RSA-SvI: A Rational Speech-Act model for the pragmatic use of vague terms in natural language

Supplementary files

Alexandre Cremers

July 2022

This file contains all the code to reproduce the analysis of Cremers (2022). It was completed later than the paper, and there are some minor differences between the two (e.g., in random effects structures of some models), which are documented wherever they occur.

Pre-computing utility functions

Relative adjective 'tall'

Semantic assumptions (S0)

We assume that $P(\textsc{min}) = 0$ for now (otherwise the "not tall" alternative has an undefined utility).

The distribution of heights for adult males in the US population is Gaussian with a mean of $69.2in$ and a standard deviation of $2.66in$. We can assume that this is the prior. We now turn to the distribution of the thresholds θ and $theta'$ for 'tall' and 'very tall' respectively.

We assume that for any given participant, $\theta' \geq \theta$ holds so that the entailment from "very tall" to "tall" is valid. However, it is clear from existing data that the distribution of the two variables overlap. The only way to reconcile these two observations is to accept that θ and θ' cannot be independent variables. The simplest deterministic way to ensure this is to assume that $\theta' = \theta + \delta$ with $\delta \geq 0$. We further assume that $\theta \sim \mathcal{N}(\mu, \sigma)$ (this corresponds to first-order vagueness), and δ is exponentially distributed for a given participant (first-order vagueness), with parameter λ. We assume that δ and θ are independent (again, crucially, θ and θ' are *not* independent). A participant is therefore parametrized by the triple (μ, σ, λ). We will assume that in the population, μ is normally distributed, σ and λ log-normal. The scale parameters of these distributions correspond to second order vagueness. We further assume that (μ, σ, λ) can be correlated (we will use a hybrid multivariate normal and lognormal distribution, see Flechter & Zupanski 2006).

With the model in place, we can estimate its parameters using the data of Leffel et al.

For this, we need the "'S_0" model, i.e. the probability that messages are true given (μ, σ, λ).

$$S_0(\textsc{pos tall}|d) = P(\theta < d) = \Phi\left(\frac{d-\mu}{\sigma}\right)$$

$$S_0(\text{very tall}|d) = P(\theta + \delta < d) = \Phi\left(\frac{d-\mu}{\sigma}\right) - \mathrm{e}^{\frac{\lambda^2\sigma^2}{2} - \lambda(d-\mu)}\Phi\left(\frac{d-\mu-\lambda\sigma^2}{\sigma}\right)$$

We can also write down the probabilities for 'not very tall':

$$S_0(\text{not very tall}|d) = 1 - P(\theta + \delta < d) = \Phi^c\left(\frac{d-\mu}{\sigma}\right) + \mathrm{e}^{\frac{\lambda^2\sigma^2}{2} - \lambda(d-\mu)}\Phi\left(\frac{d-\mu-\lambda\sigma^2}{\sigma}\right)$$

$$S_0(\text{EXH}[\text{not very tall}]|d) = P(\theta < d \leq \theta + \delta) = \mathrm{e}^{\frac{\lambda^2\sigma^2}{2} - \lambda(d-\mu)}\Phi\left(\frac{d - \mu - \lambda\sigma^2}{\sigma}\right)$$

NB: Given a set of parameters (μ, σ, λ), we can directly write: $P(\theta < d \leq \theta + \delta) = P(\theta < d) - P(\theta + \delta < d)$.

```
Leffel_data <-
read_csv("https://raw.githubusercontent.com/lefft/not_very_adj/master/data/Expt1-data_cleaned_screened.c
show_col_types = FALSE)
```

The next block fits the model, but you can use the saved posterior parameters loaded in the next block.

```
# We remove participant 6, as they're a clear outlier and mess up the evaluation of sigma
Leffel_data %>%
  filter(subj_id == "subj06" & Pred %in% c("tall", "veryTall", "notTall", "notVeryTall"))
  %>%
  ggplot(aes(x = Unit, y = response, group = Repetition, col = factor(Repetition))) +
  facet_wrap(. ~ Pred) +
  geom_line()


Leffel_data_tall <- Leffel_data %>%
  mutate(subj_id = as.numeric(substr(subj_id, 5, 6))) %>%
  filter(Adj == "Tall" & Pred %in% c("tall", "veryTall") & subj_id != 6) %>%
  mutate(
    Unit = as.numeric(as.character(Unit)),
    Unit = (Unit - 70) / 4
  ) # For the stan model we normalize the scale of heights

Leffel_stan_data <- list(
  N = nrow(Leffel_data_tall),
  S = n_distinct(Leffel_data_tall$subj_id),
  adv = if_else(Leffel_data_tall$Pred == "veryTall", 1, 0),
  unit = Leffel_data_tall$Unit,
  subject = as.numeric(factor(Leffel_data_tall$subj_id)),
  y = Leffel_data_tall$response / 100
)

tall_init_function <- function() {
  list(
    m_mu = runif(1, -1, 1),
    m_sigma = runif(1, -1, 0.5),
    m_lambda = log(runif(1, 1.5, 3.5)),
    s_mu = runif(1, .2, 1),
    s_sigma = runif(1, 0.2, 1.2),
    s_lambda = runif(1, 0.2, 1),
    z_u = matrix(runif(3 * n_distinct(Leffel_data_tall$subj_id), -.25, .25), nrow = 3),
    eps = runif(1, .1, .4)
  )
}

Leffel_stan_model_tall <- cmdstan_model("Stan-models/affirmative_model_tall.stan")

fit_tall <- Leffel_stan_model_tall$sample(
  data = Leffel_stan_data,
```

```r
  init = tall_init_function,
  chains = 12,
  parallel_chains = 14,
  iter_warmup = 1000,
  iter_sampling = 5000,
  step_size = 0.03,
  adapt_delta = 0.9,
  max_treedepth = 15
)
Leffel_stan_fit_tall <- read_stan_csv(fit_tall$output_files())

## Optional graph if one wants to vizualize the model fit:
# tmp <- (apply(rstan::extract(Leffel_stan_fit_tall,pars=c("pred"))$pred,2,mean))
# Leffel_data_tall %>%
#   mutate(Prediction = exp(tmp)) %>%
#   group_by(subj_id,NormUnit,Pred) %>%
#   summarise(mean_resp = mean(response)/100,
#             prediction = mean(Prediction)) %>%
#   ggplot(aes(x=70+NormUnit/3,y=mean_resp,group=Pred,linetype=Pred))+
#   facet_wrap(.~subj_id)+
#   geom_line(col="blue")+
#   geom_line(aes(x=70+NormUnit/3,y=prediction),col="red")+
#   theme_bw()+
#   scale_x_continuous(name="height (in)")+
#   scale_y_continuous(name="response",labels=scales::percent)

# Extract the parameters for each participant:
fitted_mu_theta_tall <- apply(rstan::extract(Leffel_stan_fit_tall, pars = c("mu"))$mu, 2,
mean)
fitted_sigma_theta_tall <- apply(rstan::extract(Leffel_stan_fit_tall, pars =
c("sigma"))$sigma, 2, mean)
fitted_lambda_delta_tall <- apply(rstan::extract(Leffel_stan_fit_tall, pars =
c("lambda"))$lambda, 2, median)

# Save estimates parameters in csv files for later use:
write.csv(fitted_mu_theta_tall, "precomputed-parameters/fitted_mu_theta_tall.csv",
row.names = F)
write.csv(fitted_sigma_theta_tall, "precomputed-parameters/fitted_sigma_theta_tall.csv",
row.names = F)
write.csv(fitted_lambda_delta_tall,
"precomputed-parameters/fitted_lambda_delta_tall.csv", row.names = F)

# We approximate an average correlation matrix for practical purposes:
L_matrix <- matrix(summary(Leffel_stan_fit_tall, pars = c("L_u"))$summary[, 1], ncol = 3,
byrow = T)
M <- L_matrix %*% t(L_matrix)
M <- M / sqrt(diag(M) %*% t(diag(M)))
print(M)

# Mean parameters:
fitted_params <- summary(Leffel_stan_fit_tall, pars = c("m_mu", "m_sigma", "m_lambda",
"s_mu", "s_sigma", "s_lambda", "eps"))$summary[, 1]
```

```
# save average Stan fitted parameters (Omega):
emp_values_tall <- c(
  m_mu = as.numeric(70 + 4 * fitted_params[1]), # Here we convert back to the "natural"
  scale (in inches)
  s_mu = as.numeric(fitted_params[4]) * 4,
  m_sigma = as.numeric(fitted_params[2]) + log(4), # sigma is multiplied by 4, but
  m_sigma is on the log-scale
  s_sigma = as.numeric(fitted_params[5]), # on the log-scale, the normalization does not
  affect s_sigma
  m_lambda = as.numeric(fitted_params[3]) - log(4), # delta needs to be multiplied by 4,
  so lambda_delta divided by 4, hence m_lambda adjusted by -log(4).
  s_lambda = as.numeric(fitted_params[6]), # again, s_lambda is unaffected by the
  normalization
  rho_mu_sigma = M[2, 1],
  rho_mu_lambda = M[3, 1],
  rho_sigma_lambda = M[3, 2]
)

write.csv(emp_values_tall, "precomputed-parameters/fitted_values_tall.csv")
```

Read saved files instead of fitting the model:

```
tmp <- read.csv("precomputed-parameters/fitted_values_tall.csv")
emp_values_tall <- tmp[, 2]
names(emp_values_tall) <- tmp[, 1]
rm(tmp)
```

Model equations

We first need to characterize our L_0:

$$L_0(d|\text{tall}, \textsc{pos}) \propto f(d)\Phi\left(\frac{d-\mu}{\sigma}\right)$$

$$L_0(d|\text{not tall}, \textsc{pos}) \propto f(d)\Phi^c\left(\frac{d-\mu}{\sigma}\right)$$

$$L_0(d|\text{very tall}) \propto f(d)\left[\Phi\left(\frac{d-\mu}{\sigma}\right) - e^{\frac{\lambda^2\sigma^2}{2}-\lambda(d-\mu)}\Phi\left(\frac{d-\mu-\lambda\sigma^2}{\sigma}\right)\right]$$

Similarly:

$$L_0(d|\text{not very tall}, \textsc{lit}) = f(d)\frac{\Phi^c\left(\frac{d-\mu}{\sigma}\right) + e^{\frac{\lambda^2\sigma^2}{2}-\lambda(d-\mu)}\Phi\left(\frac{d-\mu-\lambda\sigma^2}{\sigma}\right)}{\int_d \varphi(d;m,s)\Phi^c(d;\mu,\sigma)\,\mathrm{d}d + e^{\lambda^2\frac{\sigma^2+s^2}{2}-\lambda(m-\mu)}\int_d \varphi(d;m-\lambda s^2,s)\Phi(d;\mu+\lambda\sigma^2,\sigma)\,\mathrm{d}d}$$

$$L_0(d|\text{not very tall}, \textsc{exh}) = \frac{f(d)e^{\frac{\lambda^2\sigma^2}{2}-\lambda(d-\mu)}\Phi\left(\frac{d-\mu-\lambda\sigma^2}{\sigma}\right)}{e^{\lambda^2\frac{\sigma^2+s^2}{2}-\lambda(m-\mu)}\int_d \varphi(d;m-\lambda s^2,s)\Phi(d;\mu+\lambda\sigma^2,\sigma)\,\mathrm{d}d}$$

Which can be further simplified:

$$L_0(d|\text{not very tall}, \textsc{exh}) = e^{\lambda m-\frac{\lambda^2 s^2}{2}}\frac{f(d)e^{-\lambda d}\Phi\left(\frac{d-\mu-\lambda\sigma^2}{\sigma}\right)}{\int_d \varphi(d;m-\lambda s^2,s)\Phi(d;\mu+\lambda\sigma^2,\sigma)\,\mathrm{d}d}$$

Quick note to possibly optimize computations, using integration by parts:

$$\int_{-\infty}^{+\infty} f(x; m, s)\,(1 - F(x; \mu, \sigma))\,\mathrm{d}x = \int_{-\infty}^{+\infty} F(x; m, s) f(x; \mu, \sigma)\,\mathrm{d}x$$

```
# Prior (taken from somewhere on the internet):
tall_prior_mean <- 69.2
tall_prior_sig <- 2.66
```

The utility functions have also been pre-computed for an array of degree values, so no need to run the next few blocks.

```
# Parameters for integrals:
int_sub <- 1000L
S1_tol <- 1e-4
L0_tol <- 1e-4

# NB: integrate is actually faster than hcubature and pcubature here
# The log1mexp function from copula is faster than the one from VGAM

logL0_tall_pos <- function(d, mu_1, sigma_1) {
  dnorm(d, tall_prior_mean, tall_prior_sig, log = T) + pnorm(d, mu_1, sigma_1, log.p = T)
  -
    log(integrate(function(x) {
      dnorm(x, tall_prior_mean, tall_prior_sig) * pnorm(x, mu_1, sigma_1)
    },
    50, 90,
    rel.tol = L0_tol, subdivisions = int_sub
    )$value)
}
logL0_not_tall <- function(d, mu_1, sigma_1) {
  dnorm(d, tall_prior_mean, tall_prior_sig, log = T) + pnorm(d, mu_1, sigma_1, lower.tail
  = F, log.p = T) - log(integrate(function(x) {
    dnorm(x, tall_prior_mean, tall_prior_sig) * pnorm(x, mu_1, sigma_1, lower.tail = F)
  }, 50, 90, rel.tol = L0_tol, subdivisions = int_sub)$value)
}

# Functions to optimize the computation of some exponentials
very_exponent <- function(x, ls) { # ls is lambda*sigma > 0, x is (d-mu)/sigma, unbound
  ls^2 / 2 - ls * x + pnorm(x - ls, log.p = T) - pnorm(x, log.p = T)
}
not_very_exponent <- function(x, ls) { # ls is lambda*sigma > 0, x is (d-mu)/sigma,
unbound
  ls^2 / 2 - ls * x + pnorm(x - ls, log.p = T) - pnorm(x, log.p = T, lower.tail = F)
}
not_very_exh_exponent <- function(x, ls) { # ls is lambda*sigma > 0, x is (d-mu)/sigma,
unbound
  ls^2 / 2 - ls * x + pnorm(x - ls, log.p = T)
}

logL0_very_tall <- function(d, mu_1, sigma_1, lambda_1) {
  dnorm(d, tall_prior_mean, tall_prior_sig, log = T) + pnorm(d, mu_1, sigma_1, log.p = T)
  + log1mexp(-very_exponent((d - mu_1) / sigma_1, sigma_1 * lambda_1)) - log(
    tryCatch(integrate(function(x) {
      exp(dnorm(x, tall_prior_mean, tall_prior_sig, log = T) + pnorm(x, mu_1, sigma_1,
      log.p = T) + log1mexp(-very_exponent((x - mu_1) / sigma_1, sigma_1 * lambda_1)))
```

```r
    }, 50, 90, rel.tol = L0_tol, subdivisions = int_sub)$value,
    error = function(e) {
      print(c(d, mu_1, sigma_1, lambda_1))
      NaN
    }
    )
  )
}

logL0_not_very_tall_lit <- function(d, mu_1, sigma_1, lambda_1) {
  dnorm(d, tall_prior_mean, tall_prior_sig, log = T) + pnorm(d, mu_1, sigma_1, log.p = T,
  lower.tail = F) + log1pexp(not_very_exponent((d - mu_1) / sigma_1, sigma_1 * lambda_1))
  - log(
    tryCatch(integrate(function(x) {
      exp(dnorm(x, tall_prior_mean, tall_prior_sig, log = T) + pnorm(x, mu_1, sigma_1,
      log.p = T, lower.tail = F) + log1pexp(not_very_exponent((x - mu_1) / sigma_1,
      sigma_1 * lambda_1)))
    }, 50, 90, rel.tol = L0_tol, subdivisions = int_sub)$value,
    error = function(e) {
      print(c(d, mu_1, sigma_1, lambda_1))
      NaN
    }
    )
  )
}
logL0_not_very_tall_exh <- function(d, mu_1, sigma_1, lambda_1) {
  dnorm(d, tall_prior_mean, tall_prior_sig, log = T) + not_very_exh_exponent((d - mu_1) /
  sigma_1, sigma_1 * lambda_1) - log(
    tryCatch(integrate(function(x) {
      exp(dnorm(x, tall_prior_mean, tall_prior_sig, log = T) + not_very_exh_exponent((x -
      mu_1) / sigma_1, sigma_1 * lambda_1))
    }, 50, 90, rel.tol = L0_tol, subdivisions = int_sub)$value,
    error = function(e) {
      print(c(d, mu_1, sigma_1, lambda_1))
      NaN
    }
    )
  )
}
```

We now turn to U_1 functions (without costs for the moment).

```r
# We need the joint pdf of mu/sigma and mu/sigma/lambda
# See Flechter & Zupanski (2006) for proofs
# For mu/sigma, it's a bit faster to define it by hand:
joint_pdf_ms <- function(mu, sigma, m_mu, m_sig, s_mu, s_sig, rho) {
  exp(
    -((((mu - m_mu) / s_mu)^2 - 2 * rho * (mu - m_mu) * (log(sigma) - m_sig) / (s_mu *
    s_sig) + ((log(sigma) - m_sig) / s_sig)^2) / (2 * (1 - rho^2))
  ) / (
    2 * pi * s_mu * s_sig * sigma * sqrt(1 - rho^2)
  )
}
# For mu/sigma/lambda... let's just keep it readable even if it might be further
optimized.
```

```r
joint_pdf_msl <- function(mu, sigma, lambda, m_m, m_s, m_l, M) {
  dmvnorm(cbind(mu, log(sigma), log(lambda)), c(m_m, m_s, m_l), M) / (sigma * lambda)
}

# Define U1 functions:
U1_null_tall <- function(d) {
  dnorm(d, tall_prior_mean, tall_prior_sig, log = T)
}
U1_not_tall <- function(d, m_mu, s_mu, m_sigma, s_sigma, rho) {
  integrand <- function(x) {
    joint_pdf_ms(x[1], x[2], m_mu, m_sigma, s_mu, s_sigma, rho) * logL0_not_tall(d, x[1],
    x[2])
  }
  hcubature(integrand,
    lowerLimit = c(qnorm(1e-6, m_mu, s_mu), qlnorm(1e-6, m_sigma, s_sigma)),
    upperLimit = c(
      qnorm(1e-6, m_mu, s_mu, lower.tail = F),
      qlnorm(1e-4, m_sigma, s_sigma, lower.tail = F)
    ),
    tol = S1_tol
  )$integral
}
U1_tall_pos <- function(d, m_mu, s_mu, m_sigma, s_sigma, rho) {
  integrand <- function(x) {
    joint_pdf_ms(x[1], x[2], m_mu, m_sigma, s_mu, s_sigma, rho) * logL0_tall_pos(d, x[1],
    x[2])
  }
  hcubature(integrand,
    lowerLimit = c(qnorm(1e-6, m_mu, s_mu), qlnorm(1e-6, m_sigma, s_sigma)),
    upperLimit = c(
      qnorm(1e-6, m_mu, s_mu, lower.tail = F),
      qlnorm(1e-4, m_sigma, s_sigma, lower.tail = F)
    ),
    tol = S1_tol
  )$integral
}
U1_very_tall <- function(d, m_mu, s_mu, m_sigma, s_sigma, m_lambda, s_lambda, M) {
  integrand <- function(x) {
    joint_pdf_msl(x[1], x[2], x[3], m_mu, m_sigma, m_lambda, M) * logL0_very_tall(d,
    x[1], x[2], x[3])
  }
  hcubature(integrand,
    lowerLimit = c(qnorm(1e-6, m_mu, s_mu), qlnorm(1e-6, m_sigma, s_sigma), qlnorm(5e-6,
    m_lambda, s_lambda)),
    upperLimit = c(
      qnorm(1e-6, m_mu, s_mu, lower.tail = F),
      qlnorm(1e-4, m_sigma, s_sigma, lower.tail = F),
      qlnorm(1e-6, m_lambda, s_lambda, lower.tail = F)
    ),
    tol = S1_tol
  )$integral
}
U1_not_very_tall_lit <- function(d, m_mu, s_mu, m_sigma, s_sigma, m_lambda, s_lambda, M)
{
```

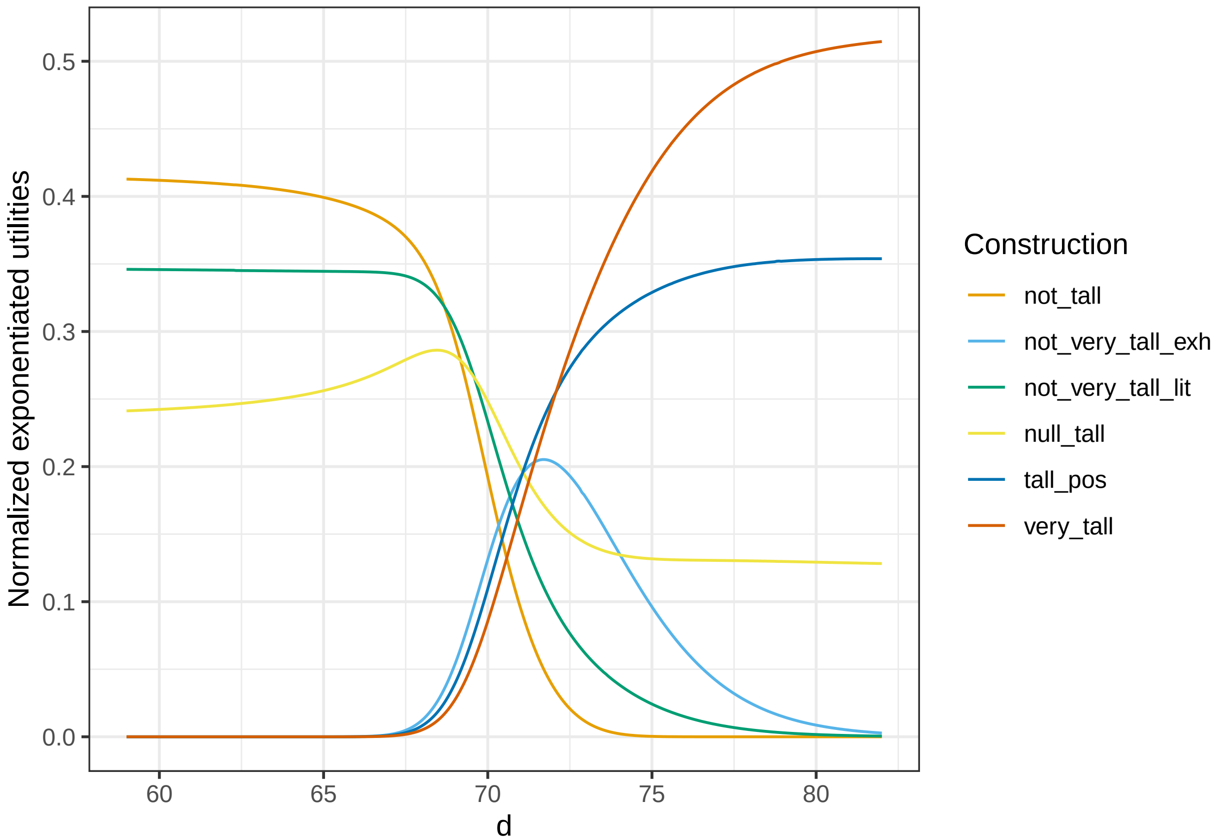
```r
  integrand <- function(x) {
    joint_pdf_msl(x[1], x[2], x[3], m_mu, m_sigma, m_lambda, M) *
    logL0_not_very_tall_lit(d, x[1], x[2], x[3])
  }
  hcubature(integrand,
    lowerLimit = c(qnorm(1e-6, m_mu, s_mu), qlnorm(1e-6, m_sigma, s_sigma), qlnorm(5e-6,
    m_lambda, s_lambda)),
    upperLimit = c(
      qnorm(1e-6, m_mu, s_mu, lower.tail = F),
      qlnorm(1e-4, m_sigma, s_sigma, lower.tail = F),
      qlnorm(1e-6, m_lambda, s_lambda, lower.tail = F)
    ),
    tol = S1_tol
  )$integral
}
U1_not_very_tall_exh <- function(d, m_mu, s_mu, m_sigma, s_sigma, m_lambda, s_lambda, M)
{
  integrand <- function(x) {
    joint_pdf_msl(x[1], x[2], x[3], m_mu, m_sigma, m_lambda, M) *
    logL0_not_very_tall_exh(d, x[1], x[2], x[3])
  }
  hcubature(integrand,
    lowerLimit = c(qnorm(1e-6, m_mu, s_mu), qlnorm(1e-6, m_sigma, s_sigma), qlnorm(5e-6,
    m_lambda, s_lambda)),
    upperLimit = c(
      qnorm(1e-6, m_mu, s_mu, lower.tail = F),
      qlnorm(1e-4, m_sigma, s_sigma, lower.tail = F),
      qlnorm(1e-6, m_lambda, s_lambda, lower.tail = F)
    ),
    tol = S1_tol
  )$integral
}


m_mu <- emp_values_tall["m_mu"]
s_mu <- emp_values_tall["s_mu"]
m_sigma <- emp_values_tall["m_sigma"]
s_sigma <- emp_values_tall["s_sigma"]
m_lambda <- emp_values_tall["m_lambda"]
s_lambda <- emp_values_tall["s_lambda"]
rho_mu_sigma <- emp_values_tall["rho_mu_sigma"]
rho_mu_lambda <- emp_values_tall["rho_mu_lambda"]
rho_sigma_lambda <- emp_values_tall["rho_sigma_lambda"]

M_msl <- matrix(c(
  s_mu^2, rho_mu_sigma * s_mu * s_sigma, rho_mu_lambda * s_mu * s_lambda,
  rho_mu_sigma * s_mu * s_sigma, s_sigma^2, rho_sigma_lambda * s_sigma * s_lambda,
  rho_mu_lambda * s_mu * s_lambda, rho_sigma_lambda * s_sigma * s_lambda, s_lambda^2
), nrow = 3)
```

```r
height_samples <- seq(59, 82, length.out = 346) # step size: 1/15
```

```r
# Takes 10-15min:
t <- Sys.time()
fixed_U1_tall_pos <- future_sapply(height_samples, function(d) U1_tall_pos(d, m_mu, s_mu,
m_sigma, s_sigma, rho_mu_sigma), future.seed = T)
fixed_U1_not_tall <- future_sapply(height_samples, function(d) U1_not_tall(d, m_mu, s_mu,
m_sigma, s_sigma, rho_mu_sigma), future.seed = T)
fixed_U1_very_tall <- future_sapply(height_samples, function(d) U1_very_tall(d, m_mu,
s_mu, m_sigma, s_sigma, m_lambda, s_lambda, M_msl), future.seed = T)
fixed_U1_notvery_tall_exh <- future_sapply(height_samples, function(d)
U1_not_very_tall_exh(d, m_mu, s_mu, m_sigma, s_sigma, m_lambda, s_lambda, M_msl),
future.seed = T)
fixed_U1_notvery_tall_lit <- future_sapply(height_samples, function(d)
U1_not_very_tall_lit(d, m_mu, s_mu, m_sigma, s_sigma, m_lambda, s_lambda, M_msl),
future.seed = T)
print(Sys.time() - t)

fixed_U1_tall <- tibble(
  d = height_samples,
  null_tall = U1_null_tall(height_samples),
  tall_pos = fixed_U1_tall_pos,
  not_tall = fixed_U1_not_tall,
  very_tall = fixed_U1_very_tall,
  not_very_tall_lit = fixed_U1_notvery_tall_lit,
  not_very_tall_exh = fixed_U1_notvery_tall_exh
)
write_csv(fixed_U1_tall, "precomputed-parameters/U1_tall.csv")
```

Here we load pre-computed utilities (if the previous blocks weren't run), and plot the normalized exponential utilities (i.e., S1 with rationality set to 1 and all costs set to 0).

```r
fixed_U1_tall <- read_csv("precomputed-parameters/U1_tall.csv", show_col_types = FALSE)

# This graphs indicates what S1 would look like with lambda=1 and all costs at 0:
fixed_U1_tall %>%
  mutate_at(2:7, exp) %>%
  mutate(row_sums = rowSums(select(., 2:7))) %>%
  mutate_at(2:7, ~ . / row_sums) %>%
  select(-row_sums) %>%
  pivot_longer(
    cols = -d,
    names_to = "Construction",
    values_to = "Utility"
  ) %>%
  ggplot(aes(x = d, y = Utility, col = Construction, group = Construction)) +
  geom_line() +
  theme_bw() +
  scale_color_manual(values = cbbPalette) +
  ylab("Normalized exponentiated utilities")
```



Minimum standard adjective 'late'

Semantic assumptions (S0)

We assume that θ follows an exponential distribution with parameter λ_θ. Again, we want to maintain the entailment from 'very late' to 'late' (i.e. $\theta' \geq \theta$), so we assume that $\theta' = \theta + \delta$, with δ independent from θ and following another exponential distribution parametrized by λ_δ.

For 'very late', this yields:

$$P(\theta' < d) = \left| \begin{array}{ll} 1 - \dfrac{\lambda_\delta \mathrm{e}^{-\lambda_\theta d} - \lambda_\theta \mathrm{e}^{-\lambda_\delta d}}{\lambda_\delta - \lambda_\theta} & \text{if } \lambda_\delta \neq \lambda_\theta \\ 1 - (1 + \lambda d)\mathrm{e}^{-\lambda d} & \text{if } \lambda_\delta = \lambda_\theta \end{array} \right.$$

We consider that the acceptability of 'late' as measured in the experiments of Leffel et al. already takes into account the possibility to use MIN, so we will fit the parameter $\zeta = P(\text{MIN})$:

$$P([\![\text{late}]\!] = 1|d) = \zeta 1_{d>0} + (1 - \zeta)P(\theta < d)$$

A participant is then parametrized by a triplet $(\zeta, \lambda_\theta, \lambda_\delta)$. We assume that these parameters are distributed as follows in the population:

- ζ follows a logistic-normal distribution with parameters m_ζ, s_ζ
- λ_θ is log-normal with parameters $-m_\theta$ and s_θ (because the mean of θ is λ_θ^{-1}, so m_θ is the mean of $\log \theta$)
- λ_δ is log-normal with parameters $-m_\delta$ and s_δ

We further allow possible correlations between $\zeta, \lambda_\theta, \lambda_\delta$.

```r
Leffel_data <-
read_csv("https://raw.githubusercontent.com/lefft/not_very_adj/master/data/Expt1-data_cleaned_screened.c
show_col_types = FALSE)

Leffel_data_late <- Leffel_data %>%
  mutate(subj_id = as.numeric(substr(subj_id, 5, 6))) %>%
  filter(Adj == "Late" & Pred %in% c("late", "veryLate")) %>%
  mutate(Unit = NormUnit / 16) # Divide to scale around an sd of ~1

Leffel_stan_data <- list(
  N = nrow(Leffel_data_late),
  S = n_distinct(Leffel_data_late$subj_id),
  adv = if_else(Leffel_data_late$Pred == "veryLate", 1, 0),
  unit = Leffel_data_late$Unit,
  subject = as.numeric(factor(Leffel_data_late$subj_id)),
  y = Leffel_data_late$response / 100
)

Leffel_stan_model_late <- cmdstan_model("Stan-models/affirmative_model_late.stan")

# Takes just a few minutes:
fit_late <- Leffel_stan_model_late$sample(
  data = Leffel_stan_data,
  chains = 12,
  parallel_chains = 14,
  iter_warmup = 1000,
  iter_sampling = 5000,
  step_size = 0.1,
  adapt_delta = 0.95,
  max_treedepth = 15
)
Leffel_stan_fit_late <- read_stan_csv(fit_late$output_files())

# We approximate an average correlation matrix for practical purposes:
L_matrix <- matrix(summary(Leffel_stan_fit_late, pars = c("L_u"))$summary[, 1], ncol = 3,
byrow = T)
M <- L_matrix %*% t(L_matrix)
M <- M / sqrt(diag(M) %*% t(diag(M)))
print(M)

# Mean parameters:
fitted_params_late <- summary(Leffel_stan_fit_late, pars = c("m_zeta", "m_lambda_theta",
"m_lambda_delta", "sigma_u", "eps"))$summary[, 1]

# Save by-participant parameters for fitting the evaluation model:
# For lambda_delta we take the median because some posterior distributions are very
skewed
fitted_lambda_theta <- apply(rstan::extract(Leffel_stan_fit_late, pars =
c("lambda_theta"))$lambda_theta, 2, mean)
fitted_lambda_delta <- apply(rstan::extract(Leffel_stan_fit_late, pars =
c("lambda_delta"))$lambda_delta, 2, median)
fitted_p_min <- apply(rstan::extract(Leffel_stan_fit_late, pars = c("zeta"))$zeta, 2,
mean)
```

```r
# save average Stan fitted parameters (Omega):
# theta and delta must be multiplied by 16 to get back to the original scale (in min), so the corresponding lambda's need a -log(16) correction
emp_values_late <- c(
  m_zeta = as.numeric(fitted_params_late[1]),
  s_zeta = as.numeric(fitted_params_late[4]),
  m_lambda_theta = as.numeric(fitted_params_late[2]) - log(16),
  s_lambda_theta = as.numeric(fitted_params_late[5]),
  m_lambda_delta = as.numeric(fitted_params_late[3]) - log(16),
  s_lambda_delta = as.numeric(fitted_params_late[6]),
  rho_zeta_theta = M[2, 1],
  rho_zeta_delta = M[3, 1],
  rho_theta_delta = M[3, 2]
)

write.csv(emp_values_late, "precomputed-parameters/fitted_values_late.csv")
write.csv(fitted_lambda_theta, "precomputed-parameters/fitted_lambda_theta_late.csv", row.names = F)
write.csv(fitted_lambda_delta, "precomputed-parameters/fitted_lambda_delta_late.csv", row.names = F)
write.csv(fitted_p_min, "precomputed-parameters/fitted_p_min_late.csv", row.names = F)
```

Just read values saved earlier:

```r
tmp <- read.csv("precomputed-parameters/fitted_values_late.csv")
emp_values_late <- tmp[, 2]
names(emp_values_late) <- tmp[, 1]
rm(tmp)
```

Model equations

We will define L_0 and U_1 for all possible parses, although we will end up considering models which may not use all these available parses. The full list of parses includes: * MIN and POS parses for the bare adjective and its negation. * Three parses for 'not very late': literal, exhaustive with negation of 'MIN not late' `($\textsc{exh}_\textsc{min}$'' parse) and exhaustive with negation of '\textsc{pos} not late'` (EXH_{POS}'' parse, which entails the EXH_{MIN} parse since $\theta \geq 0$).

We assume a normal prior centered on 0 with sd 10min (arbitrary, but doesn't make much difference anyway).

```r
late_prior_mean <- 0 # this is mu_d
late_prior_sig <- 10 # this is sigma_d
```

The L_0 speaker is now parametrized by a pair $(\lambda_\theta, \lambda_\delta)$, a choice between MIN and POS for the bare adjective, and a choice to exhaustify or not for the "not very late" utterance. While we have measured ζ, we are not going to use it at this point, since the RSA-SvI model generates predictions for the posterior probability of all parses, including MIN.

The L_0 listener quickly becomes very complicated unfortunately.

$$L_0(d|\text{late}, \text{MIN}) = \frac{\varphi(d)}{\Phi^c(-\mu_d/\sigma_d)} \quad \text{if } d > 0$$

$$L_0(d|\text{late}, \textsc{pos}, \lambda_\theta) = \frac{\varphi(d)(1-\mathrm{e}^{-\lambda_\theta d})}{\Phi^c(-\mu_d/\sigma_d) - \mathrm{e}^{\frac{\lambda_\theta^2\sigma_d^2}{2}-\lambda_\theta\mu_d}\Phi^c(\lambda_\theta\sigma_d - \mu_d/\sigma_d)} \quad \text{if } d > 0$$

$$L_0(d|\text{not late}, \textsc{min}) = \frac{\varphi(d)}{\Phi(-\mu_d/\sigma_d)} \quad \text{if } d \leq 0$$

$$L_0(d|\text{not late}, \textsc{pos}, \lambda_\theta) = \frac{\varphi(d)\min(1, \mathrm{e}^{-\lambda_\theta d})}{\Phi^c(-\mu_d/\sigma_d) + \mathrm{e}^{\frac{\lambda_\theta^2\sigma_d^2}{2}-\lambda_\theta\mu_d}\Phi^c(\lambda_\theta\sigma_d - \mu_d/\sigma_d)}$$

If $d > 0$ and $\lambda_\delta \neq \lambda_\theta$:

$$L_0(d|\text{very late}, \lambda_\theta, \lambda_\delta) = \frac{\varphi(d)\left(\lambda_\delta(1-\mathrm{e}^{-\lambda_\theta d}) - \lambda_\theta(1-\mathrm{e}^{-\lambda_\delta d})\right)}{(\lambda_\delta-\lambda_\theta)\Phi^c(-\mu_d/\sigma_d) - \lambda_\delta\mathrm{e}^{\frac{\lambda_\theta^2\sigma_d^2}{2}-\lambda_\theta\mu_d}\Phi^c(\lambda_\theta\sigma_d - \mu_d/\sigma_d) + \lambda_\theta\mathrm{e}^{\frac{\lambda_\delta^2\sigma_d^2}{2}-\lambda_\delta\mu_d}\Phi^c(\lambda_\delta\sigma_d - \mu_d/\sigma_d)}$$

Note: in case $\lambda_\delta = \lambda_\theta = \lambda$, we have $P(\theta' < d) = 1 - (1+\lambda d)\mathrm{e}^{-\lambda d}$.

If $d > 0$ and $\lambda_\delta \neq \lambda_\theta$:

$$L_0(d|\text{not very late}, \lambda_\theta, \lambda_\delta, \textsc{lit}) = \frac{\varphi(d)\,\text{ifelse}\left(d<0, \lambda_\delta-\lambda_\theta, \left(\lambda_\delta\mathrm{e}^{-\lambda_\theta d} - \lambda_\theta\mathrm{e}^{-\lambda_\delta d}\right)\right)}{(\lambda_\delta-\lambda_\theta)\Phi^c(-\mu_d/\sigma_d) + \lambda_\delta\mathrm{e}^{\frac{\lambda_\theta^2\sigma_d^2}{2}-\lambda_\theta\mu_d}\Phi^c(\lambda_\theta\sigma_d - \mu_d/\sigma_d) - \lambda_\theta\mathrm{e}^{\frac{\lambda_\delta^2\sigma_d^2}{2}-\lambda_\delta\mu_d}\Phi^c(\lambda_\delta\sigma_d - \mu_d/\sigma_d)}$$

$$L_0(d|\text{not very late}, \lambda_\theta, \lambda_\delta, \textsc{exh}_{\textsc{pos}}) = \frac{\varphi(d)\left(\mathrm{e}^{-\lambda_\theta d} - \mathrm{e}^{-\lambda_\delta d}\right)}{\mathrm{e}^{\frac{\lambda_\theta^2\sigma_d^2}{2}-\lambda_\theta\mu_d}\Phi^c(\lambda_\theta\sigma_d - \mu_d/\sigma_d) - \mathrm{e}^{\frac{\lambda_\delta^2\sigma_d^2}{2}-\lambda_\delta\mu_d}\Phi^c(\lambda_\delta\sigma_d - \mu_d/\sigma_d)} \quad \text{if } d > 0 \text{ and } \lambda_\delta \neq \lambda_\theta$$

In case $\lambda_\delta = \lambda_\theta = \lambda$ (but it turns out we don't really need this):

$$L_0(d|\text{not very late}, \lambda, \lambda, \textsc{exh}_{\textsc{pos}}) = \frac{\varphi(d)h\mathrm{e}^{-\lambda d}}{\mathrm{e}^{\frac{\lambda^2\sigma_d^2}{2}-\lambda\mu_d}\left[\mu\Phi^c(\lambda\sigma_d - \frac{\mu_d}{\sigma_d}) + \frac{\sigma_d}{\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}(\lambda\sigma_d-\mu_d/\sigma_d)^2}\right]} \quad \text{if } d > 0$$

$$L_0(d|\text{not very late}, \lambda_\theta, \lambda_\delta, \textsc{exh}_{\textsc{min}}) = \frac{\varphi(d)\left(\lambda_\delta\mathrm{e}^{-\lambda_\theta d} - \lambda_\theta\mathrm{e}^{-\lambda_\delta d}\right)}{\lambda_\delta\mathrm{e}^{\frac{\lambda_\theta^2\sigma_d^2}{2}-\lambda_\theta\mu_d}\Phi^c(\lambda_\theta\sigma_d - \mu_d/\sigma_d) - \lambda_\theta\mathrm{e}^{\frac{\lambda_\delta^2\sigma_d^2}{2}-\lambda_\delta\mu_d}\Phi^c(\lambda_\delta\sigma_d - \mu_d/\sigma_d)} \quad \text{if } d > 0 \text{ and } \lambda_\delta \neq \lambda_\theta$$

We can now move to implementing all these L_0 functions. The most efficient implementation is to group together all the exponential terms (including the one coming from $\varphi(d)$), so we directly look at $\log L_0$. Again, you can skip the following blocks, as the utilities have already been pre-computed.

```
logL0_late_min <- function(d) {
  if_else(d <= 0, -Inf, dnorm(d, late_prior_mean, late_prior_sig, log = T) -
  pnorm(-late_prior_mean / late_prior_sig, log.p = T))
}
logL0_late_pos <- function(d, l_theta) {
  if_else(d <= 0, -Inf,
    dnorm(d, late_prior_mean, late_prior_sig, log = T) +
      log1p(-exp(-abs(l_theta * d))) - log(pnorm(-late_prior_mean / late_prior_sig,
      lower.tail = F) -
        exp(pnorm(l_theta * late_prior_sig - late_prior_mean / late_prior_sig, lower.tail
        = F, log.p = T) +
          l_theta^2 * late_prior_sig^2 / 2 - l_theta * late_prior_mean))
  )
}
```

```r
logL0_not_late_min <- function(d) {
  if_else(d > 0, -Inf, dnorm(d, late_prior_mean, late_prior_sig, log = T) -
  pnorm(-late_prior_mean / late_prior_sig, lower.tail = F, log.p = T))
}
logL0_not_late_pos <- function(d, l_theta) {
  dnorm(d, late_prior_mean, late_prior_sig, log = T) + pmin(0, -l_theta * d) -
    pnorm(-late_prior_mean / late_prior_sig, lower.tail = F, log.p = T) -
    log1p(exp(pnorm(l_theta * late_prior_sig - late_prior_mean / late_prior_sig,
    lower.tail = F, log.p = T) +
      l_theta^2 * late_prior_sig^2 / 2 - l_theta * late_prior_mean -
pnorm(-late_prior_mean / late_prior_sig, lower.tail = F, log.p = T)))
}

# Let's start optimizing!
# First, we need to take into account sign(l_delta-l_theta), as the signs of the
num/denom depend on it
# Second, we need to avoid the product pnorm*exp, as the former goes to 0 and the latter
diverges with lambda (the product actually goes to 0)
# Third, if we want to integrate on the whole R^2, we might need a special case for equal
lambdas (alternatively, it's possible to integrate on all lambda_delta but only limited
lambda_theta). But that doesn't seem necessary fortunately.
logL0_very_late <- function(d, l_theta, l_delta) {
  if_else(d <= 0, -Inf,
    dnorm(d, late_prior_mean, late_prior_sig, log = T) +
      log(-sign(l_delta - l_theta) * l_delta * (expm1(-l_theta * d)) + sign(l_delta -
      l_theta) * l_theta * expm1(-l_delta * d)) -
      log(abs(l_delta - l_theta) * pnorm(-late_prior_mean / late_prior_sig, lower.tail =
      F) -
        sign(l_delta - l_theta) * l_delta * exp(pnorm(l_theta * late_prior_sig -
        late_prior_mean / late_prior_sig, lower.tail = F, log.p = T) +
          l_theta^2 * late_prior_sig^2 / 2 - l_theta * late_prior_mean) +
        sign(l_delta - l_theta) * l_theta * exp(pnorm(l_delta * late_prior_sig -
        late_prior_mean / late_prior_sig, lower.tail = F, log.p = T) +
          l_delta^2 * late_prior_sig^2 / 2 - l_delta * late_prior_mean))
  )
}

logL0_not_very_late_lit <- function(d, l_theta, l_delta) {
  dnorm(d, late_prior_mean, late_prior_sig, log = T) +
    if_else(d < 0, log(abs(l_delta - l_theta)), log(abs(l_delta * (exp(-l_theta * d)) -
    l_theta * exp(-l_delta * d)))) -
    log(abs(l_delta - l_theta) * pnorm(-late_prior_mean / late_prior_sig, lower.tail = F)
    +
      sign(l_delta - l_theta) * l_delta * exp(pnorm(l_theta * late_prior_sig -
      late_prior_mean / late_prior_sig, lower.tail = F, log.p = T) +
        l_theta^2 * late_prior_sig^2 / 2 - l_theta * late_prior_mean) -
      sign(l_delta - l_theta) * l_theta * exp(pnorm(l_delta * late_prior_sig -
      late_prior_mean / late_prior_sig, lower.tail = F, log.p = T) +
        l_delta^2 * late_prior_sig^2 / 2 - l_delta * late_prior_mean))
}
logL0_not_very_late_exhpos <- function(d, l_theta, l_delta) {
  if_else(d <= 0, -Inf,
    dnorm(d, late_prior_mean, late_prior_sig, log = T) -
```

```
      pmin(l_theta, l_delta) * d + log1mexp(abs((l_theta - l_delta) * d)) -
      pnorm(pmin(l_theta, l_delta) * late_prior_sig - late_prior_mean / late_prior_sig,
      lower.tail = F, log.p = T) - pmin(l_theta, l_delta)^2 * late_prior_sig^2 / 2 +
      pmin(l_theta, l_delta) * late_prior_mean -
      log1mexp(abs(pnorm(l_delta * late_prior_sig - late_prior_mean / late_prior_sig,
      lower.tail = F, log.p = T) + l_delta^2 * late_prior_sig^2 / 2 - l_delta *
      late_prior_mean - pnorm(l_theta * late_prior_sig - late_prior_mean /
      late_prior_sig, lower.tail = F, log.p = T) - l_theta^2 * late_prior_sig^2 / 2 +
      l_theta * late_prior_mean))
  )
}

logL0_not_very_late_exhmin <- function(d, l_theta, l_delta) {
  l1 <- pmin(l_theta, l_delta)
  l2 <- pmax(l_theta, l_delta)
  if_else(d <= 0, -Inf,
    dnorm(d, late_prior_mean, late_prior_sig, log = T) -
      l1 * (d - late_prior_mean) + log1mexp((l2 - l1) * d + log(l2) - log(l1)) -
      (l1 * late_prior_sig)^2 / 2 - pnorm(l1 * late_prior_sig - late_prior_mean /
late_prior_sig, lower.tail = F, log.p = T) -
      log1mexp(log(l2) - log(l1) + (l2 - l1) * (late_prior_mean - (l1 + l2) *
      late_prior_sig^2 / 2) +
        pnorm(l1 * late_prior_sig - late_prior_mean / late_prior_sig, lower.tail = F,
        log.p = T) -
        pnorm(l2 * late_prior_sig - late_prior_mean / late_prior_sig, lower.tail = F,
        log.p = T))
  )
}
```

We can now define the utility functions.

```
m_lambda_theta <- emp_values_late[3]
s_lambda_theta <- emp_values_late[4]
m_lambda_delta <- emp_values_late[5]
s_lambda_delta <- emp_values_late[6]
rho_theta_delta <- emp_values_late[9]
# For correlated delta/theta:
M_theta_delta <- c(m_lambda_theta, m_lambda_delta)
S_theta_delta <- matrix(c(s_lambda_theta^2, s_lambda_theta * s_lambda_delta *
rho_theta_delta, s_lambda_theta * s_lambda_delta * rho_theta_delta, s_lambda_delta^2),
ncol = 2)

U1_null_late <- function(d) {
  dnorm(d, late_prior_mean, late_prior_sig, log = T)
}
U1_late_min <- function(d) {
  logL0_late_min(d)
}
U1_late_pos <- function(d, m_lambda_theta, s_lambda_theta) {
  integrand <- function(x) {
    dlnorm(x, m_lambda_theta, s_lambda_theta) * logL0_late_pos(d, x)
  }
  hcubature(integrand,
    lowerLimit = qlnorm(1e-6, m_lambda_theta, s_lambda_theta),
```

```
    upperLimit = qlnorm(1e-6, m_lambda_theta, s_lambda_theta, lower.tail = F),
    tol = S1_tol
  )$integral
}
U1_very_late <- function(d, M_theta_delta, S_theta_delta) {
  if (d <= 0) {
    return(-Inf)
  }
  integrand <- function(x) {
    dlnorm.rplus(x, M_theta_delta, S_theta_delta) * logL0_very_late(d, x[1], x[2])
  }
  hcubature(integrand,
    lowerLimit = c(0, 0),
    upperLimit = c(qlnorm(1e-6, m_lambda_theta, s_lambda_theta, lower.tail = F), Inf),
    tol = S1_tol
  )$integral
}

U1_not_late_min <- function(d) {
  logL0_not_late_min(d)
}
U1_not_late_pos <- function(d, m_lambda_theta, s_lambda_theta) {
  integrand <- function(x) {
    dlnorm(x, m_lambda_theta, s_lambda_theta) * logL0_not_late_pos(d, x)
  }
  hcubature(integrand,
    lowerLimit = qlnorm(1e-6, m_lambda_theta, s_lambda_theta),
    upperLimit = qlnorm(1e-6, m_lambda_theta, s_lambda_theta, lower.tail = F),
    tol = S1_tol
  )$integral
}
U1_not_very_late_lit <- function(d, M_theta_delta, S_theta_delta) {
  integrand <- function(x) {
    dlnorm.rplus(x, M_theta_delta, S_theta_delta) * logL0_not_very_late_lit(d, x[1],
    x[2])
  }
  hcubature(integrand,
    lowerLimit = c(0, 0),
    upperLimit = c(qlnorm(1e-6, m_lambda_theta, s_lambda_theta, lower.tail = F), Inf),
    tol = S1_tol
  )$integral
}
U1_not_very_late_exhpos <- function(d, M_theta_delta, S_theta_delta) {
  if (d <= 0) {
    return(-Inf)
  }
  integrand <- function(x) {
    dlnorm.rplus(x, M_theta_delta, S_theta_delta) * logL0_not_very_late_exhpos(d, x[1],
    x[2])
  }
  hcubature(integrand,
    lowerLimit = c(0, 0),
    upperLimit = c(qlnorm(1e-6, m_lambda_theta, s_lambda_theta, lower.tail = F), Inf),
```

```
    tol = S1_tol
  )$integral
}
U1_not_very_late_exhmin <- function(d, M_theta_delta, S_theta_delta) {
  if (d <= 0) {
    return(-Inf)
  }
  integrand <- function(x) {
    dlnorm.rplus(x, M_theta_delta, S_theta_delta) * logL0_not_very_late_exhmin(d, x[1],
    x[2])
  }
  hcubature(integrand,
    lowerLimit = c(0, 0),
    upperLimit = c(qlnorm(1e-6, m_lambda_theta, s_lambda_theta, lower.tail = F), Inf),
    tol = S1_tol
  )$integral
}
```

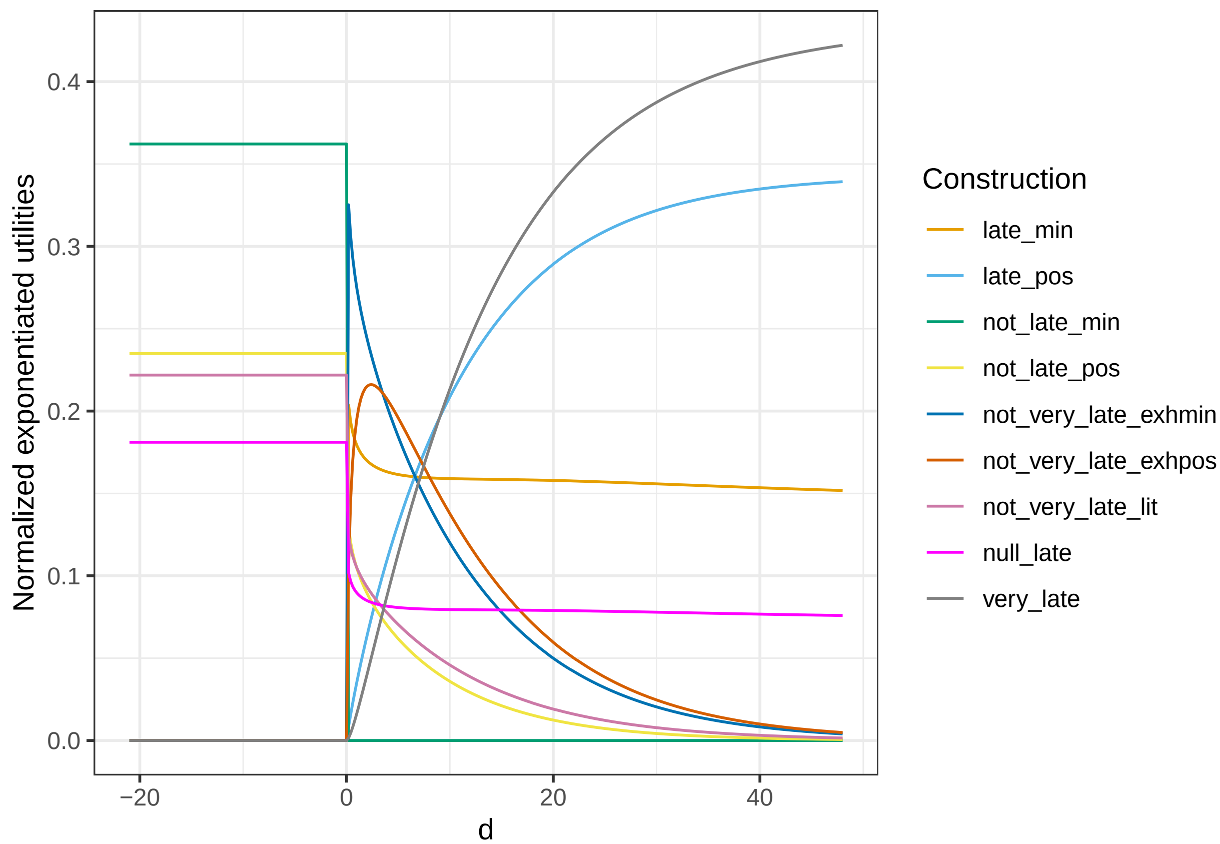
```
time_samples <- seq(-21, 48, by = .2) # make sure it's length 3n+1, but any 1/k fraction
works for the 'by' parameter.
```

```
# Needs higher precision than for 'tall', otherwise weird artifacts show up.
S1_tol <- 2e-6
t <- Sys.time()
fixed_U1_late_min <- U1_late_min(time_samples)
fixed_U1_not_late_min <- U1_not_late_min(time_samples)
fixed_U1_late_pos <- future_sapply(time_samples, function(d) U1_late_pos(d,
m_lambda_theta, s_lambda_theta), future.seed = T)
fixed_U1_not_late_pos <- future_sapply(time_samples, function(d) U1_not_late_pos(d,
m_lambda_theta, s_lambda_theta), future.seed = T)
fixed_U1_very_late <- future_sapply(time_samples, function(d) U1_very_late(d,
M_theta_delta, S_theta_delta), future.seed = T)
fixed_U1_not_very_late_lit <- future_sapply(time_samples, function(d)
U1_not_very_late_lit(d, M_theta_delta, S_theta_delta), future.seed = T)
fixed_U1_not_very_late_exhpos <- future_sapply(time_samples, function(d)
U1_not_very_late_exhpos(d, M_theta_delta, S_theta_delta), future.seed = T)
fixed_U1_not_very_late_exhmin <- future_sapply(time_samples, function(d)
U1_not_very_late_exhmin(d, M_theta_delta, S_theta_delta), future.seed = T)
print(Sys.time() - t)


fixed_U1_late <- data.frame(
  d = time_samples,
  null_late = U1_null_late(time_samples),
  late_min = fixed_U1_late_min,
  not_late_min = fixed_U1_not_late_min,
  late_pos = fixed_U1_late_pos,
  not_late_pos = fixed_U1_not_late_pos,
  very_late = fixed_U1_very_late,
  not_very_late_lit = fixed_U1_not_very_late_lit,
  not_very_late_exhpos = fixed_U1_not_very_late_exhpos,
  not_very_late_exhmin = fixed_U1_not_very_late_exhmin
)
```

```
write_csv(fixed_U1_late, "precomputed-parameters/U1_late.csv")
```

Load the saved data and plot the exponential utilities:

```
fixed_U1_late <- read_csv("precomputed-parameters/U1_late.csv", show_col_types = FALSE)

fixed_U1_late %>%
  mutate_at(2:10, exp) %>%
  mutate(row_sums = rowSums(select(., 2:10))) %>%
  mutate_at(2:10, ~ . / row_sums) %>%
  select(-row_sums) %>%
  pivot_longer(
    cols = -d,
    names_to = "Construction",
    values_to = "Utility"
  ) %>%
  ggplot(aes(x = d, y = Utility, col = Construction, group = Construction)) +
  geom_line() +
  theme_bw() +
  scale_color_manual(values = cbbPalette) +
  ylab("Normalized exponentiated utilities")
```



Fitting RSA-SvI and alternative models

We have computed the informativity term of the utility for each candidate parse of each message. We can now fit the RSA-SvI model, but first, we consider an alternative literal model which will serve as our baseline.

Literal model

We consider a simple literal baseline, whereby negative expressions 'not adj' and 'not very adj' are simply treated as the negations of their affirmative counterparts (i.e., no implicature is considered). This is important because the parameters we fitted on the affirmative sentences already contain a lot of information, so we need to make sure that the RSA-SvI model does provide significant added value. Presumably, the literal model should perform very well on 'tall' since the puzzle there is precisely the absence of implicatures, but not so much on 'late', where most participants derive an implicature.

NB: This model uses the fitted ζ on 'late', while the RSA-SvI model will not have access to this. We could consider refitting ζ again to properly account for the additional degree of freedom, but this would make our baseline worse if anything, so let's be conservative.

```
Leffel_data_negative <- Leffel_data %>%
  mutate(subj_id = as.numeric(substr(subj_id, 5, 6))) %>%
  filter(Pred %in% c("notTall", "notVeryTall", "notLate", "notVeryLate") & subj_id != 6)
  %>%
  mutate(NormUnit2 = if_else(Adj == "Late", NormUnit / 16, NormUnit / 12)) # Divide to
  scale around an sd of ~1

fitted_mu_theta_tall <- read.csv("precomputed-parameters/fitted_mu_theta_tall.csv")$x
fitted_sigma_theta_tall <-
read.csv("precomputed-parameters/fitted_sigma_theta_tall.csv")$x
fitted_lambda_delta_tall <-
read.csv("precomputed-parameters/fitted_lambda_delta_tall.csv")$x
fitted_lambda_theta_late <-
read.csv("precomputed-parameters/fitted_lambda_theta_late.csv")$x
fitted_lambda_delta_late <-
read.csv("precomputed-parameters/fitted_lambda_delta_late.csv")$x
fitted_p_min_late <- read.csv("precomputed-parameters/fitted_p_min_late.csv")$x

# We need an index telling us which adjective each participant saw:
subj_adj <- Leffel_data_negative %>%
  group_by(subj_id) %>%
  summarise(adj = first(Adj) == "Late") %>%
  pull(adj) %>%
  as.numeric()

stan_data_negative <- list(
  N = nrow(Leffel_data_negative),
  S = n_distinct(Leffel_data_negative$subj_id),
  unit = Leffel_data_negative$NormUnit2,
  subject = Leffel_data_negative %>% pull(subj_id) %>% factor() %>% as.numeric(),
  adj = as.numeric(Leffel_data_negative$Adj == "Late"),
  adv = if_else(Leffel_data_negative$Pred %in% c("notVeryTall", "notVeryLate"), 1, 0),
  y = Leffel_data_negative$response / 100,
  subj_adj = subj_adj,
  # Need some wizardry to map the right values to the right participant:
  mu_theta_tall = if_else(subj_adj == 0, fitted_mu_theta_tall[pmax(1, cumsum(1 -
  subj_adj))], 0),
  sigma_theta_tall = if_else(subj_adj == 0, fitted_sigma_theta_tall[pmax(1, cumsum(1 -
  subj_adj))], 0),
  lambda_delta_tall = if_else(subj_adj == 0, fitted_lambda_delta_tall[pmax(1, cumsum(1 -
  subj_adj))], 0),
  lambda_theta_late = if_else(subj_adj == 1, fitted_lambda_theta_late[pmax(1,
  cumsum(subj_adj))], 0),
```

```
  lambda_delta_late = if_else(subj_adj == 1, fitted_lambda_delta_late[pmax(1,
  cumsum(subj_adj))], 0),
  p_min_late = if_else(subj_adj == 1, fitted_p_min_late[pmax(1, cumsum(subj_adj))], 0)
)

literal_model <- cmdstan_model("Stan-models/literal_model.stan")

# takes like 5s, only fitting eps
literal_model_fit <- literal_model$sample(
  data = stan_data_negative,
  chains = 8,
  parallel_chains = 14,
  iter_warmup = 1000,
  iter_sampling = 1000,
  refresh = 0,
  show_messages = F
)
```

```
## Running MCMC with 8 chains, at most 14 in parallel...
##
## Chain 1 finished in 4.5 seconds.
## Chain 4 finished in 4.4 seconds.
## Chain 2 finished in 4.6 seconds.
## Chain 3 finished in 4.6 seconds.
## Chain 6 finished in 4.5 seconds.
## Chain 5 finished in 4.7 seconds.
## Chain 7 finished in 4.6 seconds.
## Chain 8 finished in 4.7 seconds.
##
## All 8 chains finished successfully.
## Mean chain execution time: 4.6 seconds.
## Total execution time: 5.0 seconds.
```

```
literal_loglik_matrix <- literal_model_fit$draws("log_lik")
literal_r_eff <- relative_eff(exp(literal_loglik_matrix))
literal_loo <- loo(literal_loglik_matrix, r_eff = literal_r_eff)
```

```
## Warning: Some Pareto k diagnostic values are slightly high. See help('pareto-k-diagnostic') for deta
```

```
# Diagnostic:
# plot(literal_loo, label_points = TRUE)

# Save model draws (not worth it, as this model can be fitted very quickly)
#
literal_model_fit$save_output_files(dir="saved-models-draws/literal_model",basename="literal_model")
```

RSA-SvI models

We finally fit the RSA-SvI models.

There are a few differences with the model presented in the paper, namely:

- The random effect structure has been simplified. The model in the paper had a maximal RE structure, including random variation on the costs of negation and *very*, which led to extreme values. Here we only kept random effects for the rationality parameter and the cost of the base sentence, while the costs of negation and *very* are fixed across participants.

- The priors are slightly different. In particular, the costs of negation and *very* now have gamma priors (earlier they were log-normal among participants), and the priors on the location of α and the cost of the base sentences are more informative.
- We also test two intermediate models between the full model and the simplified one, namely one with EXH$_{\text{MIN}}$ but a single cost for *tall* and *late*, and the mirror image with separate costs but no EXH$_{\text{MIN}}$ parse. The former would not converge no matter how hard we tried.

```
# Prepare the data:
Leffel_data_negative <- Leffel_data %>%
  mutate(subj_id = as.numeric(substr(subj_id, 5, 6))) %>%
  filter(Pred %in% c("notTall", "notVeryTall", "notLate", "notVeryLate") & subj_id != 6)
  %>%
  mutate(NormUnit2 = if_else(Adj == "Late", NormUnit / 16, NormUnit / 12)) # Divide to
  scale around an sd of ~1

# Load by-participant parameters fitted earlier:

fitted_mu_theta_tall <- read.csv("precomputed-parameters/fitted_mu_theta_tall.csv")$x
fitted_sigma_theta_tall <-
read.csv("precomputed-parameters/fitted_sigma_theta_tall.csv")$x
fitted_lambda_delta_tall <-
read.csv("precomputed-parameters/fitted_lambda_delta_tall.csv")$x
fitted_lambda_theta_late <-
read.csv("precomputed-parameters/fitted_lambda_theta_late.csv")$x
fitted_lambda_delta_late <-
read.csv("precomputed-parameters/fitted_lambda_delta_late.csv")$x

subj_adj <- Leffel_data_negative %>%
  group_by(subj_id) %>%
  summarise(adj = first(Adj) == "Late") %>%
  pull(adj) %>%
  as.numeric()

stan_data_negative_SvI <- list(
  N = nrow(Leffel_data_negative),
  S = n_distinct(Leffel_data_negative$subj_id),
  adj = as.numeric(Leffel_data_negative$Adj == "Late"),
  adv = if_else(Leffel_data_negative$Pred %in% c("notVeryTall", "notVeryLate"), 1, 0),
  unit = Leffel_data_negative$NormUnit2,
  subject = Leffel_data_negative %>% pull(subj_id) %>% factor() %>% as.numeric(),
  subj_adj = subj_adj,
  y = Leffel_data_negative$response / 100,
  K = length(height_samples),
  heights = (height_samples - 70) / 4,
  tall_prior_mean = (tall_prior_mean - 70) / 4,
  tall_prior_sd = tall_prior_sig / 4,
  times = time_samples / 16,
  late_prior_mean = late_prior_mean / 16,
  late_prior_sd = late_prior_sig / 16,
  U1_null_tall = fixed_U1_tall$null_tall,
  U1_tall_pos = fixed_U1_tall$tall_pos,
  U1_not_tall = fixed_U1_tall$not_tall,
  U1_very_tall = fixed_U1_tall$very_tall,
```

```
  U1_not_very_tall_lit = fixed_U1_tall$not_very_tall_lit,
  U1_not_very_tall_exh = fixed_U1_tall$not_very_tall_exh,
  U1_null_late = fixed_U1_late$null_late,
  U1_late_min = fixed_U1_late$late_min,
  U1_late_pos = fixed_U1_late$late_pos,
  U1_not_late_min = fixed_U1_late$not_late_min,
  U1_not_late_pos = fixed_U1_late$not_late_pos,
  U1_very_late = fixed_U1_late$very_late,
  U1_not_very_late_lit = fixed_U1_late$not_very_late_lit,
  U1_not_very_late_exhpos = fixed_U1_late$not_very_late_exhpos,
  U1_not_very_late_exhmin = fixed_U1_late$not_very_late_exhmin,
  # Need some wizardry to get get the right values in the right order:
  mu_theta_tall = if_else(subj_adj == 0, fitted_mu_theta_tall[pmax(1, cumsum(1 -
  subj_adj))], 0),
  sigma_theta_tall = if_else(subj_adj == 0, fitted_sigma_theta_tall[pmax(1, cumsum(1 -
  subj_adj))], 1), # set at 1 when not used to avoid accidental division by zero
  lambda_delta_tall = if_else(subj_adj == 0, fitted_lambda_delta_tall[pmax(1, cumsum(1 -
  subj_adj))], 0),
  lambda_theta_late = if_else(subj_adj == 1, fitted_lambda_theta_late[pmax(1,
  cumsum(subj_adj))], 1), # again, this is to avoid a division by 0
  lambda_delta_late = if_else(subj_adj == 1, fitted_lambda_delta_late[pmax(1,
  cumsum(subj_adj))], 0)
)
```

The next block fits the four models. Evaluation is set to false, as this would take about 2 days on a laptop.

```
# Models with separate costs and both exh_pos and exh_min parses for late:
SvI_min_exhmin_sepcost_model <-
cmdstan_model("Stan-Models/RSA-SvI_min_exhmin_sepcost_model.stan")
# Models with a shared cost for both adjectives and both exh_pos and exh_min parses for
late:
SvI_min_exhmin_onecost_model <-
cmdstan_model("Stan-Models/RSA-SvI_min_exhmin_onecost_model.stan")
# Models with separate costs and only the exh_pos parse for late:
SvI_min_sepcost_model <- cmdstan_model("Stan-Models/RSA-SvI_min_sepcost_model.stan")
# Models with a shared cost for both adjectives and only the exh_pos parse for late:
SvI_min_onecost_model <- cmdstan_model("Stan-Models/RSA-SvI_min_onecost_model.stan")

# Random init functions:
SvI_sepcost_init_function <- function() {
  list(
    m_alpha = runif(1, -0.5, 0.5),
    m_cost_sen_tall = runif(1, 1, 4),
    m_cost_sen_late = runif(1, 0, 2),
    cost_neg = runif(1, 1, 3),
    cost_very = runif(1, .5, 1),
    z_v = matrix(runif(2 * n_distinct(Leffel_data_negative$subj_id), -.15, .15), nrow =
    2),
    s_v = runif(2, 1, 2),
    eps = runif(1, .1, .4)
  )
}
SvI_onecost_init_function <- function() {
  list(
```

```
    m_alpha = runif(1, -0.5, 0.5),
    m_cost_sen = runif(1, 1, 3),
    cost_neg = runif(1, 1, 3),
    cost_very = runif(1, .5, 1),
    z_v = matrix(runif(2 * n_distinct(Leffel_data_negative$subj_id), -.15, .15), nrow =
    2),
    s_v = runif(2, 1, 2),
    eps = runif(1, .1, .4)
  )
}

# Fit the models with cmdstanr

# 16h, 3% divergent transitions with stepsize=0.5 and adapt_delta = .98
# 32h, 2% divergent transitions with stepsize=0.25 and adapt_delta = .99 + one chain hit
max treedepth (all other chains had finished by 18h)
SvI_min_exhmin_sepcost_fit <- SvI_min_exhmin_sepcost_model$sample(
  data = stan_data_negative_SvI,
  init = SvI_sepcost_init_function,
  chains = 8,
  parallel_chains = 14,
  iter_warmup = 1500,
  iter_sampling = 1000,
  step_size = 0.25,
  adapt_delta = 0.99,
  max_treedepth = 11
)
SvI_min_exhmin_sepcost_fit$save_output_files(dir =
"saved-models-draws/SvI_min_exhmin_sepcost", basename = "SvI_min_exhmin_sepcost")

# with step_size = 0.15, adapt_delta = 0.95, Rhat=1.7 for s_v[2] (less iterations though)
# almost 21h with step_size = 0.25, adapt_delta = 0.98, even more divergent transitions
(57%), and Rhat=1.5 for s_v[2]!!
# retry with stronger prior on s_v (gamma(1.3,8) instead of gamma(1.3,4)) to avoid high
alpha values. And further increase adapt_delta to .99
# -> 24h, 34% divergent transitions, Rhat=1.3 for s_v[2] -> giving up on this model.
SvI_min_exhmin_onecost_fit <- SvI_min_exhmin_onecost_model$sample(
  data = stan_data_negative_SvI,
  init = SvI_onecost_init_function,
  chains = 8,
  parallel_chains = 14,
  iter_warmup = 1500,
  iter_sampling = 1000,
  step_size = 0.25,
  adapt_delta = 0.99,
  max_treedepth = 12 # it does reach 10 occasionally
)
SvI_min_exhmin_onecost_fit$save_output_files(dir =
"saved-models-draws/SvI_min_exhmin_onecost", basename = "SvI_min_exhmin_onecost")

# <4h, perfect
SvI_min_sepcost_fit <- SvI_min_sepcost_model$sample(
  data = stan_data_negative_SvI,
```

```r
  init = SvI_sepcost_init_function,
  chains = 8,
  parallel_chains = 14,
  iter_warmup = 1500,
  iter_sampling = 1000,
  step_size = 0.15,
  adapt_delta = 0.96,
  max_treedepth = 11
)
SvI_min_sepcost_fit$save_output_files(dir = "saved-models-draws/SvI_min_sepcost",
basename = "SvI_min_sepcost")

# 6h, perfect
SvI_min_onecost_fit <- SvI_min_onecost_model$sample(
  data = stan_data_negative_SvI,
  init = SvI_onecost_init_function,
  chains = 8,
  parallel_chains = 14,
  iter_warmup = 1500,
  iter_sampling = 1000,
  step_size = 0.25,
  adapt_delta = 0.97,
  max_treedepth = 11
)
SvI_min_onecost_fit$save_output_files(dir = "saved-models-draws/SvI_min_onecost",
basename = "SvI_min_onecost")
```

Examine the results:

```r
# Work with rstan for pairs and other useful functions

# Load from cmdstanr models if previous block was run:
# SvI_min_exhmin_sepcost_rstan_fit <-
rstan::read_stan_csv(SvI_min_exhmin_sepcost_fit$output_files())
# SvI_min_exhmin_onecost_rstan_fit <-
rstan::read_stan_csv(SvI_min_exhmin_onecost_fit$output_files())
# SvI_min_sepcost_rstan_fit <- rstan::read_stan_csv(SvI_min_sepcost_fit$output_files())
# SvI_min_onecost_rstan_fit <- rstan::read_stan_csv(SvI_min_onecost_fit$output_files())

# Load from saved draws otherwise:
SvI_min_exhmin_sepcost_rstan_fit <-
rstan::read_stan_csv(paste0("saved-models-draws/SvI_min_exhmin_sepcost/",
list.files("saved-models-draws/SvI_min_exhmin_sepcost", pattern = "260eb3")))
SvI_min_exhmin_onecost_rstan_fit <-
rstan::read_stan_csv(paste0("saved-models-draws/SvI_min_exhmin_onecost/",
list.files("saved-models-draws/SvI_min_exhmin_onecost", pattern = "8ab7a7")))
SvI_min_sepcost_rstan_fit <-
rstan::read_stan_csv(paste0("saved-models-draws/SvI_min_sepcost/",
list.files("saved-models-draws/SvI_min_sepcost", pattern = "32ae9c")))
SvI_min_onecost_rstan_fit <-
rstan::read_stan_csv(paste0("saved-models-draws/SvI_min_onecost/",
list.files("saved-models-draws/SvI_min_onecost", pattern = "94b022")))

# Pairs plots:
```

```
# pairs(SvI_min_exhmin_sepcost_rstan_fit,
#
pars=c("m_alpha","m_cost_sen_tall","m_cost_sen_late","cost_neg","cost_very","s_v","lp__"))
# pairs(SvI_min_exhmin_onecost_rstan_fit,
#       pars=c("m_alpha","m_cost_sen","cost_neg","cost_very","s_v","lp__"))
# pairs(SvI_min_sepcost_rstan_fit,
#
pars=c("m_alpha","m_cost_sen_tall","m_cost_sen_late","cost_neg","cost_very","s_v","lp__"))
# pairs(SvI_min_onecost_rstan_fit,
#       pars=c("m_alpha","m_cost_sen","cost_neg","cost_very","s_v","lp__"))

# Parameters:
summary(SvI_min_exhmin_sepcost_rstan_fit, pars = c("m_alpha", "m_cost_sen_tall",
"m_cost_sen_late", "cost_neg", "cost_very", "s_v"), use_cache = F)$summary %>%
  as_tibble(rownames = NA) %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column("variable") %>%
  select(c(1:5, 9:11)) %>%
  kableExtra::kbl(digits = c(0, 2, 4, 2, 2, 2, 0, 2), booktabs = T)
```

variable	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
m_alpha	2.32	0.0073	0.25	1.75	2.74	1222	1.00
m_cost_sen_tall	2.27	0.0255	1.13	0.72	4.92	1948	1.01
m_cost_sen_late	-0.18	0.0005	0.02	-0.22	-0.14	1631	1.00
cost_neg	2.52	0.0217	1.00	1.21	5.01	2127	1.00
cost_very	1.34	0.0109	0.81	0.34	3.40	5502	1.00
s_v[1]	1.25	0.0088	0.29	0.76	1.86	1067	1.01
s_v[2]	0.09	0.0005	0.03	0.05	0.15	2310	1.00

```
summary(SvI_min_exhmin_onecost_rstan_fit, pars = c("m_alpha", "m_cost_sen", "cost_neg",
"cost_very", "s_v"), use_cache = F)$summary %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column("variable") %>%
  select(c(1:5, 9:11)) %>%
  kableExtra::kbl(digits = c(0, 2, 4, 2, 2, 2, 0, 2), booktabs = T)
```

variable	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
m_alpha	1.03	0.0297	0.27	0.50	1.58	84	1.08
m_cost_sen	-0.17	0.0031	0.07	-0.30	-0.02	453	1.03
cost_neg	3.48	0.0284	1.23	1.77	6.56	1879	1.01
cost_very	1.23	0.0136	0.77	0.29	3.17	3188	1.00
s_v[1]	1.51	0.0157	0.19	1.17	1.92	152	1.05
s_v[2]	0.23	0.0399	0.14	0.07	0.53	12	1.49

```
summary(SvI_min_sepcost_rstan_fit, pars = c("m_alpha", "m_cost_sen_tall",
"m_cost_sen_late", "cost_neg", "cost_very", "s_v"), use_cache = F)$summary %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column("variable") %>%
  select(c(1:5, 9:11)) %>%
  kableExtra::kbl(digits = c(0, 2, 4, 2, 2, 2, 0, 2), booktabs = T)
```

variable	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
m_alpha	0.27	0.0099	0.25	-0.17	0.81	638	1.01
m_cost_sen_tall	-0.55	0.0308	0.87	-2.72	0.69	806	1.01
m_cost_sen_late	0.33	0.0171	0.43	-0.40	1.27	637	1.01
cost_neg	2.16	0.0373	0.87	1.10	4.41	549	1.02
cost_very	1.17	0.0227	0.98	0.02	3.56	1860	1.00
s_v[1]	0.72	0.0106	0.19	0.40	1.14	317	1.03
s_v[2]	1.33	0.0170	0.46	0.59	2.36	727	1.01

```
summary(SvI_min_onecost_rstan_fit, pars = c("m_alpha", "m_cost_sen", "cost_neg",
"cost_very", "s_v"), use_cache = F)$summary %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column("variable") %>%
  select(c(1:5, 9:11)) %>%
  kableExtra::kbl(digits = c(0, 2, 4, 2, 2, 2, 0, 2), booktabs = T)
```

variable	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
m_alpha	0.43	0.0034	0.18	0.07	0.79	2929	1
m_cost_sen	0.14	0.0064	0.27	-0.42	0.68	1821	1
cost_neg	2.01	0.0185	0.73	1.10	3.88	1579	1
cost_very	1.33	0.0170	1.04	0.03	3.79	3739	1
s_v[1]	0.71	0.0057	0.19	0.36	1.12	1161	1
s_v[2]	1.09	0.0064	0.35	0.55	1.91	3044	1

```
# LOO evaluation:
SvI_min_exhmin_sepcost_loo <- loo(SvI_min_exhmin_sepcost_rstan_fit)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
SvI_min_exhmin_onecost_loo <- loo(SvI_min_exhmin_onecost_rstan_fit)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
SvI_min_sepcost_loo <- loo(SvI_min_sepcost_rstan_fit)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
SvI_min_onecost_loo <- loo(SvI_min_onecost_rstan_fit)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
# Diagnostic plots:
# plot(SvI_min_exhmin_sepcost_loo,label_points=T)
# plot(SvI_min_exhmin_onecost_loo,label_points=T)
# plot(SvI_min_sepcost_loo,label_points=T)
# plot(SvI_min_onecost_loo,label_points=T)

loo_comparison <- loo_compare(SvI_min_exhmin_sepcost_loo, SvI_min_exhmin_onecost_loo,
SvI_min_sepcost_loo, SvI_min_onecost_loo, literal_loo)
loo_comparison %>% kableExtra::kbl(digits = 1, booktabs = T)
```

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
model1	0.0	0.0	888.5	126.1	67.6	7.2	-1777.0	252.2
model2	-12.2	7.7	876.3	125.7	86.2	11.4	-1752.6	251.4
model3	-231.7	62.5	656.8	123.9	56.4	4.4	-1313.7	247.8
model4	-232.9	62.4	655.6	124.0	56.7	4.8	-1311.1	248.1
model5	-894.0	143.7	-5.5	165.2	14.8	3.9	11.0	330.3

We can also compare the RSA-SvI to the literal model separately for each adjective. Unsurprisingly, the RSA-SvI is particularly better with 'late', but it also outperforms the literal model on 'tall'.

```
# LOO by adjective
literal_tall_loglik_matrix <- literal_model_fit$draws("log_lik")[, , subj_adj == 0]
literal_tall_r_eff <- relative_eff(exp(literal_tall_loglik_matrix))
literal_tall_loo <- loo(literal_tall_loglik_matrix, r_eff = literal_tall_r_eff)

literal_late_loglik_matrix <- literal_model_fit$draws("log_lik")[, , subj_adj == 1]
literal_late_r_eff <- relative_eff(exp(literal_late_loglik_matrix))
literal_late_loo <- loo(literal_late_loglik_matrix, r_eff = literal_late_r_eff)
```

```
## Warning: Some Pareto k diagnostic values are slightly high. See help('pareto-k-diagnostic') for deta
```

```
SvI_loglik_tall <- rstan::extract(SvI_min_exhmin_sepcost_rstan_fit, pars = "log_lik",
permuted = F)[, , subj_adj == 0]
SvI_loglik_late <- rstan::extract(SvI_min_exhmin_sepcost_rstan_fit, pars = "log_lik",
permuted = F)[, , subj_adj == 1]
SvI_r_eff_tall <- relative_eff(exp(SvI_loglik_tall))
SvI_loo_tall <- loo(SvI_loglik_tall, r_eff = SvI_r_eff_tall)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
SvI_r_eff_late <- relative_eff(exp(SvI_loglik_late))
SvI_loo_late <- loo(SvI_loglik_late, r_eff = SvI_r_eff_late)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
loo_compare(SvI_loo_tall, literal_tall_loo) %>% kableExtra::kbl(digits = 1, booktabs = T)
```

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
model1	0.0	0.0	561.4	60.3	21.5	2.8	-1122.8	120.7
model2	-219.5	26.7	341.9	44.2	3.0	0.5	-683.8	88.3

```
loo_compare(SvI_loo_late, literal_late_loo) %>% kableExtra::kbl(digits = 1, booktabs = T)
```

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
model1	0.0	0.0	327.1	107.6	46.1	6.1	-654.2	215.3
model2	-674.5	131.6	-347.4	136.5	11.8	3.7	694.7	273.0

We then look at the posterior probabilities for EXH parses.

```
# Posterior on exhaustification:
post_exhpos <- rstan::extract(SvI_min_exhmin_sepcost_rstan_fit, pars =
"post_logit_exhpos")$post_logit_exhpos
post_exhmin <- rstan::extract(SvI_min_exhmin_sepcost_rstan_fit, pars =
"post_logit_exhmin")$post_logit_exhmin
```

```r
# Distribution among participants (tall):
range(plogis(colMeans(post_exhpos[, subj_adj == 0])))
```

```
## [1] 0.001271713 0.296359704
```

```r
mean(plogis(colMeans(post_exhpos[, subj_adj == 0])))
```

```
## [1] 0.145765
```

```r
# Overall mean and CI:
mean(plogis(post_exhpos[, subj_adj == 0]))
```

```
## [1] 0.1773016
```

```r
hdi(rowMeans(plogis(post_exhpos[, subj_adj == 0])), ci = .95)
```

```
## 95% HDI: [0.15, 0.20]
```

```r
# Distribution among participants (late):
range(plogis(colMeans(post_exhmin[, subj_adj == 1])) + plogis(colMeans(post_exhpos[,
subj_adj == 1])))
```

```
## [1] 0.00744617 0.99996715
```

```r
mean(plogis(colMeans(post_exhmin[, subj_adj == 1])) + plogis(colMeans(post_exhpos[,
subj_adj == 1])))
```

```
## [1] 0.3889174
```

```r
# Overall mean and CI:
mean(plogis(post_exhmin[, subj_adj == 1]) + plogis(post_exhpos[, subj_adj == 1]))
```

```
## [1] 0.4131475
```

```r
hdi(rowMeans(plogis(post_exhmin[, subj_adj == 1]) + plogis(post_exhpos[, subj_adj ==
1])), ci = .95)
```

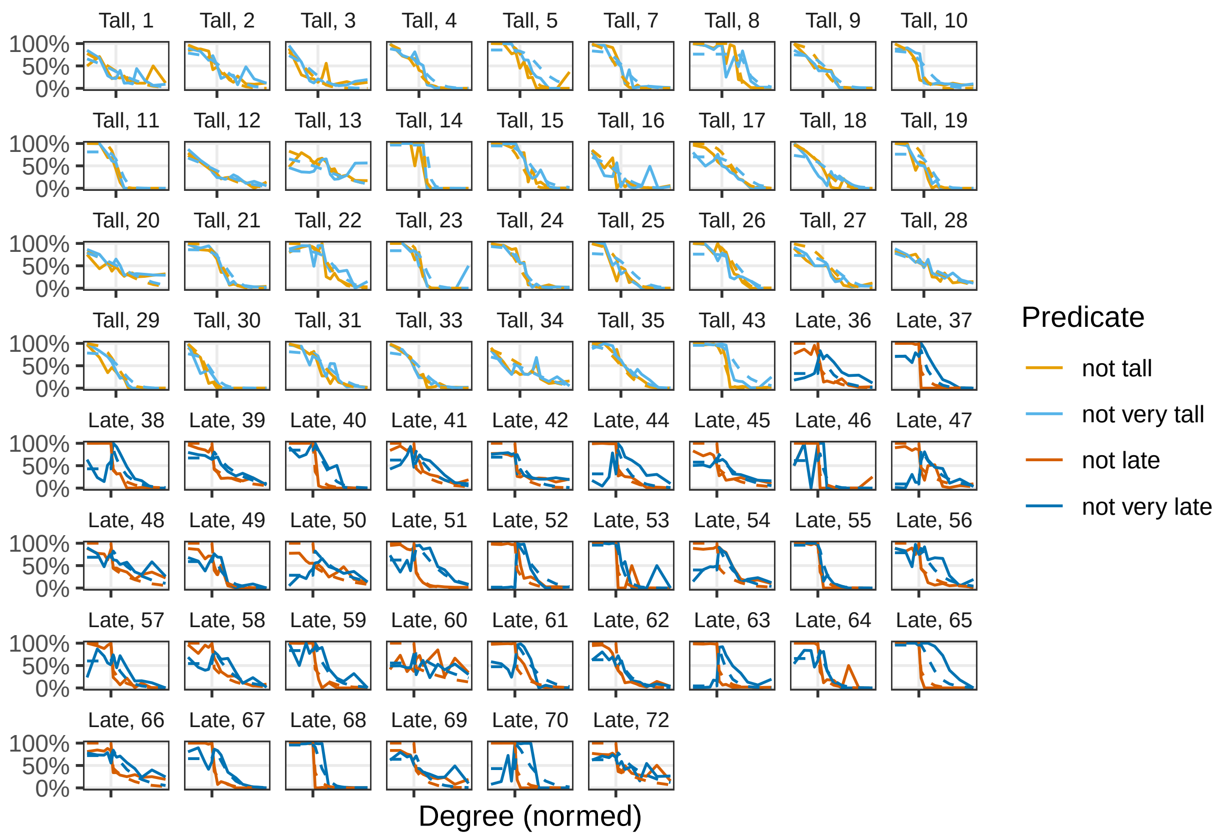
```
## 95% HDI: [0.40, 0.43]
```

We can plot the model predictions against the data.

```r
extracted_predictions <- colMeans(rstan::extract(SvI_min_exhmin_sepcost_rstan_fit, pars =
"pred")$pred)


Leffel_data_negative <- Leffel_data_negative %>%
  mutate(Prediction = extracted_predictions)


# Graph posterior predictions (not adj and not very adj)
# the 'not adj' construction isn't particularly interesting since the only degree of
freedom for the model is in the proportion of MIN parses for 'late'.
Leffel_data_negative %>%
  group_by(subj_id, Pred, Adj, NormUnit2) %>%
  summarize(
    response = mean(response) / 100,
    prediction = mean(Prediction)
```

```
  ) %>%
  mutate(
    Predicate = factor(Pred,
      levels = c("notTall", "notVeryTall", "notLate", "notVeryLate"),
      labels = c("not tall", "not very tall", "not late", "not very late")
    ),
    Adj = factor(Adj, levels = c("Tall", "Late"))
  ) %>%
  ggplot(aes(x = NormUnit2, y = response, color = Predicate)) +
  facet_wrap(~ Adj + subj_id, scales = "free_x", labeller = function(labs) {
    label_value(labs, multi_line = FALSE)
  }) +
  geom_line() +
  geom_line(aes(y = prediction), linetype = 2, show.legend = F) +
  scale_x_continuous(name = "Degree (normed)", breaks = c(0), minor_breaks = NULL, labels
  = NULL) +
  scale_y_continuous(name = NULL, breaks = c(0, .5, 1), minor_breaks = NULL, labels =
  scales::percent) +
  scale_color_manual(values = cbbPalette[c(1, 2, 6, 5)]) +
  theme_bw() +
  theme(
    strip.background = element_blank(), # element_rect(fill="transparent"),
    strip.text = element_text(size = 8),
    panel.spacing.y = unit(0, "lines")
  ) +
  coord_cartesian(ylim = c(0, 1))
```

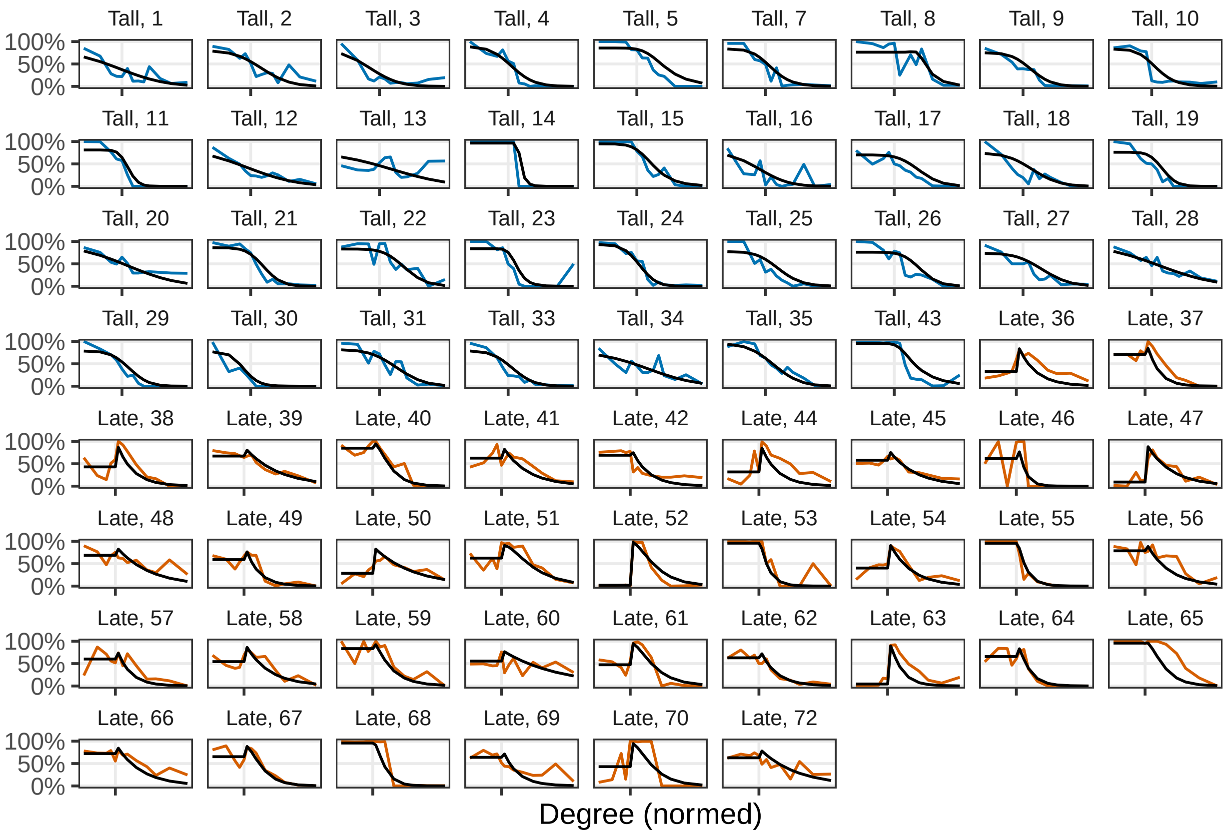
```
## `summarise()` has grouped output by 'subj_id', 'Pred', 'Adj'. You can override
## using the `.groups` argument.
```




```
# Graph posterior predictions (not very adj only)
# pdf(file="posterior_not_very_model.pdf",width=10,height=6)
Leffel_data_negative %>%
  group_by(subj_id, Pred, Adj, NormUnit2) %>%
  summarize(
    response = mean(response) / 100,
    prediction = mean(Prediction)
  ) %>%
  mutate(
    Predicate = factor(Pred,
      levels = c("notTall", "notVeryTall", "notLate", "notVeryLate"),
      labels = c("not tall", "not very tall", "not late", "not very late")
    ),
    Adj = factor(Adj, levels = c("Tall", "Late"))
  ) %>%
  filter(Predicate %in% c("not very tall", "not very late")) %>%
  ggplot(aes(x = NormUnit2, y = response, col = Adj)) +
  facet_wrap(~ Adj + subj_id, scales = "free_x", labeller = function(labs) {
    label_value(labs, multi_line = FALSE)
  }) +
  geom_line() +
  geom_line(aes(y = prediction), linetype = 1, show.legend = F, col = "black") +
  scale_x_continuous(name = "Degree (normed)", breaks = c(0), minor_breaks = NULL, labels
  = NULL) +
  scale_y_continuous(name = NULL, breaks = c(0, .5, 1), minor_breaks = NULL, labels =
  scales::percent) +
  scale_color_manual(values = cbbPalette[5:6], guide = "none") +
```

```
  theme_bw() +
  theme(
    strip.background = element_blank(), # element_rect(fill="transparent"),
    strip.text = element_text(size = 8),
    panel.spacing.y = unit(0, "lines")
  )
```

```
## `summarise()` has grouped output by 'subj_id', 'Pred', 'Adj'. You can override
## using the `.groups` argument.
```

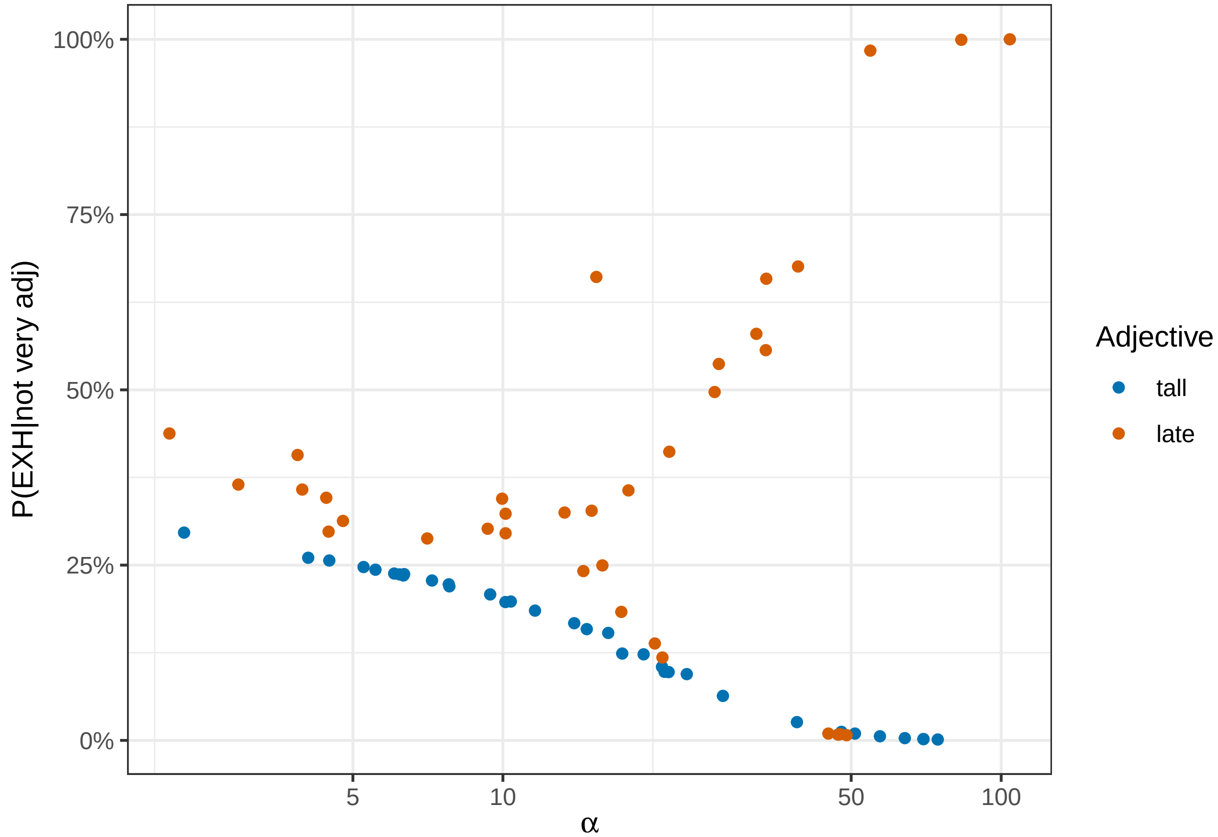



```
# dev.off()
```

Finally, we can look at the fitted posterior on EXH as a function of the rationality parameter. The plot confirms that for 'tall', exhaustivity is impossible with high rationality, whereas for 'late' both the exhaustive and the literal interpretation are possible.

```
alpha <- rstan::extract(SvI_min_exhmin_sepcost_rstan_fit, pars = "alpha")$alpha

# pdf(file="pEXH-by-alpha.pdf",width=7,height=4)
ggplot(data = NULL, aes(x = colMeans(alpha), y = plogis(colMeans(post_exhpos)) + subj_adj
* plogis(colMeans(post_exhmin)), color = factor(subj_adj, levels = c(0, 1), labels =
c("tall", "late")))) +
  geom_point() +
  scale_color_manual(name = "Adjective", values = cbbPalette[5:6]) +
  scale_x_continuous(name = expression(alpha), trans = "log", breaks = c(.1, .5, 1, 5,
  10, 50, 100), minor_breaks = c(.2, 2, 20)) +
  scale_y_continuous(name = expression("P(EXH|not very adj)"), labels = scales::percent)
  +
```

```
theme_bw()
```




```
# dev.off()
```