

# Computational Cognitive Science 2025-2026

```
library(CircStats)

## Warning: package 'CircStats' was built under R version 4.4.3
## Loading required package: MASS
## Loading required package: boot
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.4.3
library(dplyr)

##
## Attaching package: 'dplyr'
## The following object is masked from 'package:MASS':
##
##      select
## The following objects are masked from 'package:stats':
##
##      filter, lag
## The following objects are masked from 'package:base':
##
##      intersect, setdiff, setequal, union
library(patchwork)

## Warning: package 'patchwork' was built under R version 4.4.3
##
## Attaching package: 'patchwork'
## The following object is masked from 'package:MASS':
##
##      area
```

## Tutorial 7: From idea to results

In this tutorial, we want to recap some of the major methodological elements you have learned in this class. There is not so much coding here for you to do, but more of a conceptual tour and questions related to that.

Hopefully, this gives you a blueprint and set of tools that you can use in your own research going forward.

We will go through the following steps:

- 1) Formulate two models.
- 2) Ensure we can in principle recover critical model parameters.
- 3) Ensure we can in principle recover the correct model.
- 4) Thinking about study design.

- 5) Determine the winning model from data.
- 6) Model criticism: Where does our winning model fail?

## The phenomenon:

Visual working memory capacity.

Working memory (WM) is a crucial component of cognition. It allows you to maintain relevant information for a short moment in time and to manipulate it in your mind (e.g., during mental calculation). A key property of WM is its capacity limitation. A famous paper by George Miller (1957) stated that people can hold  $7 \pm 2$  items in WM. Typically, people have used things like lists of words or numbers. However, these can be chunked (e.g., the words “tree” and “leaf” may be remembered as one chunk). Luck & Vogel (1997) used visual information like colors, which are harder (or impossible) to chunk, and estimated a capacity of 3-4 independent items.

While visual WM has been shown to be clearly capacity-limited, a debate started in the 2000s, what the nature of this capacity limitation is. The idea that there is a discrete number of items that can be memorized was put under scrutiny. We will look at two canonical models of visual WM capacity limitations (note, many hybrids and variations exist).

1. Slot theory of WM: WM is limited by a discrete number of items. If the capacity of WM is  $K$  items, only up to  $K$  items can be held in memory at the same time.
2. Resource theory of WM: There is no limit on the number of items. Instead, there is a continuous fixed resource, which can be distributed across (in principle) infinitely many items. The more resource an items is allocated, the more precisely it is remembered.

**Question 1:** Before even starting to think about models. What type of experiment should we probably run? We typically have limited time and money and need to carefully select how to design an experiment that best

**Solution 1:** *Several correct answers here: We need to probe memory at and beyond capacity limitations. The models make different predictions on what happens if there are multiple items - we should manipulate this. We also need to be able to measure “precision” of WM somehow.*

## The data

At this point, we would not necessarily have any data from your own experiment. However, there might be open data available that you could study first to get an intuition. We here use openly available data to make things more concrete.

The experiment: On each trial, participants see  $M$  number of items on the screen (e.g.,  $M = 2$  or  $M = 4$ ) for 400 milliseconds. Each item is an arrow pointing in a particular direction. An arrow can point in any possible direction and we can describe the direction by an angle (e.g., relative to the vertical midline). Instead of degrees, we use radians (0 to 360 degrees correspond to 0 to  $2\pi$  in terms of radians). The participants are instructed to try to remember the  $M$  items as well as possible (encoding period). After the 400ms, the items disappear and the screen turns blank for 1s (delay period). Now a randomly oriented arrow appears at one of the location where an items was (recall period). By moving the mouse, the participant can turn the arrow until it matches as closely as possible the remembered orientation at that location. We can now record the error between what the participant thought the orientation was and what it actually was (“response error”).

We will use parts of data from an actual study. Note, we are aggregating the data into one “super subject” by dropping the subjects variable. This is certainly not the best strategy (for many reasons you have learned), but it will make things simpler for this tutorial.

```
df <- read.table("data.txt", skip = 3, header = TRUE, sep = "")
# the original data was using a line orientation (0-180 degrees), not arrows
error <- df$AdjustmentError1 / 180 * 2 * pi
setsize <- df$SetSize
df <- data.frame(error=error, setsize=setsize)
```

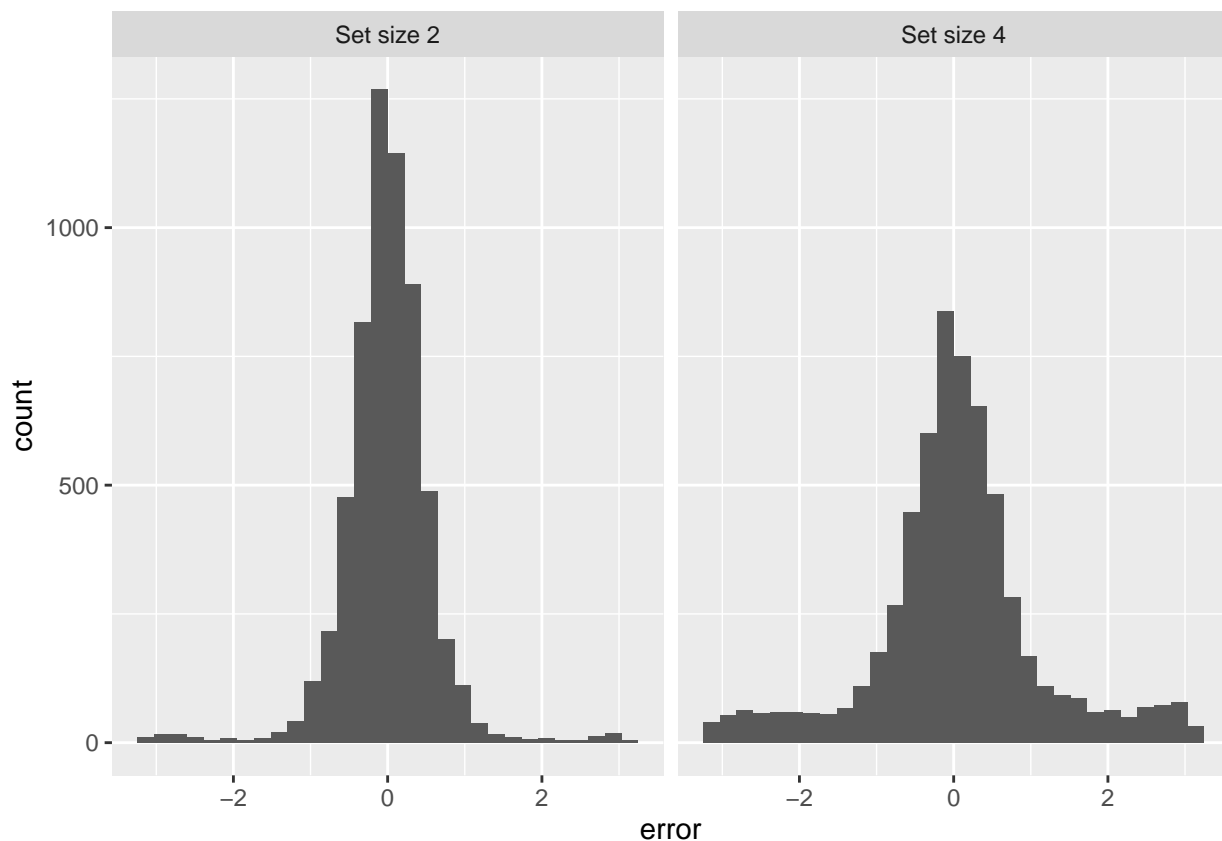
There are  $N = 12000$  trials - 6000 per setsize. Let's plot how the errors distribute for each setsize:

```
library(ggplot2)
library(patchwork)

labels_setsize <- c(
  `2` = "Set size 2",
  `4` = "Set size 4"
)

ggplot(df, aes(x = error)) +
  geom_histogram() +
  facet_wrap(~ setsize, labeller = labeller(setsize = labels_setsize))

## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



**Question 2:** How do the response errors differ between the two setsize conditions?

**Solution 2:** (1) Responses at set size 2 are more precise (closer to 0) than at set size 4. (2) The distribution of set size 2 looks Gaussian. At set size 4 it is unclear whether the errors at  $< -2$  or  $> 2$  could be captured with a Gaussian distribution. One could suspect that this may reflect random guessing. (3) The plots make it look like there are small bumps at  $-\pi$  and  $\pi$ . Maybe this is when participants remember the opposite direction?

## 1) From idea to models

Let's consider our two hypotheses of how these capacity limitations may arise.

A) We have a fixed slot limit. On each trial, only up to  $K$  items can be memorized.

B) We have a continuous resource, that we can allocate across items in memory. On each trial, all items are memorized, but with more items in memory, the precision of each one of them decreases.

Let's define the relevant variables:

- $e_i$  is the error that is made on the  $i$ -th trial in reporting the orientation  $e$  of the arrow in Radians (from 0 to  $2\pi$ )
- $N$ : the number trials
- $M_i$ : the number of items in trial  $i$ .

If an item was memorized, it will not be recalled exactly, but with some error. A good start (in absence of any contradicting evidence) is a Gaussian error distribution. Gaussian error on a circular variable is modeled by a [vonMises distribution](#).

$$p(e) = VM(e; 0, \kappa)$$

The first parameter of the VM distribution is the mean. We set it to zero, assuming that there is no systematic bias (i.e., clockwise or counterclockwise) in the response errors.

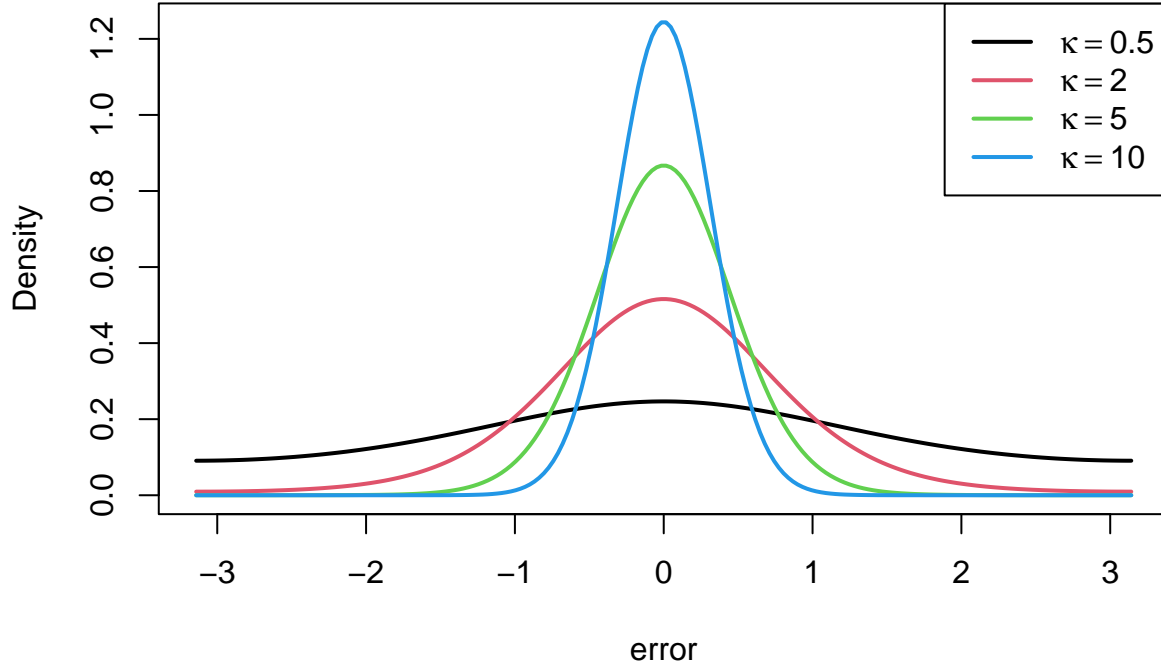
The second parameter  $\kappa$  determines the precision of the distribution ( $\sim$  inverse of the standard deviation). Let's visualize the distribution for a few values of  $\kappa$ . These distributions show the density of the response error  $e$ .

```
e <- seq(-pi, pi, length = 200)

kappa_values <- c(0.5, 2, 5, 10)
densities <- sapply(kappa_values, function(k) dvm(e, mu = 0, kappa = k))

plot(as.numeric(e), densities[,1], type = "l", lwd = 2,
     ylim = c(0, max(densities)), xlab = "error", ylab = "Density",
     main = "vonMises distribution for different Kappa values")
for(i in 2:length(kappa_values)){
  lines(as.numeric(e), densities[,i], lwd = 2, col = i)
}
legend("topright", legend = sapply(kappa_values, function(v) bquote(kappa == .(v))),
     col = 1:length(kappa_values), lwd = 2)
```

## vonMises distribution for different Kappa values



Equipped with this error distribution, we can think about our two models.

### Model A: slot limit

- On trial  $i$ ,  $M_i$  items were presented.
- We assume a slot limit of  $K$  items in memory.
  - $K \geq M$ : all  $N$  items are stored in memory.
  - $K < M$ : a random subset of  $K$  out of the  $M_i$  items is stored in memory.
- During the recall period, a random item  $j$  (probability  $1/N$ ) is probed.
  - if the item was stored in memory (probability  $K/M$ ), the likelihood is  $p(e) = VM(e; 0, \kappa_1)$
  - if the item was not stored in memory (probability  $1 - K/M$ ), the likelihood is a uniform distribution  $p(e) = \frac{1}{2\pi}$

Note, that we have two free parameters here:  $K$  the item limit in memory and  $\kappa$ , the precision of a memorized item.

$$p(e) = \begin{cases} VM(e; 0, \kappa), & \text{if } K \leq M, \\ \frac{K}{M} VM(e; 0, \kappa) + (1 - \frac{K}{M}) \frac{1}{2\pi}, & \text{if } K > M. \end{cases}$$

Here, we define the likelihood function:

```
ll_mA <- function(response_errors, setSizes, params) {
  K <- params[2]
  kappa <- params[1]

  nTrials <- length(response_errors) # N
  ll <- rep(0, nTrials)

  for (i in 1:nTrials) {
    M <- setSizes[i]
```

```

if (K >= M) {
  ll[i] <- log(dvm(response_errors[i], mu=0, kappa=kappa))
} else {
  ll[i] <- log(K/M * dvm(response_errors[i], mu=0, kappa=kappa) + (1-K/M) * 1/(2*pi))
}
}
ll
}

```

Let's plot the likelihood for  $K = 3$  and  $\kappa = 3$

```

e <- seq(-pi, pi, length = 200)

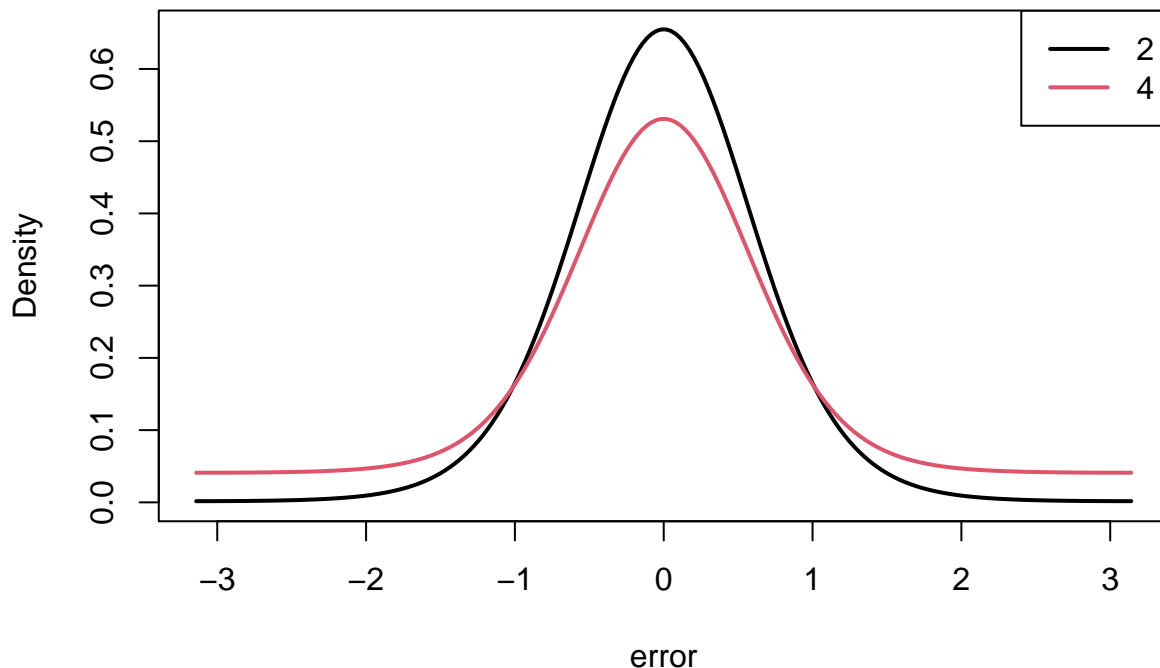
K <- 3
kappa <- 3

setSizes <- c(2, 4)
densities <- sapply(setSize, function(s) exp(ll_mA(e, setSize=rep(s, 200), c(kappa, K))))

plot(as.numeric(e), densities[,1], type = "l", lwd = 2,
     ylim = c(0, max(densities)), xlab = "error", ylab = "Density",
     main = "likelihood for model A at different set sizes")
for(i in 2:length(setSize)){
  lines(as.numeric(e), densities[,i], lwd = 2, col = i)
}
legend("topright", legend = sapply(setSize, function(s) paste(s)),
      col = 1:length(setSize), lwd = 2)

```

**likelihood for model A at different set sizes**



## Model B: resource limit

For the resource model, we do not assume an item limit  $K$ . Instead, we assume the maximum precision at setsize 1 ( $\kappa_1$ ) be the resource, which is then divided among other items in case there is more than one item. (For a more principled approach, see van den Berg, Awh, & Ma, 2014).

$$\kappa_M = \frac{\kappa_1}{M}$$

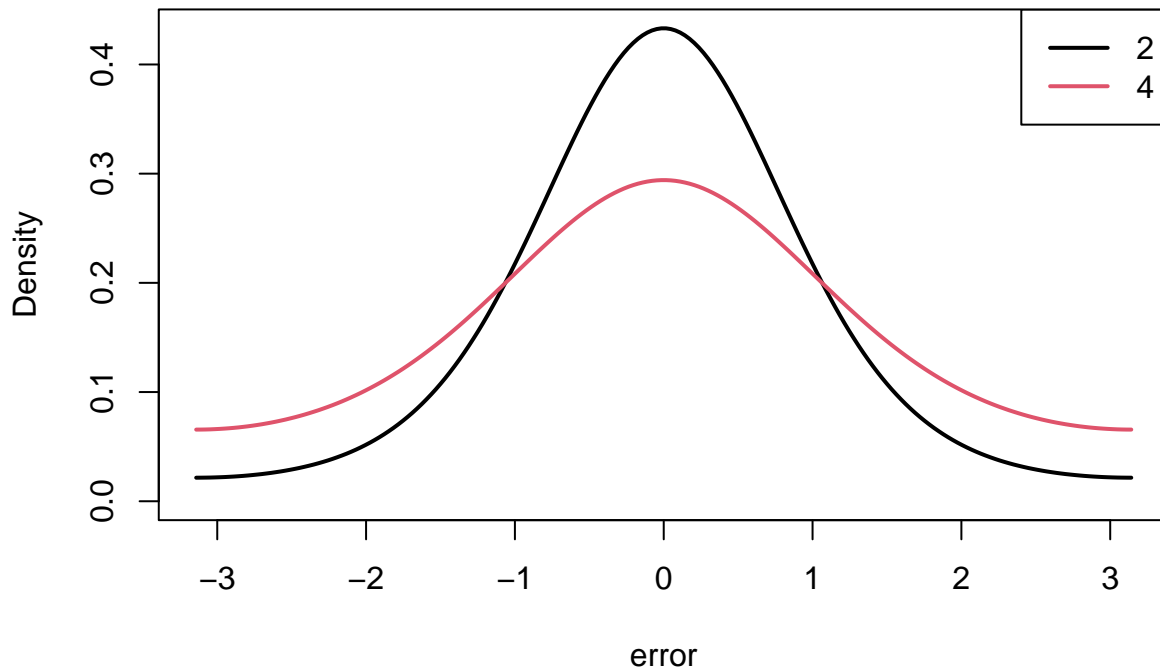
We therefore have a single parameter,  $\kappa_1$ .

```
ll_mB <- function(response_errors, setSizes, params) {  
  kappa_1 <- params[1]  
  
  nTrials <- length(response_errors) # N  
  ll <- rep(0, nTrials)  
  
  for (i in 1:nTrials) {  
    M <- setSizes[i]  
    kappa <- kappa_1 / M  
    ll[i] <- log(dvm(response_errors[i], mu=0, kappa=kappa))  
  }  
  ll  
}
```

Let's plot the likelihood:

```
e <- seq(-pi, pi, length = 200)  
  
kappa <- 3  
  
setSizes <- c(2, 4)  
densities <- sapply(setSizes, function(s) exp(ll_mB(e, setSizes=rep(s, 200), c(kappa))))  
  
plot(as.numeric(e), densities[,1], type = "l", lwd = 2,  
     ylim = c(0, max(densities)), xlab = "error", ylab = "Density",  
     main = "likelihood for model B at different set sizes")  
for(i in 2:length(setSizes)){  
  lines(as.numeric(e), densities[,i], lwd = 2, col = i)  
}  
legend("topright", legend = sapply(setSizes, function(s) paste(s)),  
     col = 1:length(setSizes), lwd = 2)
```

## likelihood for model B at different set sizes



## Recovering parameters

A first thing we might want to check is whether our models are well specified to correctly estimate the parameters underlying the data. We can evaluate this on fake data, for which we know the correct parameter value.

- (1) generate data from our models with known parameter values
- (2) fit the model to the generated data and check whether the estimated parameters correspond to the ones we used in (1)

We first need data generators for our models. This mostly just involves replacing the density function with a sampler:

### data generators

```
wrap_angle <- function(x) ((x + pi) %% (2*pi)) - pi + 0.0

# generates data from model A
r_mA <- function(setSizes, params) {
  K <- params[2]
  kappa <- params[1]

  nTrials <- length(setSizes) # N
  responses <- rep(0, nTrials)

  for (i in 1:nTrials) {
    M <- setSizes[i]

    is_in_memory = 1
    if (K < M) {
```



```

    is_in_memory <- rbinom(n = 1, size = 1, prob = K/M)
  }

  if (is_in_memory) {
    responses[i] <- rvm(1, 0, kappa)
  } else {
    responses[i] <- runif(1, min=-pi, max=pi)
  }
}
responses
}

# generates data from model B
r_mB <- function(setSizes, params) {
  kappa_1 <- params[1]

  nTrials <- length(setSizes) # N
  responses <- rep(0, nTrials)

  for (i in 1:nTrials) {
    M <- setSizes[i]
    kappa <- kappa_1 / M
    responses[i] <- rvm(1, 0, kappa)
  }
  responses
}

```

Let's plot 100 trials (50 trials for two set sizes) for each model:

```

N = 100
setSizeLevels <- c(2,4)
setSizes <- rep(setSizeLevels, each=N/length(setSizeLevels))

params_A <- c(5, 3)
response_errors_A <- r_mA(setSizes, params_A)

params_B <- c(5)
response_errors_B <- r_mB(setSizes, params_B)

df_A <- data.frame(response = wrap_angle(response_errors_A), model = "A", setSize = factor(setSizes))
df_B <- data.frame(response = wrap_angle(response_errors_B), model = "B", setSize = factor(setSizes))
df_AB <- bind_rows(df_A, df_B)

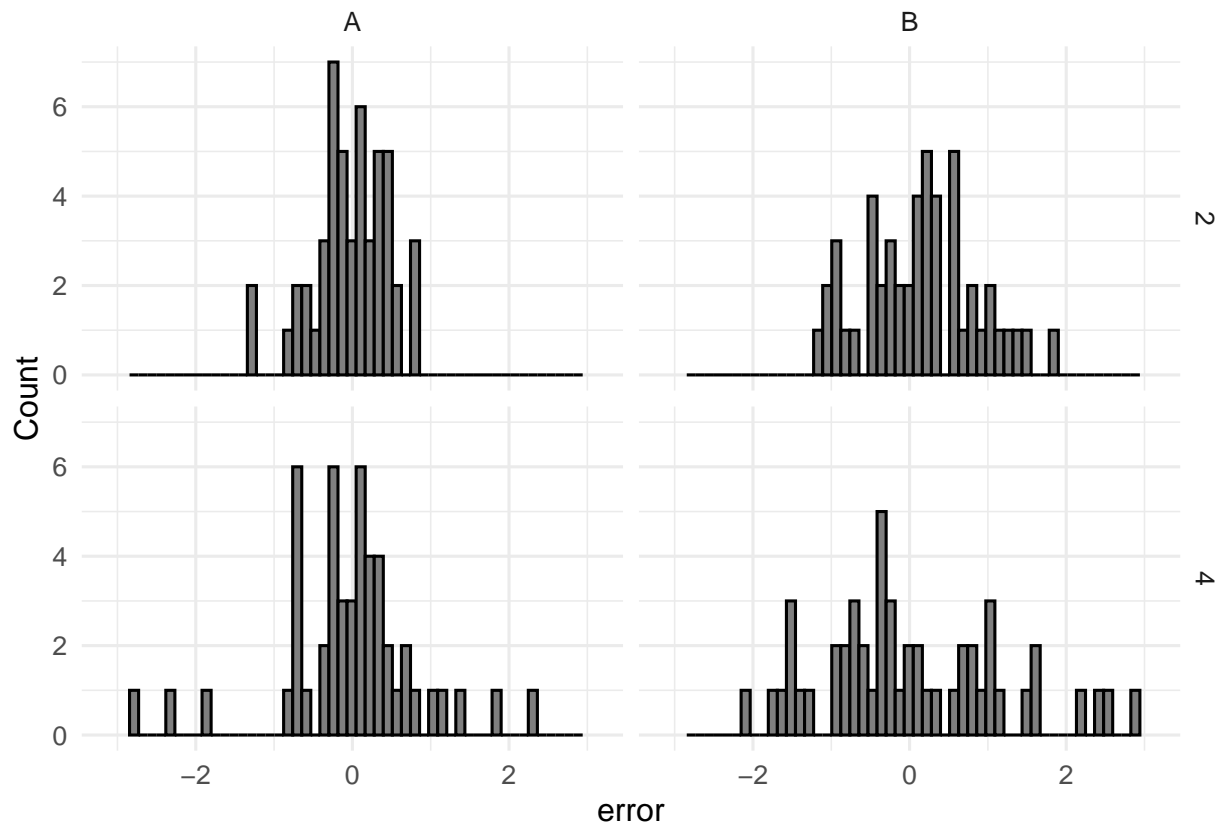
pA <- ggplot(df_AB %>% filter(model == "A"), aes(x = response)) +
  geom_histogram(bins = 50, fill = "gray40", color = "black") +
  xlim(-pi, pi) +
  labs(title = "Model A", x = "error", y = "Count") +
  theme_minimal()

p <- ggplot(df_AB, aes(x = response)) +
  geom_histogram(bins = 50, fill = "gray50", color = "black") +
  coord_cartesian(xlim = c(-pi, pi)) +
  labs(x = "error", y = "Count") +
  theme_minimal(base_size = 12) +

```

```
facet_grid(setSize ~ model)
```

p



### Parameter recovery

Let's only consider Model B for parameter recovery here. Model B has a single parameter  $\kappa_1$ .

We need a way to fit model B. We will use the `optim()` function from R.

```
negLL <- function(ll_fun, response_errors, setSizes) {
  function(params) {
    lls <- ll_fun(response_errors, setSizes, params)
    sum(-lls)
  }
}

fit_modelB <- function(response_errors, setSizes) {
  optim(par=c(4), fn=negLL(ll_mB, response_errors, setSizes), method="BFGS")
}

# check: is it working?
optimal_kappa1 <- fit_modelB(response_errors_B, setSizes)$par
```

Now we can look for different “true” parameters  $\kappa_1$  and a particular experiment (that is: number of trials, presented set sizes) recovers the correct parameter.

```
# Our "experiment":
N_per_setsize <- 20
```

```

setSizeLevels <- c(2, 4)
N <- N_per_setsize * length(setSizeLevels)
setSizeLevels <- rep(setSizeLevels, each=N/length(setSizeLevels))

# True kappas that we are going to test
true_kappas <- c(2, 5, 10)

# how often we are going to repeat the simulation
n_rep <- 500

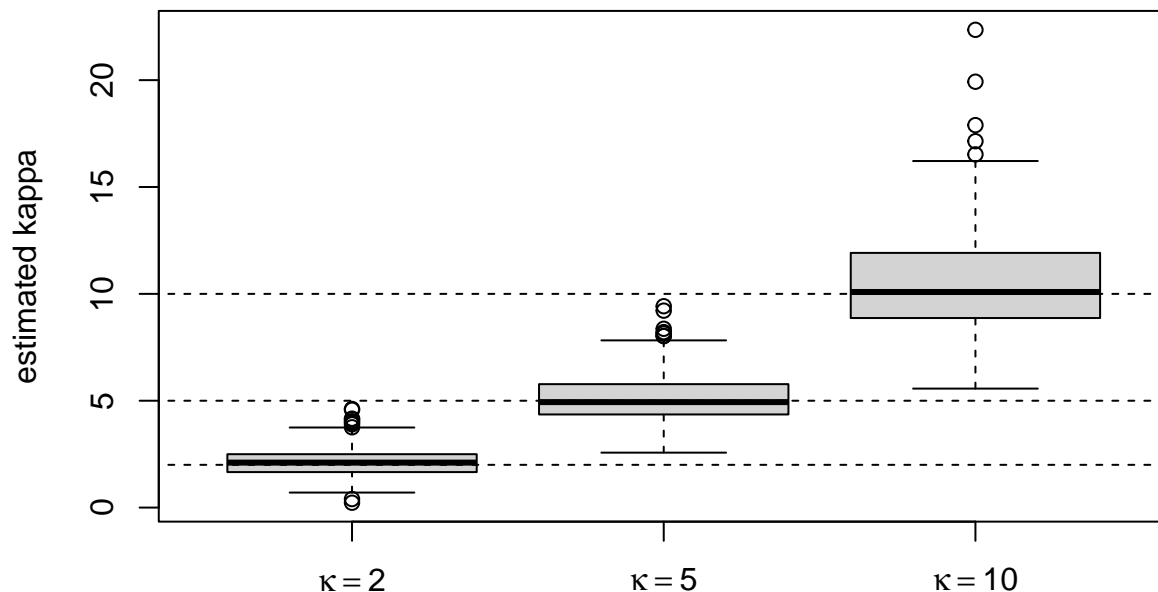
res <- matrix(NA, nrow = n_rep, ncol=length(true_kappas))

for (r in 1:n_rep) {
  for (i in 1:length(true_kappas)) {
    # simulate responses
    response_errors_B <- r_mB(setSizes, c(true_kappas[i]))
    # fit model to the responses and obtain best parameter
    suppressWarnings({
      optimal_kappa1 <- fit_modelB(response_errors_B, setSizes)$par
    })
    res[r,i] <- optimal_kappa1
  }
}

xlab <- as.expression(lapply(true_kappas, function(x) bquote(kappa == .(x))))

boxplot(res, names=xlab, ylab="estimated kappa")
for (tk in true_kappas) {
  lines(c(0,4), c(tk,tk), lty=2)
}

```



**Question 3:** What does this plot tell us?

**Solution 3:** The horizontal dotted lines are the true kappas we used to simulate the data. The boxplots show the distribution of ML-fitted kappas. The dotted lines fall onto the median of the boxplot. Hence, there is no bias - on average the model recovers the true kappa. The variance increases for larger kappas. At these value,

we may get unlucky and be quite far off from the true  $\kappa$ .

**Question 4:** What could we do if we wanted our parameter estimates to be closer to the correct value?

**Solution 4:** One obvious thing to do would be to increase the number of trial ( $N$ ). We could also check, whether the variability in the estimates is due to local minima during fitting and optimize our fitting procedure (better optimization algorithm or repeat fitting with multiple starting points). If this all doesn't help, we may have a situation in which the structure of the model itself doesn't allow for correctly recovering the parameter. E.g., we discussed the example of collinearity in multiple regression. If we care about recovering the correct parameter, we have to change our model.

**Question 5:** When do we need to make sure we can recover a parameter well and when would it maybe not matter too much?

**Solution 5:** If we want to interpret the parameter in any way, it is of course important that we are able to recover the correct parameter in this idealized setting. There are parameters we may not really care much about, like nuisance parameters (remember the left/right bias in the sequential sampling model of categorization).

## Model recovery

Here, we want to see whether we can correctly identify the true model.

We need to define a fitting function for model A now as well. Model A comes with a particular challenge: it involves a discrete parameter ( $K$ ) for which gradient-based methods are not well-suited or require other adaptations (i.e., continuous relaxations).

We here will fit the model for different fixed values of  $K$  and then pick the one with the overall lowest negative LL.

```
grid_K <- c(1, 2, 3, 4, 5, 6, 7, 8) # the values of K we will consider during fitting

fit_modelA <- function(response_errors, setSizes) {

  res <- {}
  res$value <- Inf
  res$par <- NA
  res$bestK <- NA

  for (i in 1:length(grid_K)) {
    ll_mA_K <- function(response_errors, setSizes, params) {
      kappa <- params[1]
      #print(paste(kappa, grid_K[i], sep=", "))
      ll_mA(response_errors, setSizes, c(kappa, grid_K[i]))
    }
    suppressWarnings({
      r<-optim(par=c(4), fn=negLL(ll_mA_K, response_errors, setSizes), method="BFGS")
    })
    if (r$value < res$value) {
      res$par <- r$par
      res$bestK <- grid_K[i]
      res$value <- r$value
    }
  }
  res
}
```

```

# Experiment
N_per_setsize <- 20
setSizeLevels <- c(2, 4, 8)
N <- N_per_setsize * length(setSizeLevels)
setSizes <- rep(setSizeLevels, each=N/length(setSizeLevels))

response_errors_A <- r_mA(setSizes, c(4, 1))
res <- fit_modelA(response_errors_A, setSizes)

print(paste("NLL", res$value, sep=" = "))

```

```
## [1] "NLL = 105.258384269297"
```

```
print(paste("kappa", res$par, sep=" = "))
```

```
## [1] "kappa = 4.70600545244699"
```

```
print(paste("bestK", res$bestK, sep=" = "))
```

```
## [1] "bestK = 1"
```

Now we have everything to test the model recovery ability.

```

# Experiment
N_per_setsize <- 25
setSizeLevels <- c(2, 4)
N <- N_per_setsize * length(setSizeLevels)
targetOrientations <- rep(0, N)
setSizes <- rep(setSizeLevels, each=N/length(setSizeLevels))

n_rep <- 100

# ----

# Case model A is correct
mA_true_kappa <- 4
mA_true_K <- 3

NLL_Atrue <- matrix(NA, nrow=2, ncol=n_rep)

for (r in 1:n_rep) {
  response_errors_A <- r_mA(setSizes, c(mA_true_kappa, mA_true_K))
  suppressWarnings({
    resA <- fit_modelA(response_errors_A, setSizes)
    resB <- fit_modelB(response_errors_A, setSizes)
  })
  NLL_Atrue[1, r] <- resA$value
  NLL_Atrue[2, r] <- resB$value
}

```

```

# Case model B is correct
mB_true_kappa1 <- 3
NLL_Btrue <- matrix(NA, nrow=2, ncol=n_rep)
for (r in 1:n_rep) {
  response_errors_B <- r_mB(setSizes, c(mB_true_kappa1))

```

```

suppressWarnings({
  resA <- fit_modelA(response_errors_B, setSizes)
  resB <- fit_modelB(response_errors_B, setSizes)
})
NLL_Btrue[1, r] <- resA$value
NLL_Btrue[2, r] <- resB$value
}

print(paste("Case model A is true - percentage model A has lower NLL than model B: ", 100*mean(NLL_Atrue < NLL_Btrue)))
## [1] "Case model A is true - percentage model A has lower NLL than model B: 79%"
print(paste("Case model B is true - percentage model B has lower NLL than model A: ", 100*mean(NLL_Btrue < NLL_Atrue)))
## [1] "Case model B is true - percentage model B has lower NLL than model A: 39%"

Remember there is an issue with NLL (what is it?). Let's look at AIC instead:

n_paramsA <- 2
n_paramsB <- 1

AIC_Atrue <- matrix(NA, nrow=2, ncol=n_rep)
AIC_Atrue[1,] <- 2 * NLL_Atrue[1,] + 2*n_paramsA
AIC_Atrue[2,] <- 2 * NLL_Atrue[2,] + 2*n_paramsB

AIC_Btrue <- matrix(NA, nrow=2, ncol=n_rep)
AIC_Btrue[1,] <- 2 * NLL_Btrue[1,] + 2*n_paramsA
AIC_Btrue[2,] <- 2 * NLL_Btrue[2,] + 2*n_paramsB

print(paste("Case model A is true - percentage model A has lower AIC than model B: ", 100*mean(AIC_Atrue < AIC_Btrue)))
## [1] "Case model A is true - percentage model A has lower AIC than model B: 60%"
print(paste("Case model B is true - percentage model B has lower AIC than model A: ", 100*mean(AIC_Btrue < AIC_Atrue)))
## [1] "Case model B is true - percentage model B has lower AIC than model A: 86%"

```

Note, AIC solves some problems of NLL, but not necessarily in a very satisfactory way as the number of parameters is a very crude measure of model complexity (cf. our discussions on predictive scope and marginal likelihood).

**Question 6:** Are we happy with model recovery? What can we do to improve it?

**Solution 6:** *We are looking at an ideal scenario, in which the data comes from either of the models at hand. In reality, the scenario is much less ideal: the data might be more noisy and a neither of our two models is going to be exactly identical with the true data generating process. Model recovery here is above chance, but it could be better. One obvious thing to do is to increase the number of trials. We can also think of changing the experimental design (e.g., more set sizes). We have chosen the true underlying data generating parameters rather ad-hoc here. It is important to choose these to be in the range of what we expect in the real data. One way to do this is to fit the models to real data first and use the estimated parameters for model recovery simulations.*

## Optimizing your experiment

**Question 7:** Suppose we don't want to bother participants for too long and have a fixed budget of trials per person. Describe how you would go about to determine which experiment to best run (no coding needed)?

**Solution 7:** *This is a question about experimental design: can we find a set of conditions that maximizes our ability to differentiate between our two models? One could evaluate model recovery performance for different*

designs (e.g., more set size levels). This approach can be formalized and automatized with approaches from optimal experimental designs.

## Fitting to real data

Ideally at this point, you'd conduct your own experiment - informed by model recovery and thoughts about a good experimental design that best differentiates between the two models.

We will here use the data from the beginning instead.

**Question 8:** Which model best explains the empirical data?

**Solution 8:** *Model A seems to fit the data better.*

```
resA <- fit_modelA(df$error, df$setsize)
resB <- fit_modelB(df$error, df$setsize)

NLL_A <- resA$value
NLL_B <- resB$value

AIC_A <- 2*NLL_A + 2*n_paramsA
AIC_B <- 2*NLL_B + 2*n_paramsB

print(paste("Model A: ", "kappa=", round(resA$par,2), " K=", resA$bestK, " NLL=", round(resA$value,2),
## [1] "Model A: kappa=4.15 K=3 NLL=12773.64 AIC=25551.28"
print(paste("Model B: ", "kappa=", round(resB$par,2), " NLL=", round(resB$value,2), " AIC=", round(AIC_B,2),
## [1] "Model B: kappa=7.17 NLL=13280.36 AIC=26562.72"
```

**Question 9 (OPTIONAL):** Does the better model fit the data well? Is there something systematic about the misfit?

**Solution 9:** *Let's plot both model fits overlayed onto the data:*

```
# study the best model where it may not fit the data well.

setSize <- c(2, 4)

par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

for(model in c("A", "B")) {
  for(s in setSize) {
    errs <- df$error[df$setsize == s]
    h <- hist(errs, breaks = 200,
              freq = FALSE, #use density instead of counts
              xlim = c(-pi, pi),
              main = paste("Model", model, "- SetSize", s),
              xlab = "error", ylab = "Density")

    e <- h$mids

    if(model == "A") {
      dens <- exp(ll_mA(e,
                        setSize = rep(s, length(e)),
                        c(resA$par, resA$bestK)))
    } else {
```

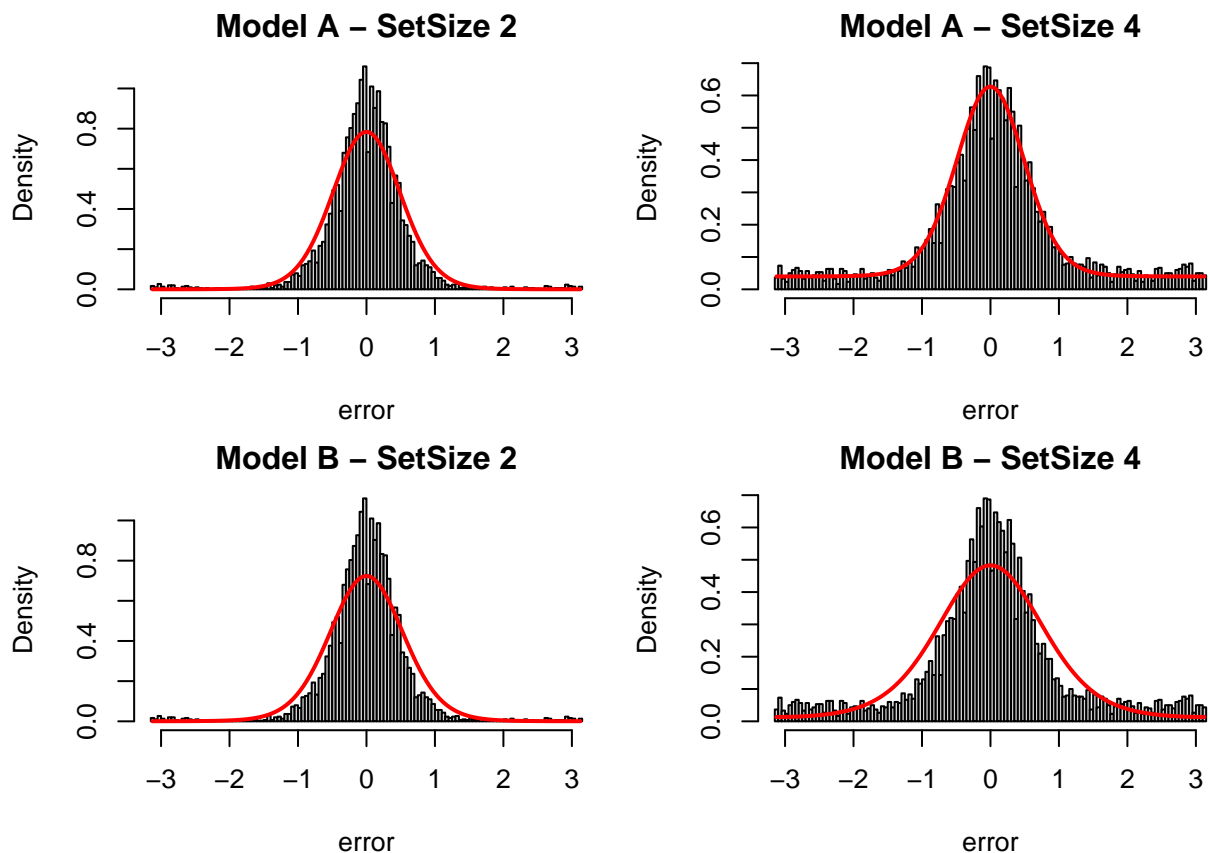
```

dens <- exp(ll_mB(e,
                  setSizes = rep(s, length(e)),
                  c(resB$par)))
}

# rescale likelihood to histogram density scale
area <- sum(dens) * (e[2] - e[1])
dens_scaled <- dens / area

# overlay likelihood curve
lines(e, dens_scaled, lwd = 2, col="red")
}

```



We will focus on Model A as our winning model. The histograms give a first indication, that set size 2 is not particularly well captured. Let's look at the deviations of the likelihood from the histogram densities for both set sizes.

```

h2 <- hist(df$error[df$setsize == 2], breaks = 200, plot=F)
h4 <- hist(df$error[df$setsize == 4], breaks = 200, plot=F)

e <- h2$mids

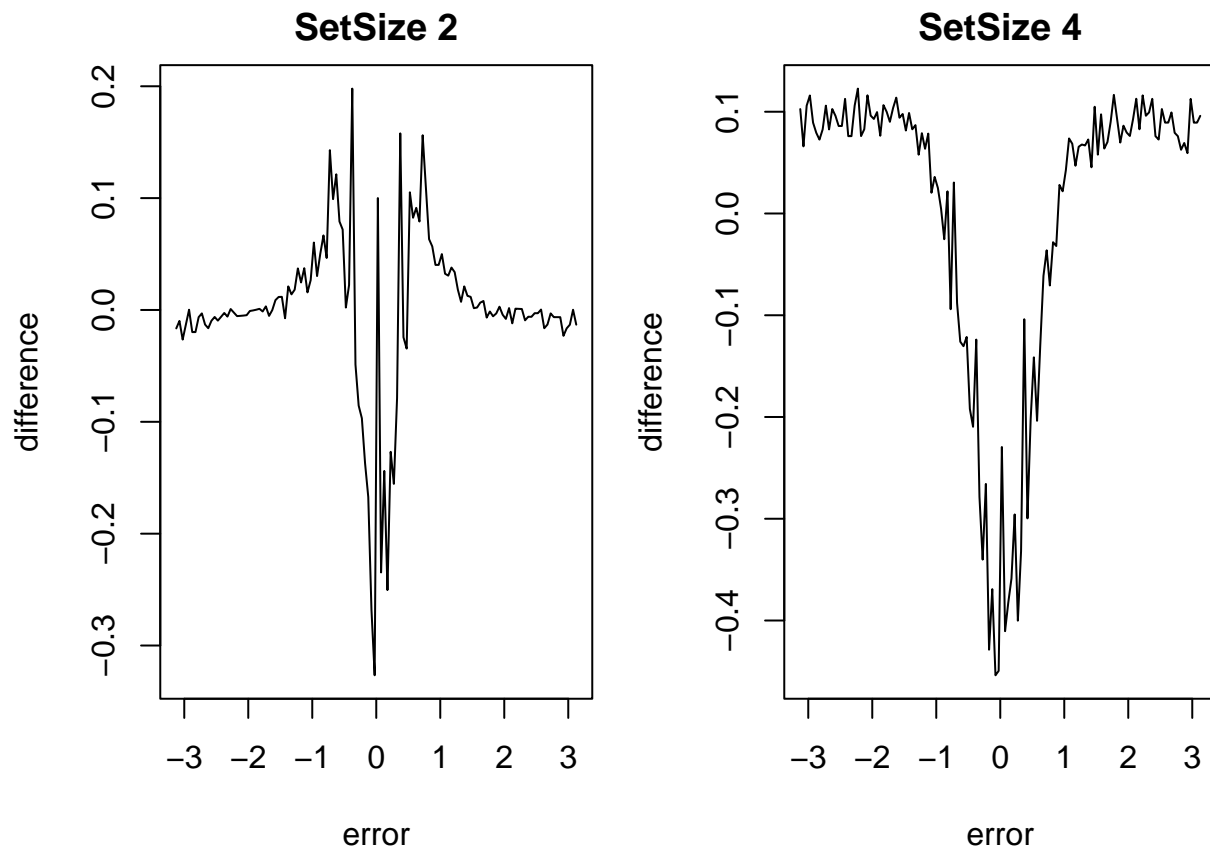
densA2 <- exp(ll_mA(e, setSizes = rep(2, length(e)),
                  c(resA$par, resA$bestK)))
densA4 <- exp(ll_mA(e, setSizes = rep(24, length(e)),
                  c(resA$par, resA$bestK)))

```



```
par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))

plot(h2$mids, densA2-h2$density, type="l",
     xlab="error", ylab="difference", main="SetSize 2")
plot(h4$mids, densA4-h4$density, type="l",
     xlab="error", ylab="difference", main="SetSize 4")
```



It seems like the model captures the tails reasonably well (perhaps slightly overestimating the tails in SetSize4). The model underestimates the amount of high precision responses (that is response errors close to zero) and overestimates the amount of responses at the flanks of the distribution (around  $\pm 1$ ). This indicates, that a Gaussian / vonMises distribution is not capturing the errors appropriately. van den Berg et al. (2012) showed that this particular deviation from a vonMises distribution can be explained by trial-by-trial variability of the memory precision  $\kappa$ . Remember however, that we pooled all subjects in our analysis. The misfit might also be explainable by differences of memory precision across participants.

Our analysis here compares the data to the best fitting model at its ML parameter estimates. Suppose instead of ML, we would perform Bayesian parameter estimation (for which we need priors over our parameters). Such a model wouldn't be as easy to visualize any more, because we have an infinite number of likelihoods, each with a weight according to its parameters' probability under the posterior. What we could do instead is for each sampled response error, we first sample parameters from the posterior and then sample the response error from the likelihood conditioned on the sampled parameters. This yields the posterior predictive distribution. The posterior predictive can then be compare against the actual data and scrutinized for unexpected/notable patterns, which may give us pointers for model adaptations and highlight strength and weaknesses of the model.

## References

- Luck, S. J., & Vogel, E. K. (1997). The capacity of visual working memory for features and conjunctions. *Nature*, 390(6657), 279–284. <https://doi.org/10.1038/36846>
- Peters, B., Rahm, B., Kaiser, J., & Bledowski, C. (2019). Differential trajectories of memory quality and guessing across sequential reports from working memory. *Journal of Vision*, 19(7), 1–14. <https://doi.org/10.1167/19.7.3>
- van den Berg, R., Shin, H., Chou, W.-C., George, R., & Ma, W. J. (2012). Variability in encoding precision accounts for visual short-term memory limitations. *Proceedings of the National Academy of Sciences*, 109(22), 8780–8785. <https://doi.org/10.1073/pnas.1117465109>
- van den Berg, R., Awh, E., & Ma, W. J. (2014). Factorial comparison of working memory models. *Psychological Review*, 121(1), 124–149. <https://doi.org/10.1037/a0035234>