmeans(fit.lang.indicator, ~ L)
)

cmeans.emmeans %>%
  select(
    L
    , emmean
    , asymp.LCL
    , asymp.UCL
  ) %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , asymp.LCL
      , ', '
      , asymp.UCL
      , ']'
    )
  ) %>%
  select(
    -starts_with('asymp')
  ) %>%
  gt(
    , rowname_col = 'L'
  )
```

	emmean	95% CI
English	7.48	[7.4, 7.55]
Spanish	7.82	[7.73, 7.91]

```r
cmeans.marginaleffects <- data.frame(
  predictions(
    fit.lang.indicator
    , variables = 'L'
    , by = 'L'
    , re.form = NA
  )
)
```

```
Error in predictions(fit.lang.indicator, variables = "L", by = "L", re.form = NA): could not
```

```r
cmeans.marginaleffects %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    L
    , estimate
    , `95% CI`
  ) %>%
  gt(
    , rowname_col = 'L'
  )
```

```
Error: object 'cmeans.marginaleffects' not found
```

emmean	95% CI
7.65	[7.58, 7.71]

Finally, weighted averages can be calculated too.

```
gmean.emmeans <- data.frame(
  emmeans(fit.lang.indicator, ~ 1)
)

gmean.emmeans %>%
  select(
    , emmean
    , asymp.LCL
    , asymp.UCL
  ) %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , asymp.LCL
      , ', '
      , asymp.UCL
      , ']'
    )
  ) %>%
  select(
    -starts_with('asymp')
  ) %>%
  gt()
```

```
gmean.marginaleffects <- data.frame(
  predictions(
    fit.lang.indicator
    # without the next 2 lines
    # it returns the marginal avg
    # with the weights in the dataset
    , variables = 'L'
    , by = T
    , re.form = NA
  )
)
```

```
Error in predictions(fit.lang.indicator, variables = "L", by = T, re.form = NA): could not f
```

```
gmean.marginaleffects %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    , estimate
    , `95% CI`
  ) %>%
  gt()
```

```
Error: object 'gmean.marginaleffects' not found
```

4.2 Sum-coding treatment

Next, we shall discuss sum-coding. Sum-coding consists of replacing the levels of the categorical predictor(s) with (equidistant) numeric values so that their sum equals to 1. For instance, with a 2-level categorical predictor, one level would be replaced with -.5, and the other with .5. Another common sum-coding scheme is -1/1, instead of -.5/.5.

When sum-coded, the categorical predictors become computationally like any continuous predictor. Thus, unlike with indicator treatment, sum-coding returns an intercept estimate, and that estimate is a true intercept. The intercept of a model with sum-coded categorical predictors represents the mean response variable, when all predictors are 0, including the categorical predictor(s). This means that the intercept is the weighted mean between the levels of the categorical predictor, assuming all levels are equally distributed, i.e. it is the mean of the means of all the levels of the categorical predictor(s). If there are no numeric predictors to consider, the intercept of the model with sum-coded categorical predictors will be the marginal mean.

The interpretation of the slope of the sum-coded predictor depends on the exact sum-coding scheme used. When using a -1/1 scheme, the slope represents the distance from each level to the grand mean (or sometimes the marginal mean) of the data set, and it answers to the question "Do the levels of the categorical predictor differ from the grand mean?". Half that estimate would be, if the categorical predictors has just 2 levels, the difference directly obtained from the model where the categorical predictor was treated as an indicator, i.e. the difference between the levels of the categorical predictors. The difference between levels is the meaning of the slope when the categorical predictor(s) are sum-coded following the -.5/.5 scheme.

4.2.1 Fitting the model

```
salig.df$s1_l <- ifelse(
  salig.df$L == 'English'
  , -1
  , 1
)

salig.df$s.5_l <- ifelse(
  salig.df$L == 'English'
  , -.5
  , .5
)
```

```
fit.lang.sum.1 <- lmer(
  log(T) ~ s1_l + (1 | S) + (1 | I)
  , data = salig.df
)

fit.lang.sum.05 <- lmer(
  log(T) ~ s.5_l + (1 | S) + (1 | I)
  , data = salig.df
)
```

Alternatively, sum-coding can be set as the default treatment of all categorical predictors , although the only possible sum-coding scheme for categorical predictors with two levels is -1/1:

```
# change default contrast method
# to sum-coding
# R handles it under the hood
options(contrasts = c('contr.sum','contr.sum'))
```

When the default treatment is set to sum-coding, the predictors provided to the regression are the original ones, and R handles the transformation to numeric values under the hood. For instance, we would run the following code to run the exact same regression as before; note that the name of the categorical predictor is exactly the same as in the indicator treatment, and not the ‘s_’ variable that we created before.

```
fit.lang.indicator <- lmer(
  log(T) ~ L + (1 | S) + (1 | I)
  , data = salig.df
)

# equivalent formula
fit.lang.indicator <- lmer(
  log(T) ~ 1 + L + (1 | S) + (1 | I)
  , data = salig.df
)
```

4.2.2 Output of the model

```
scoded.lang.coef.5 <- broom.mixed::tidy(
  fit.lang.sum.05
  , conf.int = T
)

scoded.lang.coef.5 <-
  scoded.lang.coef.5[
    scoded.lang.coef.5$effect == 'fixed'
    # cols inherited from the indicator model
    , cols[3:ncol(scoded.lang.coef.5)]
  ]

scoded.lang.coef.1 <- broom.mixed::tidy(
  fit.lang.sum.1
  , conf.int = T
)
```

```r
scoded.lang.coef.1 <-
  scoded.lang.coef.1[
    scoded.lang.coef.1$effect == 'fixed'
    , cols[3:ncol(scoded.lang.coef.1)]
  ]
```

4.2.3 Conditional means

```r
cmeans.scoded.indicator.5.emm <- data.frame(
  emmeans(
    fit.lang.sum.05
    , ~ s.5_l
  )
)
```

```
Note: D.f. calculations have been disabled because the number of observations exceeds 3000.
To enable adjustments, add the argument 'pbkrtest.limit = 9024' (or larger)
[or, globally, 'set emm_options(pbkrtest.limit = 9024)' or larger];
but be warned that this may result in large computation time and memory use.
```

```
Note: D.f. calculations have been disabled because the number of observations exceeds 3000.
To enable adjustments, add the argument 'lmerTest.limit = 9024' (or larger)
[or, globally, 'set emm_options(lmerTest.limit = 9024)' or larger];
but be warned that this may result in large computation time and memory use.
```

```r
cmeans.scoded.indicator.1.emm <- data.frame(
  emmeans(
    fit.lang.sum.1
    , ~ s1_l
  )
)
```

```
Note: D.f. calculations have been disabled because the number of observations exceeds 3000.
To enable adjustments, add the argument 'pbkrtest.limit = 9024' (or larger)
[or, globally, 'set emm_options(pbkrtest.limit = 9024)' or larger];
but be warned that this may result in large computation time and memory use.
Note: D.f. calculations have been disabled because the number of observations exceeds 3000.
To enable adjustments, add the argument 'lmerTest.limit = 9024' (or larger)
```

```
[or, globally, 'set emm_options(lmerTest.limit = 9024)' or larger];
but be warned that this may result in large computation time and memory use.
```

```
cmeans.scoded.indicator.5.marg <- data.frame(
  predictions(
    fit.lang.sum.05
    , by = 's.5_l'
    , re.form = NA
  )
)
```

```
Error in predictions(fit.lang.sum.05, by = "s.5_l", re.form = NA): could not find function "p
```

```
cmeans.scoded.indicator.1.marg <- data.frame(
  predictions(
    fit.lang.sum.1
    , by = 's1_l'
    , re.form = NA
  )
)
```

```
Error in predictions(fit.lang.sum.1, by = "s1_l", re.form = NA): could not find function "pre
```

4.3 Indexing treatment

When indexing the categorical predictor the model estimates a conditional mean for each level of the predictor, and no contrast between them. Hence, the interpretation of the output of the model is straightforward, but the contrasts have to be computed afterwards.

4.3.1 Fitting the model

```
fit.lang.index <- lmer(
  log(T) ~ 0 + L + (1 | S) + (1 | I)
  , data = salig.df
)
```

4.3.2 Output of the model

```r
# extract population-level estimates
# from the model
index.lang.coef <- broom.mixed::tidy(
  fit.lang.index
  , conf.int = T
)

index.lang.coef <- index.lang.coef[
  index.lang.coef$effect == 'fixed'
  ,
  grepl(
    'term|^est|^conf'
    , cols
  )
]

index.lang.coef %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    -starts_with('conf')
  ) %>%
  gt(
    , rowname_col = 'term'
  )
```

	estimate	95% CI
LEnglish	7.48	[7.4, 7.55]
LSpanish	7.82	[7.73, 7.91]

4.3.3 Difference between levels

```r
index.dif.emm <- data.frame(
  emmeans(
    fit.lang.index, pairwise ~ L
  )$contrasts
)
```

```
Note: D.f. calculations have been disabled because the number of observations exceeds 3000.
To enable adjustments, add the argument 'pbkrtest.limit = 9024' (or larger)
[or, globally, 'set emm_options(pbkrtest.limit = 9024)' or larger];
but be warned that this may result in large computation time and memory use.
```

```
Note: D.f. calculations have been disabled because the number of observations exceeds 3000.
To enable adjustments, add the argument 'lmerTest.limit = 9024' (or larger)
[or, globally, 'set emm_options(lmerTest.limit = 9024)' or larger];
but be warned that this may result in large computation time and memory use.
```

4.3.4 Marginal mean

```r
gmean.emmeans <- data.frame(
  emmeans(fit.lang.index, ~ 1)
)
```

```
Note: D.f. calculations have been disabled because the number of observations exceeds 3000.
To enable adjustments, add the argument 'pbkrtest.limit = 9024' (or larger)
[or, globally, 'set emm_options(pbkrtest.limit = 9024)' or larger];
but be warned that this may result in large computation time and memory use.
```

```
Note: D.f. calculations have been disabled because the number of observations exceeds 3000.
To enable adjustments, add the argument 'lmerTest.limit = 9024' (or larger)
[or, globally, 'set emm_options(lmerTest.limit = 9024)' or larger];
but be warned that this may result in large computation time and memory use.
```

emmean	95% CI
7.65	[7.58, 7.71]

```r
gmean.emmeans %>%
  select(
    , emmean
    , asymp.LCL
    , asymp.UCL
  ) %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , asymp.LCL
      , ', '
      , asymp.UCL
      , ']'
    )
  ) %>%
  select(
    -starts_with('asymp')
  ) %>%
  gt()
```

```r
gmean.marginaleffects <- data.frame(
  predictions(
    fit.lang.index
    , re.form = NA
    , variables = 'L'
    , by = T
  )
)
```

```
Error in predictions(fit.lang.index, re.form = NA, variables = "L", by = T): could not find
```

```
gmean.marginaleffects %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    , estimate
    , `95% CI`
  ) %>%
  gt()
```

```
Error: object 'gmean.marginaleffects' not found
```

5 Interactions

5.1 Explanation of interaction with numeric predictors

5.2 Interaction between categorical and continuous predictors

5.3 Interaction between two (or more) categorical predictors

6 Pros and cons of each approach

7 Treatment of categorical predictors in Bayesian regression

The treatment of categorical predictors is not inherently different in Bayesian regression. Yet, one of the crucial differences between Bayesian and traditional, frequentist statistics is that the former incorporates prior information, knowledge or belief in the form of prior distributions of the parameters that will be estimated by the model. As we have shown before, all three treatments allow to recover the same pieces of information. However, the exact parameters estimated by the model change depending on the treatment given to categorical predictors. Hence, the treatment given to categorical predictors determines which prior distributions have to be supplied to the model.

```r
# load package
library(brms)
# auto-parallel
# use as many cores as chains
# if ncores <= nchains
options(mc.cores = parallel::detectCores())
# use threading (split chains)
rstan::rstan_options(threads_per_chain = 2)
# write stan code to temp file
rstan::rstan_options(auto_write = TRUE)
# use cmdstanr backend
# it's much faster
options(brms.backend = "cmdstanr")
# call it explicitly
# so that it's cited directly
# otherwise it is used under the hood
# but grateful does not pick it
library(cmdstanr)
# seed for models
# ensure exact replication
```

When treating categorical predictors as indicators, a prior for the *intercept*, i.e. the reference level, has to be provided, alongside priors for the differences for the rest of the intercepts. Besides, if there are interactions between multiple predictors, priors for those interactions are needed. This is problematic to some extent, since we are covertly assuming that the conditional means of the levels that are not the reference have a greater uncertainty than the reference level (McElreath 2020, p.).

8 Conclusion

9 Session info

```r
# grateful to create a text
# with the packages used directly
# dependencies are not cited
# because some are creating a problem
# with R4.5
grateful::cite_packages(
  output = 'paragraph'
  , out.dir = ('.')
  , pkgs = 'All'
  , cite.tidyverse = F
  , omit = NULL # include grateful
  , include.RStudio = F
  , dependencies = F
)
```

```
Error in (function (pkg, lib.loc = NULL) : there is no package called 'kableExtra'
```

```r
# obtain session info
sessionInfo()
```

```
R version 4.5.0 (2025-04-11)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.2 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;  LAPACK version

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

time zone: Etc/UTC
tzcode source: system (glibc)

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base

other attached packages:
 [1] cmdstanr_0.9.0      brms_2.22.0         Rcpp_1.0.14
 [4] emmeans_1.11.1      broom.mixed_0.2.9.6 broom_1.0.8
 [7] lme4_1.1-37         Matrix_1.7-3        gtExtras_0.6.0
[10] gt_1.0.0            viridis_0.6.5       viridisLite_0.4.2
[13] patchwork_1.3.0     geomtextpath_0.1.5  ggdist_3.3.3
[16] ggplot2_3.5.2       tidyr_1.3.1         dplyr_1.1.4
[19] magrittr_2.0.3

loaded via a namespace (and not attached):
 [1] Rdpack_2.6.4          gridExtra_2.3         grateful_0.2.12
 [4] inline_0.3.21         rematch2_2.1.2        rlang_1.1.6
 [7] furrr_0.3.1           matrixStats_1.5.0     compiler_4.5.0
[10] loo_2.8.0.9000        systemfonts_1.2.3     vctrs_0.6.5
[13] stringr_1.5.1         pkgconfig_2.0.3       fastmap_1.2.0
[16] backports_1.5.0       fontawesome_0.5.3     labeling_0.4.3
[19] rmarkdown_2.29        ps_1.9.1              nloptr_2.2.1
[22] purrr_1.0.4           xfun_0.52             jsonlite_2.0.0
[25] parallel_4.5.0        R6_2.6.1              stringi_1.8.7
[28] RColorBrewer_1.1-3    StanHeaders_2.32.10   parallelly_1.45.0
[31] boot_1.3-31           estimability_1.5.1    rstan_2.32.7
[34] knitr_1.50            bayesplot_1.12.0      splines_4.5.0
[37] tidyselect_1.2.1      abind_1.4-8           yaml_2.3.10
[40] codetools_0.2-20      curl_6.3.0            processx_3.8.6
[43] listenv_0.9.1         pkgbuild_1.4.8        lattice_0.22-7
[46] tibble_3.2.1          withr_3.0.2           bridgesampling_1.1-2
[49] posterior_1.6.1       coda_0.19-4.1         evaluate_1.0.3
[52] future_1.58.0         RcppParallel_5.1.10   xml2_1.3.8
[55] pillar_1.10.2         tensorA_0.36.2.1      checkmate_2.3.2
[58] renv_1.1.4            stats4_4.5.0          reformulas_0.4.1
[61] distributional_0.5.0 generics_0.1.4        paletteer_1.6.0
[64] rstantools_2.4.0      scales_1.4.0          minqa_1.2.8
[67] globals_0.18.0        glue_1.8.0            tools_4.5.0
[70] forcats_1.0.0         mvtnorm_1.3-3         grid_4.5.0
[73] rbibutils_2.3         QuickJSR_1.7.0        nlme_3.1-168
[76] cli_3.6.5             textshaping_1.0.1     Brobdingnag_1.2-9
[79] V8_6.0.4              gtable_0.3.6          digest_0.6.37
[82] farver_2.1.2          htmltools_0.5.8.1     lifecycle_1.0.4
[85] MASS_7.3-65
```

10 Funding

References

McElreath, Richard. 2020. *Statistical Rethinking*. Second edition. Chapman & Hall/CRC Texts in Statistical Science Series. Boca Raton: Taylor & Francis Group.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Salig, Lauren K., Jorge R. Valdés Kroff, L. Robert Slevc, and Jared M. Novick. 2024. "Linking Frequency to Bilingual Switch Costs During Real-Time Sentence Comprehension." *Bilingualism: Language and Cognition* 27 (1): 25–40. https://doi.org/10.1017/S1366728923000366.

Appendix: Code used

```r
#
knitr::read_chunk(
  'scripts/00-setup-packages.R'
)
knitr::read_chunk(
  'scripts/01-data-design-grid.R'
)
knitr::read_chunk(
  'scripts/02-read-data.R'
)
knitr::read_chunk(
  'scripts/03-plot-descriptive.R'
)
knitr::read_chunk(
  'scripts/04-frequentist-lang-indicator.R'
)
knitr::read_chunk(
  'scripts/05-frequentist-lang-scoded.R'
)
knitr::read_chunk(
  'scripts/06-frequentist-lang-indexed.R'
)
knitr::read_chunk(
  'scripts/05-frequentist-interaction.R'
)
```

```r
knitr::read_chunk(
  'scripts/99-session-info.R'
)


#___ tbl-design-grid-html
design.tb %>%
  # create gt object
  gt::gt(
    # languages as rownames
    rowname_col = 'language'
  ) %>%
  gt::tab_stubhead(label = "Language") %>% # add theming
  gtExtras::gt_theme_nytimes()

#___ tbl-design-grid-pdf
design.tb %>%
  kbl(
    , booktabs = T
  ) %>%
  kable_styling(
    full_width = F
    , latex_options = 'striped'
  )

salig.df <- read.csv(
  'data/salig2024.csv'
)

# variable renaming
# to type less

# subject id to S
colnames(salig.df) <-
  gsub(
    '^id$'
    , 'S'
    , colnames(salig.df)
  )

# group to G
# otherwise marginaleffects does tot work
```

```r
colnames(salig.df) <-
  gsub(
    '^group$'
    , 'G'
    , colnames(salig.df)
  )

# reaction times to T
colnames(salig.df) <-
  gsub(
    '^QuestionRT$'
    , 'T'
    , colnames(salig.df)
  )

# baselang to L
colnames(salig.df) <-
  gsub(
    '^baselang$'
    , 'L'
    , colnames(salig.df)
  )

# item number to I
colnames(salig.df) <-
  gsub(
    '^itemnum$'
    , 'I'
    , colnames(salig.df)
  )
salig.df %>%
  # create a variable named combination
  # for the discrete positions in the y axis
  mutate(
    combination = paste(L, currtrial)
  ) %>%
  # plot start
  ggplot(
    aes(
      y = combination
      , x = log(T)
      , fill = L
      , color = L
    )
  ) +
  # jitter of means of participants
  geom_jitter(
    data = . %>%
      group_by(
        S, combination, L
      ) %>%
      summarise(
        x = mean(log(T))
      )
    , aes(
      x = x
    )
    , width = 0
    , height = .06
    , alpha = .3
    , size = .8
    # rings
    , shape = 1
  ) +
  # shade of the observations
  geom_point(
    shape = 95
    , size = 40
    , alpha = 0.01
    , show.legend = F
    , color = 'gray'
  ) +
  # point interval of distribution
  stat_pointinterval(
    , fill = NA
    , .width = c(.5, .75, .95)
  ) +
  # density
  stat_slab(
    fill = NA
  ) +
  # line separating EN & ES
  geom_hline(
    yintercept = 2.93
```

```r
  , linetype = 'dashed'
  , color = 'lightgray'
) +
# theming
# labs
labs(
  x = 'Reading time (log-ms)'
  , fill = NULL
) +
# scales
scale_y_discrete(
  name = NULL
  , expand = c(0.0,0)
) +
scale_x_continuous(
  breaks = seq(5, 11, 1.5)
) +
# theme
theme(
  axis.line.y = element_blank()
  , axis.ticks.y = element_blank()
  , axis.text.y = element_blank()
  , legend.position = 'none'
) +
# colour
viridis::scale_fill_viridis(
  discrete = T
  , option = 'H'
) +
viridis::scale_color_viridis(
  discrete = T
  , option = 'H'
  , name = NULL
) +
# text of condition
geom_text(
  inherit.aes = F
  , data = . %>%
    group_by(
      combination, currtrial
    ) %>%
    summarise(
        T = median(log(T))
      )
    , aes(
      x = T
      , y = combination
      , label = currtrial
    )
    , color = 'black'
    , vjust = -1
    , size = 5
    , family = 'Carlito'
  ) +
  # language text
  geom_text(
    inherit.aes = F
    , data = . %>%
      mutate(
        y = ifelse(
          L == 'English'
          , 2
          , 4
        )
      )
    , aes(
      x = 5
      , y = y
      , label = L
    )
    , angle = 90
    , hjust = 0.5
    , color = 'gray'
    , size = 6
    , family = 'Carlito'
  )
salig.df$s1_l <- ifelse(
  salig.df$L == 'English'
  , -1
  , 1
)

salig.df$s.5_l <- ifelse(
  salig.df$L == 'English'
  , -.5
  , .5
)
fit.lang.sum.1 <- lmer(
  log(T) ~ s1_l + (1 | S) + (1 | I)
  , data = salig.df
)

fit.lang.sum.05 <- lmer(
  log(T) ~ s.5_l + (1 | S) + (1 | I)
  , data = salig.df
)
fit.lang.indicator <- lmer(
  log(T) ~ L + (1 | S) + (1 | I)
  , data = salig.df
)

# equivalent formula
fit.lang.indicator <- lmer(
  log(T) ~ 1 + L + (1 | S) + (1 | I)
  , data = salig.df
)
scoded.lang.coef.5 <- broom.mixed::tidy(
  fit.lang.sum.05
  , conf.int = T
)

scoded.lang.coef.5 <-
  scoded.lang.coef.5[
    scoded.lang.coef.5$effect == 'fixed'
    # cols inherited from the indicator model
    , cols[3:ncol(scoded.lang.coef.5)]
  ]

scoded.lang.coef.1 <- broom.mixed::tidy(
  fit.lang.sum.1
  , conf.int = T
)

scoded.lang.coef.1 <-
  scoded.lang.coef.1[
    scoded.lang.coef.1$effect == 'fixed'
    , cols[3:ncol(scoded.lang.coef.1)]
  ]
cmeans.scoded.indicator.5.emm <- data.frame(
  emmeans(
    fit.lang.sum.05
    , ~ s.5_l
  )
)

cmeans.scoded.indicator.1.emm <- data.frame(
  emmeans(
    fit.lang.sum.1
    , ~ s1_l
  )
)
cmeans.scoded.indicator.5.marg <- data.frame(
  predictions(
    fit.lang.sum.05
    , by = 's.5_l'
    , re.form = NA
  )
)

cmeans.scoded.indicator.1.marg <- data.frame(
  predictions(
    fit.lang.sum.1
    , by = 's1_l'
    , re.form = NA
  )
)
fit.lang.index <- lmer(
  log(T) ~ 0 + L + (1 | S) + (1 | I)
  , data = salig.df
)
# extract population-level estimates
# from the model
index.lang.coef <- broom.mixed::tidy(
  fit.lang.index
  , conf.int = T
)

index.lang.coef <- index.lang.coef[
  index.lang.coef$effect == 'fixed'
  ,
  grepl(
    'term|^est|^conf'
    , cols
  )
]

index.lang.coef %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    -starts_with('conf')
  ) %>%
  gt(
    , rowname_col = 'term'
  )
index.dif.emm <- data.frame(
  emmeans(
    fit.lang.index, pairwise ~ L
  )$contrasts
)
gmean.emmeans <- data.frame(
  emmeans(fit.lang.index, ~ 1)
)

gmean.emmeans %>%
  select(
    , emmean
    , asymp.LCL
    , asymp.UCL
  ) %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , asymp.LCL
      , ', '
      , asymp.UCL
      , ']'
    )
  ) %>%
  select(
    -starts_with('asymp')
  ) %>%
  gt()
gmean.marginaleffects <- data.frame(
  predictions(
    fit.lang.index
    , re.form = NA
    , variables = 'L'
    , by = T
  )
)

gmean.marginaleffects %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
```

```r
    )
  ) %>%
  select(
    , estimate
    , `95% CI`
  ) %>%
  gt()
# grateful to create a text
# with the packages used directly
# dependencies are not cited
# because some are creating a problem
# with R4.5
grateful::cite_packages(
  output = 'paragraph'
  , out.dir = ('.')
  , pkgs = 'All'
  , cite.tidyverse = F
  , omit = NULL # include grateful
  , include.RStudio = F
  , dependencies = F
)
# obtain session info
sessionInfo()
library(magrittr) # pipe operator
library(dplyr) # easy manipulation
library(tidyr)
library(ggplot2) # plot
library(ggdist) # plot distributions
library(geomtextpath) # text to plots
library(patchwork) # combine plots
library(viridis)

# set basic theme
theme_set(
  theme_classic( # theme
    base_size = 16 # size of non-graph
    , base_family = 'Carlito' # font
  )
)
library(gt)
library(gtExtras)
library(kableExtra)
library(lme4)
library(broom)
library(broom.mixed)
library(emmeans)
library(marginaleffects)
# load package
library(brms)
# auto-parallel
# use as many cores as chains
# if ncores <= nchains
options(mc.cores = parallel::detectCores())
# use threading (split chains)
rstan::rstan_options(threads_per_chain = 2)
# write stan code to temp file
rstan::rstan_options(auto_write = TRUE)
# use cmdstanr backend
# it's much faster
options(brms.backend = "cmdstanr")
# call it explicitly
# so that it's cited directly
# otherwise it is used under the hood
# but grateful does not pick it
library(cmdstanr)
# seed for models
# ensure exact replication
source('scripts/00-setup-packages.R')
# create vectors of the variables
language <- c('English', 'Spanish')
switching <- c('+ switch', '- switch')

# create grid combining the variables
design.grid <- expand.grid(language = language, switching = switching)

# create combinations of variables
design.grid$combination <- paste(
  design.grid$language
  , design.grid$switching
)

# create table in gt
design.tb <- design.grid %>%
  # pivot to wider
  # to have combinations in 2x2
  tidyr::pivot_wider(
    names_from = switching
    , values_from = combination
  )
source('scripts/00-setup-packages.R')
source('scripts/02-read-data.R')
source('scripts/02-read-data.R')
# fit-treatment-indicator
fit.lang.indicator <- lmer(
  log(T) ~ L + (1 | S) + (1 | I)
  , data = salig.df
)

# equivalent formula
fit.lang.indicator <- lmer(
  log(T) ~ 1 + L + (1 | S) + (1 | I)
  , data = salig.df
)
summary(fit.lang.indicator)
# extract population-level estimates
# from the model
indicator.lang.coef <- broom.mixed::tidy(
  fit.lang.indicator
  , conf.int = T
)

# the first two columns of the
# tidy output are irrelevant
# and I don't want std error and statistic
# obtain the colnames
# just to keep the relevant ones
cols <- colnames(indicator.lang.coef)

indicator.lang.coef <- indicator.lang.coef[
  indicator.lang.coef$effect == 'fixed'
  ,
  grepl(
    'term|^est|^conf'
    , cols
  )
]
```

```r
indicator.lang.coef %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    -starts_with('conf')
  ) %>%
  gt(
    , rowname_col = 'term'
  )
cmeans.emmeans <- data.frame(
  emmeans(fit.lang.indicator, ~ L)
)

cmeans.emmeans %>%
  select(
    L
    , emmean
    , asymp.LCL
    , asymp.UCL
  ) %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , asymp.LCL
      , ', '
      , asymp.UCL
      , ']'
    )
  ) %>%
  select(
    -starts_with('asymp')
  ) %>%
  gt(
    , rowname_col = 'L'
  )
cmeans.marginaleffects <- data.frame(
  predictions(
    fit.lang.indicator
    , variables = 'L'
    , by = 'L'
    , re.form = NA
  )
)

cmeans.marginaleffects %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    L
    , estimate
    , `95% CI`
  ) %>%
  gt(
    , rowname_col = 'L'
  )
```

```r
gmean.emmeans <- data.frame(
  emmeans(fit.lang.indicator, ~ 1)
)

gmean.emmeans %>%
  select(
    , emmean
    , asymp.LCL
    , asymp.UCL
  ) %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , asymp.LCL
      , ', '
      , asymp.UCL
      , ']'
    )
  ) %>%
  select(
    -starts_with('asymp')
  ) %>%
  gt()
gmean.marginaleffects <- data.frame(
  predictions(
    fit.lang.indicator
    # without the next 2 lines
    # it returns the marginal avg
    # with the weights in the dataset
    , variables = 'L'
    , by = T
    , re.form = NA
  )
)


gmean.marginaleffects %>%
  mutate_if(
    is.numeric
    , round
    , digits = 2
  ) %>%
  mutate(
    `95% CI` = paste0(
      '['
      , conf.low
      , ', '
      , conf.high
      , ']'
    )
  ) %>%
  select(
    , estimate
    , `95% CI`
  ) %>%
  gt()
source('scripts/02-read-data.R')
# change default contrast method
# to sum-coding
# R handles it under the hood
options(contrasts = c('contr.sum','contr.sum'))
source('scripts/02-read-data.R')
```