glambox

Felix Molter, Armin W. Thomas

Nov 07, 2019

CONTENTS

GLAMbox is a Python toolbox for investigating the association between gaze allocation and decision behaviour, and applying the Gaze-weighted Linear Accumulator Model (Thomas, Molter et al., 2019).

See the BioRxiv preprint for detailed background, model description and example applications.

CHAPTER

ONE

INSTALLATION

GLAMbox is written for Python 3.7 and requires a working Python environment running on your computer. We recommend to install the Anaconda Distribution (available for all major platforms). With the Python environment fully set up, the GLAMbox module can be installed from the command line using pip:

```
pip install glambox
```

This command also installs all of GLAMbox's dependencies, which are listed in the `requirements.txt` file in the Github repository.

CHAPTER

TWO

QUICKSTART

Fitting the GLAM to a dataset can be done in just a few lines of code:

```
import glambox as gb
import pandas as pd

# load dataset (format must be GLAMbox compatible, of course)
data = pd.read_csv('data.csv')

# create the GLAM model object
model = gb.GLAM(data)

# build the PyMC3 model
model.make_model(kind='individual')

# perform MCMC sampling
model.fit()

# inspect parameter estimates
print(model.estimates)

# predict data using MAP estimates, save predictions
model.predict()
model.prediction.to_csv('prediction.csv')
```

A more detailed overview of the available functions can be found in the *Basic Usage section* and the *API Reference*.

CHAPTER

THREE

APPLICATION EXAMPLES

This documentation includes the three usage examples outlined in the BioRxiv preprint with full code. The original Jupyter notebook files for these examples can be found in the `examples` folder in the Github repository. When downloaded and run with Jupyter (also included in the Anaconda Python distribution), the notebooks can be ran interactively.

3.1 Example 1: Individual gaze biases

In this example, we demonstrate individual model fitting, model comparisons between model variants, and out-of-sample prediction.

3.2 Example 2: Hierarchical parameter estimation

In the second example, we demonstrate how to setup a hierarchical model with multiple groups, and compare parameter estimates between groups.

3.3 Example 3: Parameter Recovery

In the last example, we demonstrate how to perform a basic parameter recovery analyis for a given dataset, using GLAMbox.

3.3.1 Basic usage

Data format, the `GLAM` class

The core functionality of the GLAMbox is implemented in the `GLAM` model class. To apply the GLAM to data, an instance of the model class needs to be instantiated and supplied with the experimental data, first:

```
import glambox as gb
glam = gb.GLAM(data=data)
```

The data must be a pandas (McKinney, 2010) DataFrame with one row per trial, containing the following variable entries:

- `subject`: Subject index (`int`, starting with 0)
- `trial`: Trial index (`int`, starting with 0)

- `choice`: Chosen item (`int`, items should be 0, 1, . . . , N)
- `rt`: Response time (`float`, in seconds)
- for each item *i* in the choice set:
  - `item_value_i`: The item value (`float`)
  - `gaze_i`: The fraction of total time in this trial that the participant spent looking at this item (`float`, between 0 and 1)
- additional variables coding groups or conditions (`str` or `int`)

For reference, the first two rows of a pandas DataFrame ready to be used with GLAMbox could look like this:

Next, the respective PyMC3 model, which will later be used to estimate the model's parameters, can be built using the `make_model` method. Here, the researcher specifies the kind of the model: `'individual'` if the parameters should be estimated for each subject individually, `'hierarchical'` for hierarchical parameter estimation, or `'pooled'` to estimate a single parameter set for all subjects. At this stage, the researcher can also specify experimental parameter dependencies: For example, a parameter could be expected to vary between groups or conditions. In line with existing modeling toolboxes (e.g., Voss & Voss, 2007; Wiecki, Sofer, Frank, 2013) dependencies are defined using the `depends_on` argument. `depends_on` expects a dictionary with parameters as keys and experimental factors as values (e.g., `depends_on=dict(v='speed')` for factor `'speed'` with conditions `'fast'` and `'slow'` in the data). The toolbox internally handles within- and between subject designs and assigns parameters accordingly. If multiple conditions are given for a factor, one parameter will be designated for each condition. Finally, the `make_model` method allows parameters to be fixed to a specific value using the `*_val` arguments (e.g., `gamma_val=1` for a model without gaze bias). If parameters should be fixed for individual subjects, a list of individual values needs to be passed.

```
model.make_model(kind='individual',
                 depends_on=dict(v='speed'),
                 gamma_val=1)
```

Inference

Once the PyMC3 model is built, parameters can be estimated using the `fit` method:

```
model.fit(method='MCMC',
          tune=5000,
          draws=5000)
```

The `fit` method defaults to Metropolis Hastings Markov-Chain-Monte-Carlo (MCMC) sampling, but also allows for Variational Inference.

Accessing parameter estimates

After parameter estimation is completed, the resulting estimates can be accessed with the `estimates` attribute of the GLAM model instance. This returns a table with one row for each set of parameter estimates for each individual and condition in the data. For each parameter, a maximum a posteriori (MAP) estimate is given, in addition to the 95% Highest-Posterior Density Interval (HPD). If the parameters were estimated hierarchically, the table also contains estimates of the group-level parameters.

Comparing parameters between groups or conditions

Parameter estimates can be compared between different experimental groups or conditions (specified with the `depends_on` keyword when calling `make_model`) using the `compare_parameters` function from the

`analysis` module. It takes as input the fitted GLAM instance, a list of parameters (`'v'`, `'s'`, `'gamma'`, `'tau'`), and a list of pairwise comparisons between groups or conditions. The comparison argument expects a list of tuples (e.g., `[('group1', 'group2'), ('group1', 'group3')]`). For example, given a fitted model instance (here `glam`) a comparison of the γ parameter between two groups (`group1` and `group2`) can be computed as:

```
from gb.analysis import compare_parameters
comparison = compare_parameters(model=glam,
                                parameters=['gamma'],
                                comparisons=[('group1', 'group2')])
```

The function then returns a table with one row per specified comparison, and columns containing the mean posterior difference, percentage of the posterior above zero, and corresponding 95% HPD interval. If supplied with a hierarchical model, the function computes differences between group-level parameters. If an individual type model is given, it returns comparison statistics for each individual.

Comparisons can be visualized using the `compare_parameters` function from the `plots` module. It takes the same input as its analogue in the `analysis` module. It plots posterior distributions of parameters and the posterior distributions of any differences specified using the `comparisons` argument. For a usage example and plot see usage example 2.

Comparing model variants

Model comparisons between multiple GLAM variants (e.g., full and restricted variants) can be performed using the `compare_models` function, which wraps the function of the same name from the PyMC3 library. The `compare_models` function takes as input a list of fitted model instances that are to be compared. Additional keyword arguments can be given and are passed on to the underlying PyMC3 `compare` function. This allows the user, for example, to specify the information criterion used for the comparison via the `ic` argument (`'WAIC'` or `'LOO'` for Leave-One-Out cross validation). It returns a table containing an estimate of the specified information criterion, standard errors, difference to the best-fitting model, standard error of the difference, and other output variables from PyMC3 for each inputted model (and subject, if individually estimated models were given). We refer the reader to usage example 1 for the full code and exemplary output from the `compare_models` function.

Predicting choices and response times

Choices and RTs can be predicted with the GLAM by the use of the `predict`method:

```
model.predict(n_repeats=50)
```

For each trial of the dataset that is attached to the model instance, this method predicts a choice and RT using the previously determined MAP parameter estimates. To obtain a stable estimate of the GLAM's predictions, as well as the noise contained within them, it is recommended to repeat every trial multiple times during the prediction. The number of trial repeats can be specified with the `n_repeats` argument. After the prediction is completed, the predicted data can be accessed with the `prediction` attribute of the model.

References

- McKinney, W. (2010, June). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51-56).
- Thomas, A. W., Molter, F., Krajbich, I., Heekeren, H. R., & Mohr, P. N. (2019). Gaze bias differences capture individual choice behaviour. Nature human behaviour, 3(6), 625.

- Voss, A., & Voss, J. (2007). Fast-dm: A free program for efficient diffusion model analysis. Behavior Research Methods, 39(4), 767-775.
- Wiecki, T. V., Sofer, I., & Frank, M. J. (2013). HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python. Frontiers in neuroinformatics, 7, 14.

3.3.2 API Reference

The GLAM class

class GLAM(*data=None*, *name=None*)

Bases: `object`

GLAM model instance that includes data, pymc3.model.Model instance, trace, parameter estimates, fit indices and predictions.

Parameters

- **data** (*pandas.DataFrame*) – DataFrame containing the experimental data. Each row corresponds to one trial. Must include the following columns:
  - *subject* (int, consecutive, starting with 0)
  - *trial* (int, starting with 0)
  - *choice* (int, items should be 0, 1, . . . , N)
  - *rt* (float, in seconds)
  - additional variables coding groups or conditions (str or int)

  For each item *i* in the choice set:
  - *item_value_i*: The item value (float, best on a scale between 1 and 10)
  - *gaze_i*: The fraction of total trial time the item was looked at in the trial (float, between 0 and 1)
- **name** (*str*) – A name for the model. Useful if multiple models are fitted and compared.

compute_waic()

Compute WAIC for all appended PyMC3 models.

Returns Adds WAIC to GLAM model object.

Return type None

exchange_data(*new_data*, *verbose=True*)

Exchange GLAM model data. Useful for out-of-sample predictions using fitted parameters.

Parameters

- **new_data** (*pandas.DataFrame*) – new data to exchange old data with
- **verbose** (*bool, optional*) – Toggle verbosity.

Returns *new_data* replaces *data* attribute of GLAM model object

Return type None

fit(*method='MCMC'*, ***kwargs*)

Perform parameter estimation of the model.

Parameters **method** (*string ['MCMC', 'VI'], optional*) – specifies fitting method to use, can be either 'MCMC' for MCMC-sampling or 'VI' for variational inference. Defaults to 'MCMC'.

Returns Adds *trace* and well as *estimates* to GLAM model object

Return type None

make_model(*kind='individual'*, *depends_on={'gamma': None*, *'s': None*, *'t0': None*, *'tau': None*, *'v': None}*, *within_dependent=[]*, ***kwargs*)

Build the GLAM PyMC3 model, specifying model kind and dependencies.

Parameters

- **kind** (*str, optional*) – should be one of ['individual', 'hierarchical', 'pooled'], defaults to 'individual'
- **depends_on** (*dict, optional*) – dictionary specifying for each GLAM model parameter whether the parameter is dependent on any levels of the data e.g. {'v': 'speed'}, here one v parameter is created for each level of the 'speed' factor in the response data (factor must be encoded in data)
- **within_dependent** (*list, optional*) – list of parameter names ('v', 'gamma', 's', 'tau') each included parameter is modeled as dependent within a subject (i.e., as drawn from the same meta-distribution) only, if parameter dependency-structure specified in depends_on

Returns Adds PyMC3 *model*, *depends_on*, *within_dependent* and *design* to GLAM model object

Return type None

predict(*n_repeats=1*, *boundary=1.0*, *error_weight=0.05*, *verbose=True*)

Predict choices and RTs for included data

Parameters

- **n_repeats** (*int, optional*) – Number of repeats of each trial included in data during prediction Defaults to 1.
- **boundary** (*float, optional*) – Magnitude of decision boundary. Defaults to 1.0
- **error_weight** (*float, optional*) – float between [0, 1], defaults to 0.05 determining probability that choice and RT are drawn according to a unifrom error distribution (see manuscript)
- **verbose** (*bool, optional*) – Toggle verbosity.

Returns Adds *predictions* to GLAM model object

Return type None

simulate_group(*kind='hierarchical'*, *n_individuals=10*, *n_trials=100*, *n_items=3*, *individual_idx=None*, *stimuli=None*, *parameters=None*, *error_weight=0.05*, *error_range=(0*, *5)*, *value_range=(0*, *10)*, *label=None*, *seed=None*)

Simulate data from known parameters. Data is added to any existing attached dataframes, which allows blockwise simulation of multiple groups.

Parameters

- **kind** (*str, optional*) – Should be one of ['individual', 'hierarchical', 'pooled'], defaults to 'hierarchical'
- **n_individuals** (*int, optional*) – number of individuals to be added for this condition

- **n_trials** (*int, optional*) – number of trials per individual
- **n_items** (*int, optional*) – number of items per trial
- **individual_idx** (*array of ints, optional*) – individual indices to use (for within individual design simulation) defaults to continuous participant numbering across conditions
- **stimuli** (*DataFrame, optional*) – instead of simulating item_value and gaze data for the given number of individuals and trials, a DataFrame containing item_values, gaze_data and participant indices can be supplied. This overrides n_individuals and n_trials arguments.
- **parameters** (*dict, optional*) – dict with keys: 'v', 'gamma', 's', 't0', 'tau' if kind is individual:

  values: arrays of length n_individual

  if kind is hierarchical:

  values: dicts with keys: mu, sd, bounds values: floats for mu, sd, tuple for bounds

- **error_weight** (*float, optional*) – range: [0, 1], probability of simulating error trial with random choice and uniform RT
- **error_range** (*int tuple of length 2, optional*) – range of error RTs
- **value_range** (*tuple of length 2, optional*) – range of item value ratings to be simulated
- **label** (*str, optional*) – condition label. defaults "to condition_n"
- **seed** (*int, optional*) – np.random.seed(argument)

Returns Adds *data* to GLAM model instance

Return type None

The *analysis* and *plots* submodules

glambox.analysis

aggregate_subject_level_data(*data*, *n_items*)

Compute subject-level response characteristics on: RT, P(choose best), gaze influence score

The gaze influence score is defined as the average difference between the corrected choice probability of all positive and negative relative gaze values (see manuscript)

Parameters

- **data** (*pandas.DataFrame*) – DataFrame containing the experimental data. Each row corresponds to one trial. Must include the following columns:
  - *subject* (int, consecutive, starting with 0)
  - *trial* (int, starting with 0)
  - *choice* (int, items should be 0, 1, . . . , N)
  - *rt* (float, in seconds)
  - additional variables coding groups or conditions (str or int)

For each item *i* in the choice set:

- *item_value_i*: The item value (float, best on a scale between 1 and 10)
- *gaze_i*: The fraction of total trial time the item was looked at in the trial (float, between 0 and 1)

- **n_items** (*int*) – number of choice alternatives in the data

Returns DataFrame of subject-level response characteristics.

Return type pandas.DataFrame

aggregate_group_level_data(*data*, *n_items*)

Compute group-level response characteristics on: RT, P(choose best), gaze influence score

The gaze influence score is defined as the average difference between the corrected choice probability of all positive and negative relative gaze values (see manuscript)

Parameters

- **data** (*pandas.DataFrame*) – DataFrame containing the experimental data. Each row corresponds to one trial. Must include the following columns:
  - *subject* (int, consecutive, starting with 0)
  - *trial* (int, starting with 0)
  - *choice* (int, items should be 0, 1, . . . , N)
  - *rt* (float, in seconds)
  - additional variables coding groups or conditions (str or int)

  For each item *i* in the choice set:
  - *item_value_i*: The item value (float, best on a scale between 1 and 10)
  - *gaze_i*: The fraction of total trial time the item was looked at in the trial (float, between 0 and 1)
- **n_items** (*int*) – number of choice alternatives in the data

Returns DataFrame of group-level response characteristics

Return type pandas.DataFrame

compare_parameters(*model, parameters=['v', 's', 'gamma', 'tau'], comparisons=None, **kwargs*)

Perform comparisons between parameters and return statistics as DataFrame

Parameters

- **model** (glambox.GLAM) – Fitted glambox.GLAM instance
- **parameters** (*list of str, optional*) – List of parameters to perform comparisons on. Defaults to all model parameters.
- **comparisons** (*list of tuples, optional*) – List of comparisons between groups or conditions. Each comparison must be given as a tuple (e.g., [('A', 'B'), ('A', 'C')]) Defaults to None.

Returns Distribution statistics of parameter differences.

Return type pandas.DataFrame

compare_models(*models*, ***kwargs*)

Compares multiple fitted models.

Parameters

- **models** (*list of glambox.GLAM*) – List of fitted GLAM model instances.
- ****kwargs** (*optional*) – Additional keyword arguments to be passed to pymc3.compare

Returns DataFrame containing information criteria for each model.

Return type pandas.DataFrame

glambox.plots

plot_behaviour_aggregate(*bar_data*, *line_data=None*, *line_labels=None*, *fontsize=7*, *value_bins=7*, *gaze_bins=7*, *limits={'corrected_p_choose_best': (-1, 1), 'p_choose_best': (0, 1), 'rt': (0, None)}*)

Create a group-level aggregate plot with the following four metrics: A) RT ~ (max value - max value others) B) P(choose best) ~ (item value - max value others) C) P(choose best) ~ (item gaze - max gaze others) D) Corrected P(choose best) ~ (item gaze - max gaze others) For further details on these measures, see the manuscript

Parameters

- **bar_data** (*pandas.DataFrame*) – response data to plot as bars
- **line_data** (*list of pandas.DataFrames, optional*) – response data to plot as colored lines
- **line_labels** (*array_like, strings, optional*) – legend labels for line_data
- **fontsize** (*int, optional*) – fontsize for plotting, defaults to 7
- **value_bins** (*int or array_like, optional*) – x-axis bins for panels A - B if an int is given, this many bins will be created, defaults to 7
- **gaze_bins** (*int or array_like, optional*) – x-axis bins for panels A - B if an int is given, this many bins will be created, defaults to 7
- **limits** (*dict, optional*) – dict containing one entry for: ['rt', 'p_choose_best', 'corrected_p_choose_best'] each entry is a tuple, defining the y-limits for the respective metrics

Returns matplotlib figure and axes object

Return type Tuple

plot_behaviour_associations(*data*, *nbins=20*, *fontsize=7*, *regression=True*, *annotate=True*, *figsize=(7.086614173228346, 2.7559055118110236)*, *limits={'gaze_influence': (None, None), 'p_choose_best': (0, 1), 'rt': (0, None)}*)

Plot individual differences on the following three panels: D) p(choose best) ~ response time E) gaze influence score ~ response time F) gaze influence score ~ p(choose best)

In addition to each correlation plot, distributions of the three underlying behavioural measures are given in panels A-C: A) Response time B) P(choose best) C) Gaze influence score

For further details on these measures, see the manuscript

Parameters

- **data** (*pandas.DataFrame*) – response data

- **nbins** (*int, optional*) – defining the number of bins to use for the marginal histograms
- **fontsize** (*int, optional*) – defining the plotting fontsize, defaults to 7
- **regression** (*bool, optional*) – whether to compute and plot a linear regression fit, defaults to True
- **annotate** (*bool, optional*) – whether to add pearson's r correlation coefficient and p-value to plot, defaults to True
- **figsize** (*tuple, optional*) – size of of plotting figure
- **limits** (*dict, optional*) – dict containing one entry for: ['rt', 'p_choose_best', 'corrected_p_choose_best'] each entry is a tuple, defining the y-limits for the respective metrics

Returns matplotlib figure object, axs

Return type Tuple

plot_individual_fit(*observed*, *predictions*, *prediction_labels=None*, *colors=None*, *fontsize=7*, *alpha=1.0*, *figsize=None*, *limits={'gaze_influence': (None, None), 'p_choose_best': (0, 1), 'rt': (0, None)}*)

Plot individual observed vs predicted data on three metrics: A) response time B) p(choose best) C) gaze influence score For details on these measures, see the manuscript

Parameters

- **observed** (*pandas.DataFrame*) – observed response data
- **predictions** (*list of pandas.DataFrame*) – predicted response datasets
- **prediction_labels** (*array_like, strings, optional*) – legend labels for predictions
- **colors** (*array_like, strings, optional*) – colors to use for predictions
- **fontsize** (*int, optional*) – plotting fontsize
- **alpha** (*float, optional*) – alpha level for predictions should be between [0,1]
- **figsize** (*tuple, optional*) – matplotlib figure size
- **limits** (*dict, optional*) – dict containing one entry for: ['rt', 'p_choose_best', 'corrected_p_choose_best'] each entry is a tuple, defining the y-limits for the respective metrics

Returns matplotlib figure object, axs

Return type Tuple

compare_parameters(*model, parameters=['v', 's', 'gamma', 'tau'], comparisons=None, **kwargs*)

Plot posterior distributions of parameters and differences between groups or conditions.

Parameters

- **model** (*glambox.GLAM instance*) – A fitted GLAM instance.
- **parameters** (*list of str, optional*) – A list of parameters to be plotted. Defaults to all model parameters.
- **comparisons** (*List of tuples, optional*) – List of pairwise comparisons between groups or conditions to be plotted. Each comparison must be given as a tuple of two conditions (e.g., [('A', 'B'), ('A', 'C')])

Returns matplotlib figure object, axs

Return type Tuple

traceplot (*trace*, *varnames='all'*, *combine_chains=False*, *ref_val={}*)

A traceplot replacement, because arviz is broken. This is tested for traces that come out of individual and hierarchical GLAM fits.

Parameters

- **trace** (*PyMC.MultiTrace)*) – a trace object.
- **varnames** (*str, optional*) – list of variables to include
- **combine_chains** (*bool, optional*) – toggle concatenation of chains.
- **ref_val** (*dict, optional)*) – reference values per parameter.

Returns matplotlib figure and axes objects

Return type Tuple

3.3.3 License

[1]:
```
import os, errno

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import numpy as np
import pandas as pd
from tqdm import tqdm
import pymc3 as pm

import matplotlib.pyplot as plt
%matplotlib inline

import glambox as gb
```

[2]:
```
def make_sure_path_exists(path):
    try:
        os.makedirs(path)
    except OSError as exception:
```

(continues on next page)

(continued from previous page)

```
        if exception.errno != errno.EEXIST:
            raise
```

[3]:
```
make_sure_path_exists('examples/example_1/figures/')
make_sure_path_exists('examples/example_1/results/in_sample/traces/')
make_sure_path_exists('examples/example_1/results/in_sample/model_comparison/')
make_sure_path_exists('examples/example_1/results/out_of_sample/traces/')
make_sure_path_exists('examples/example_1/results/out_of_sample/predictions/')
```

3.3.4 Example 1: Individual gaze biases

Our first example is based on the study by Thomas and colleagues (2019). Here, the authors study the association between gaze allocation and choice behaviour on the level of the individual. In particular, they explore whether (1) gaze biases are present on the individual level and (2) the strength of this association varies between individuals. In this example, we replicate this type of individual model-based analysis, including parameter estimation, comparison between multiple model variants, and out-of-sample prediction of choice and RT data.

1. Simulating data

First, we simulate a dataset containing 30 subjects, each performing 300 simple value-based choice trials. We assume that in each trial participants are asked to choose the item that they like most out of a set of three presented alternatives (e.g., snack food items; similar to the task described in Krajbich & Rangel (2011)). While participants perform the task, their eye movements, choices and RTs are measured. Before completing the choice trials, participants were asked to indicate their liking rating for each of the items used in the choice task on a liking rating scale between 1 and 10 (with 10 indicating strong liking and 1 indicating little liking). The resulting dataset contains a liking value for each item in a trial, the participants' choice and RT, as well as the participant's gaze towards each item in a trial (describing the fraction of trial time that the participant spent looking at each item in the choice set).

[4]:
```
n_subjects = 30
subjects = np.arange(n_subjects)
n_trials = 300
n_items = 3
```

To simulate individuals' response behaviour, we utilize the parameter estimates that were obtained by Thomas et al. (2019) for the individuals in the three item choice dataset by Krajbich & Rangel (2011) (for an overview, see Fig. S1 of the manuscript). Importantly, we assume that 10 individuals do not exhibit a gaze bias, meaning that their choices are independent of the time that they spend looking at each item. To this end, we set the γ value of ten randomly selected individuals to 1. We further assume that individuals' gaze is distributed randomly with respect to the values of the items in a choice set.

[5]:
```
np.random.seed(1)

# load empirical model parameters (taken from Thomas et al., 2019)
estimates = pd.read_csv('resources/individual_estimates_sec_nhb2019.csv')
kr2011 = estimates.loc[estimates['dataset'] == 'krajbich2011']
gen_parameters = dict(v=kr2011['v'].values,
                      gamma=kr2011['gamma'].values,
                      s=kr2011['s'].values,
                      tau=kr2011['tau'].values,
                      t0=np.zeros(len(kr2011)))

# define participants with no association between gaze and choice:
```

(continues on next page)

(continued from previous page)

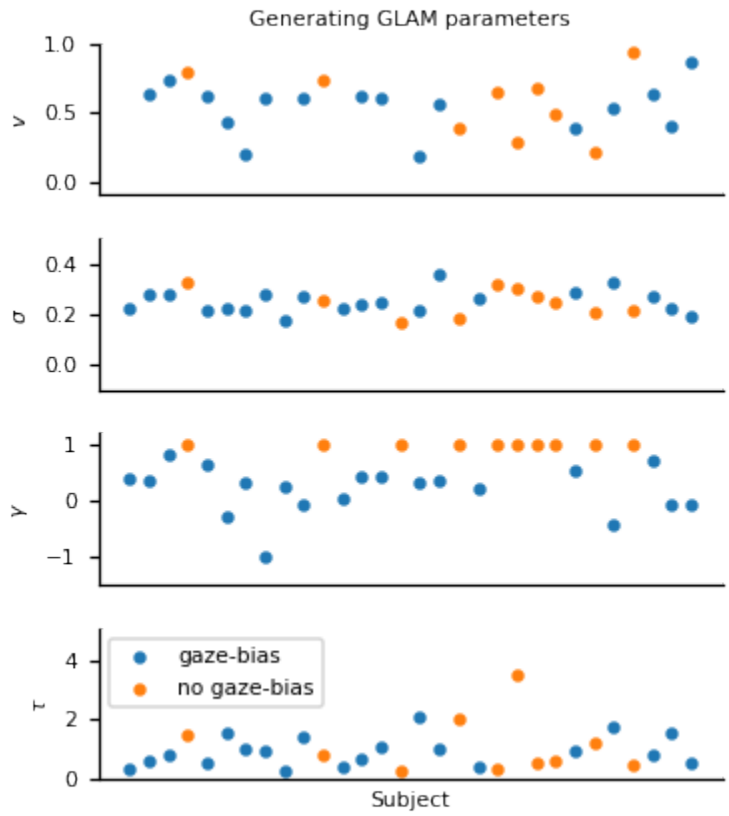
```
no_gaze_bias_subjects = np.sort(np.random.choice(n_subjects, 10, replace=False))
gaze_bias_subjects = np.array([s for s in subjects if s not in no_gaze_bias_subjects])
gen_parameters['gamma'][no_gaze_bias_subjects] = 1
```

The resulting distribution of generating model parameters looks as follows:

[6]:
```
fig, axs = plt.subplots(4, 1, figsize=gb.plots.cm2inch(9,10), dpi=110, sharex=True)

for subject_set, color, label in zip([gaze_bias_subjects,
                                      no_gaze_bias_subjects],
                                     ['C0', 'C1'],
                                     ['gaze-bias', 'no gaze-bias']):
    # v
    axs[0].scatter(subject_set,
                   gen_parameters['v'][subject_set],
                   color=color,
                   s=10)
    axs[0].set_ylabel(r'$v$', fontsize=7)

    # sigma
    axs[1].scatter(subject_set,
                   gen_parameters['s'][subject_set],
                   color=color,
                   s=10)
    axs[1].set_ylabel(r'$\sigma$', fontsize=7)

    # gamma
    axs[2].scatter(subject_set,
                   gen_parameters['gamma'][subject_set],
                   color=color,
                   s=10)
    axs[2].set_ylabel(r'$\gamma$', fontsize=7)

    # tau
    axs[3].scatter(subject_set,
                   gen_parameters['tau'][subject_set],
                   color=color,
                   label=label,
                   s=10)
    axs[3].set_ylabel(r'$\tau$', fontsize=7)
    axs[3].set_xlabel('Subject', fontsize=7)


axs[0].set_title('Generating GLAM parameters', fontsize=7)
axs[0].set_ylim(-0.1, 1)
axs[1].set_ylim(-0.1, 0.5)
axs[2].set_ylim(-1.5, 1.2)
axs[3].set_ylim(0, 5.1)
axs[-1].legend(loc='upper left', frameon=True, fontsize=7)
for ax in axs:
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)
    ax.tick_params(axis='both', which='major', labelsize=7)
    ax.set_xticks([])

fig.tight_layout()
fig.savefig('examples/example_1/figures/Supplementary_Figure_1_generating_parameters.
↪png', dpi=330)
```

(continues on next page)

(continued from previous page)



These are the subjects that we defined as having no association of gaze allocation and choice behaviour:

[7]:
```
no_gaze_bias_subjects
```

[7]:
```
array([ 3, 10, 14, 17, 19, 20, 21, 22, 24, 26])
```

We first instantiate a GLAM model instance using `gb.GLAM()` and then use its `simulate_group` method. This method requires us to specify whether the individuals of the group are either simulated individually (and thereby independent of one another) or as part of a group with hierarchical parameter structure (where the individual model parameters are drawn from a group distribution, see below). For the former, the generating model parameters (indicated in the following as `gen_parameters`) are provided as a dictionary, containing a list of the individual participant values for each model parameter:

[8]:
```
np.random.seed(2)

glam = gb.GLAM()
glam.simulate_group(kind='individual',
                    n_individuals=n_subjects,
                    n_trials=n_trials,
                    n_items=n_items,
                    parameters=gen_parameters,
                    value_range=(1, 10))
```

We can then access the simulated data as part of our GLAM model instance.

[9]:
```
data = glam.data.copy()
```

[10]:
```
data.head()
```

[10]:
```
   subject  trial  repeat  choice        rt  item_value_0    gaze_0  \
0      0.0    0.0     0.0     0.0  1.678472             9  0.649142
1      0.0    1.0     0.0     2.0  1.307034             3  0.332040
2      0.0    2.0     0.0     2.0  1.324605             3  0.423884
3      0.0    3.0     0.0     1.0  1.145805             5  0.195771
4      0.0    4.0     0.0     2.0  1.065713             8  0.209319

   item_value_1    gaze_1  item_value_2    gaze_2    condition
0             9  0.224745             7  0.126113  condition_0
1             9  0.312176             8  0.355783  condition_0
2             2  0.192437             6  0.383680  condition_0
3             5  0.454379             6  0.349851  condition_0
4             4  0.382790             7  0.407891  condition_0
```

As this example is focused on the individual level, we can further create a summary table, describing individuals' response behaviour on three behavioural metrics, using the `aggregate_subject_level_data` function from the `analysis` module. The resulting table contains individuals' mean RT, their probability of choosing the item with the highest item value from a choice set and a behavioural measure of the strength of the association between individuals' gaze allocation and choice behaviour (indicating the mean increase in choice probability for an item that was fixated on longer than the others, after correcting for the influence of the item value on choice behaviour; for further details, see Thomas et al. (2019)).

[11]:
```
np.random.seed(3)

subject_data_summary = gb.analysis.aggregate_subject_level_data(data, n_items)
```

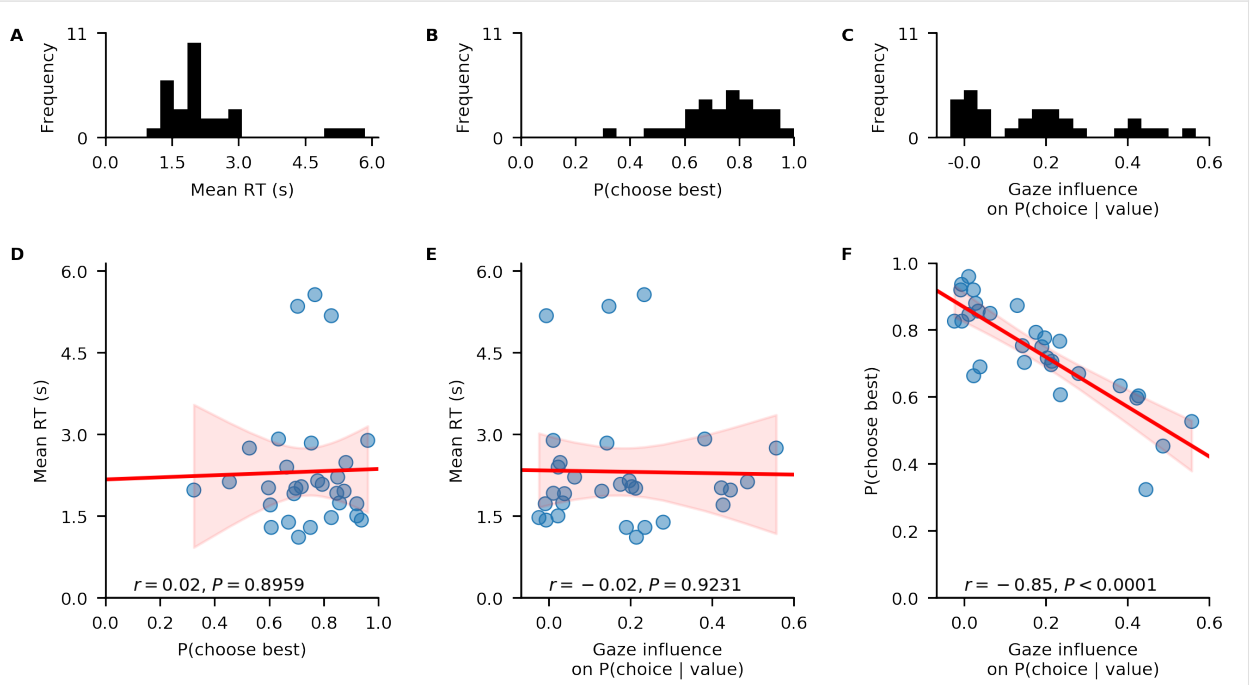
[12]:
```
subject_data_summary.head()
```

[12]:
```
                rt                                                          \
              mean       std       min       max        se        q1        q3
subject
0.0       1.292315  0.477199  0.142598  4.708733  0.027597  1.025087  1.458560
1.0       2.040758  0.878345  0.523410  6.317563  0.050796  1.413609  2.499227
2.0       1.742749  0.720414  0.137617  4.912813  0.041663  1.261841  2.002483
3.0       1.427180  0.697514  0.062293  5.375163  0.040338  0.967926  1.705945
4.0       2.217397  0.755107  0.005503  4.586606  0.043669  1.662276  2.623872

                  best_chosen gaze_influence
              iqr        mean
subject
0.0       0.433473    0.750000       0.189652
1.0       1.085618    0.716667       0.202899
2.0       0.740643    0.856667       0.033876
3.0       0.738018    0.936667      -0.006805
4.0       0.961595    0.850000       0.062939
```

2. Exploring the behavioural data

In a first step of our analysis, we explore differences in individuals' response behaviour. To this end, we plot the distributions of individuals' scores on the three behavioural metrics, and their associations, using the `plot_behaviour_associations` function implemented in the `plots` module:

[13]:
```
np.random.seed(4)

fig = gb.plots.plot_behaviour_associations(data=data)

fig.savefig('examples/example_1/figures/Figure_1_individual_differences.png', dpi=330)
```



The resulting plot shows that individuals' probability of choosing the best item, as well as the strength of their behavioural association of gaze and choice, are not associated with their mean RT (D-E). However, individuals' probability of choosing the best item increases with decreasing strength of the behavioural association of gaze and choice (F).

3. Likelihood-based model comparison

In a second step of our analysis, we want to test whether the response behaviour of each individual is better described by a decision model with or without gaze bias. To this end, we set up the two GLAM variants:

One GLAM variant that includes the gaze bias mechanism:

[14]:
```
np.random.seed(5)

glam_bias = gb.GLAM(data=data, name='glam_bias')
glam_bias.make_model(kind='individual')
```

```
Generating single subject models for 30 subjects...
```

And one without a gaze bias (as indicated by `gamma_val=1`):

[15]:
```
np.random.seed(6)

# for the no-gaze-bias variant, we set the gamma-parameter to 1, indicating no␣
↪influence of gaze allocation on choice behaviour
glam_nobias = gb.GLAM(data=data, name='glam_nobias')
glam_nobias.make_model(kind='individual', gamma_val=1)
```

```
Generating single subject models for 30 subjects...
```

Subsequently, we fit both models to the data of each individual and compare their fit by means of the Widely Applicable Information Criterion (WAIC; Vehtari et al., 2017):

The `fit` method defaults to Metropolis-Hastings MCMC sampling (for methodological details, see the Methods Section of the manuscript). The `draws` argument sets the number of samples to be drawn. This excludes the tuning (or burn-in) samples, which can be set with the `tune` argument. In addition, the `fit` method accepts the same keyword arguments as the PyMC3 sample function, which it wraps (see the PyMC3 documentation for additional details). The `chains` argument sets the number of MCMC traces (it defaults to four and should be set to at least two, in order to allow convergence diagnostics).

[16]:
```
n_tune = 5000
n_draws = 5000
n_chains = 4
```

[17]:
```
np.random.seed(7)

glam_bias.fit(method='MCMC',
              tune=n_tune,
              draws=n_draws,
              chains=n_chains)
```

```
Fitting 30 model(s) using MCMC...
  Fitting model 1 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1315.39draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 2 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1301.23draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 3 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1300.95draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 4 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1266.78draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 5 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1292.95draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 6 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1284.61draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 7 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1293.75draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 8 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1292.45draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 9 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1285.49draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 10 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1287.27draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 11 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1284.06draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 12 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1279.57draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 13 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1266.32draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 14 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:32<00:00, 1237.59draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 15 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1279.50draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 16 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1283.37draws/s]
The number of effective samples is smaller than 25% for some parameters.
```

```
  Fitting model 17 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1283.39draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 18 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1278.46draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 19 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1278.78draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 20 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1279.67draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 21 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1311.78draws/s]
The number of effective samples is smaller than 25% for some parameters.
```

```
  Fitting model 22 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1276.63draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 23 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1272.31draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 24 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1286.49draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 25 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1275.65draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 26 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1281.44draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 27 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1275.94draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 28 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1275.33draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 29 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1258.98draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 30 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1272.49draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
/!\ Automatically setting parameter precision...
```

[18]:
```
np.random.seed(8)

glam_nobias.fit(method='MCMC',
                tune=n_tune,
                draws=n_draws,
                chains=n_chains)
```

```
Fitting 30 model(s) using MCMC...
  Fitting model 1 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
```

(continues on next page)

(continued from previous page)

```
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1931.05draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 2 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1941.34draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 3 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1706.90draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 4 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1750.74draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 5 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1778.95draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 6 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1749.17draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 7 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1698.13draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 8 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1752.89draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 9 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1748.22draws/s]
The number of effective samples is smaller than 25% for some parameters.
  Fitting model 10 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1779.69draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 11 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1669.64draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 12 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1705.45draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 13 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1789.80draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 14 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1776.07draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 15 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1733.82draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 16 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1692.22draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 17 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1764.09draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 18 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1777.84draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 19 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1737.96draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 20 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1697.29draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 21 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1781.37draws/s]
The number of effective samples is smaller than 25% for some parameters.
```

```
  Fitting model 22 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1758.02draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 23 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1762.24draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 24 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1672.76draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 25 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1701.48draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 26 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
```

(continues on next page)

(continued from previous page)

```
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1765.12draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 27 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1753.78draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 28 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1703.76draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 29 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:23<00:00, 1667.33draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 30 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:22<00:00, 1775.38draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
/!\ Automatically setting parameter precision...
```

Convergence check:

[19]:
```
def check_convergence(summary, varnames=['v', 's', 'tau'],
                      n_eff_required=100, gelman_rubin_criterion=0.05):
    varnames = [varname + '__0_0' for varname in varnames]
    enough_eff_samples = np.all(summary.loc[varnames]['n_eff'] > n_eff_required)
    good_gelman = np.all(np.abs(summary.loc[varnames]['Rhat'] - 1.0) < gelman_rubin_
↪criterion)
    if not enough_eff_samples or not good_gelman:
        return False
    else:
        return True
```

[20]:
```
np.all([check_convergence(pm.summary(trace), varnames=['v', 's', 'gamma', 'tau']) for␣
↪trace in glam_bias.trace])
```

[20]:
```
True
```

[21]:
```
np.all([check_convergence(pm.summary(trace), varnames=['v', 's', 'tau']) for trace in␣
↪glam_nobias.trace])
```

[21]:
```
True
```

Saving traces and traceplots for visual inspection:

[22]:
```
for subject, subject_trace in enumerate(glam_bias.trace):
    gb.plots.traceplot(subject_trace)
    plt.savefig('examples/example_1/results/in_sample/traces/sub_{}_gaze_bias_model_
↪trace.png'.format(subject), dpi=330)
    plt.close()
    pm.trace_to_dataframe(subject_trace).to_csv(
        'examples/example_1/results/in_sample/traces/sub_{}_gaze_bias_model_trace.csv
↪'.format(subject))
```

[23]:
```
for subject, subject_trace in enumerate(glam_nobias.trace):
    gb.plots.traceplot(subject_trace)
    plt.savefig('examples/example_1/results/in_sample/traces/sub_{}_no_gaze_bias_
↪model_trace.png'.format(subject), dpi=330)
    plt.close()
    pm.trace_to_dataframe(subject_trace).to_csv(
        'examples/example_1/results/in_sample/traces/sub_{}_no_gaze_bias_model_trace.
↪csv'.format(subject))
```

After convergence has been established for all parameter traces (for details on the suggested convergence criteria, see the Methods Section of the manuscript), we perform a model comparison on the individual level, using the `compare` function:

The resulting table can be used to identify the best fitting model (indicated by the lowest WAIC score) per individual.

[24]:
```
comparison_df = gb.compare(models=[glam_bias, glam_nobias])
comparison_df
```

[24]:

	subject	model	WAIC	pWAIC	dWAIC	weight	SE	dSE	var_warn
0	0	glam_bias	523.6	5.75	0	0.94	50.25	0	0
1	0	glam_nobias	645.09	3.64	121.49	0.06	44.15	23.56	0
2	1	glam_bias	1097.86	3.69	0	1	40.32	0	0
3	1	glam_nobias	1185.02	2.85	87.16	0	38.22	18	0
4	2	glam_bias	832.43	4.02	0	1	41.15	0	0
5	2	glam_nobias	847.56	3.02	15.12	0	41.33	7.74	0
6	3	glam_nobias	690.75	2.9	0	0.79	45.73	0	0
7	3	glam_bias	691.47	3.37	0.72	0.21	45.74	2.26	0
8	4	glam_bias	927.23	3.61	0	0.99	39.4	0	0
9	4	glam_nobias	966.35	2.68	39.12	0.01	37.55	12.09	0
10	5	glam_bias	1047.15	3.85	0	0.99	42.42	0	0
11	5	glam_nobias	1460.1	2.99	412.95	0.01	33.1	30.37	0
12	6	glam_bias	1821.62	4.5	0	0.98	44.58	0	0
13	6	glam_nobias	1883.22	3.33	61.6	0.02	41.92	15.77	0
14	7	glam_bias	1042.19	3.7	0	1	43.31	0	0
15	7	glam_nobias	1354.08	2.1	311.89	0	29.78	26.01	0
16	8	glam_bias	295.23	3.34	0	1	49.43	0	0
17	8	glam_nobias	424.61	2.55	129.38	0	48.14	21.22	0
18	9	glam_bias	966.07	3.82	0	0.96	44.74	0	0
19	9	glam_nobias	1267.56	2.97	301.49	0.04	34.8	31.21	0
20	10	glam_nobias	732.99	2.81	0	1	42.6	0	0

(continues on next page)

(continued from previous page)

21	10	glam_bias	734.6	2.96	1.61	0	42.55	0.97	0
22	11	glam_bias	683.41	4.09	0	0.99	51.38	0	0
23	11	glam_nobias	879.43	2.9	196.02	0.01	44.38	26.28	0
24	12	glam_bias	957.48	4	0	0.95	42	0	0
25	12	glam_nobias	1065.61	2.76	108.13	0.05	39.32	21.55	0
26	13	glam_bias	927.51	4.13	0	0.99	45.41	0	0
27	13	glam_nobias	1049.05	3.03	121.54	0.01	42.55	21.65	0
28	14	glam_nobias	483.89	2.76	0	1	48.24	0	0
29	14	glam_bias	485.6	3.06	1.71	0	48.22	1.25	0
30	15	glam_bias	1760.66	4.02	0	0.94	43.82	0	0
31	15	glam_nobias	1899.37	3.12	138.71	0.06	40.82	25.18	0
32	16	glam_bias	1180.79	4.42	0	0.91	39.88	0	0
33	16	glam_nobias	1268.26	2.87	87.47	0.09	36.97	20.29	0
34	17	glam_nobias	935.04	3.04	0	1	42.46	0	0
35	17	glam_bias	936.51	3.18	1.47	0	42.52	1.23	0
36	18	glam_bias	747.64	4.01	0	0.96	38.14	0	0
37	18	glam_nobias	842.02	2.67	94.38	0.04	34.93	19.44	0
38	19	glam_bias	1191.53	3.91	0	0.65	38.44	0	0
39	19	glam_nobias	1192.6	2.94	1.07	0.35	37.99	3.79	0
40	20	glam_nobias	3024.94	2.17	0	1	170.6	0	0
41	20	glam_bias	3026.69	2.54	1.75	0	170.54	1.63	0
42	21	glam_nobias	904.11	2.78	0	1	42.13	0	0
43	21	glam_bias	905.02	3.14	0.91	0	42.11	1.83	0
44	22	glam_nobias	1051.61	3.29	0	1	37.47	0	0
45	22	glam_bias	1053.29	3.61	1.68	0	37.34	1.27	0
46	23	glam_bias	1432.37	4.2	0	0.95	43.26	0	0
47	23	glam_nobias	1474.78	2.99	42.41	0.05	41.99	13.38	0
48	24	glam_nobias	1695.8	3.36	0	1	39.82	0	0
49	24	glam_bias	1697.75	3.64	1.94	0	39.81	1.03	0
50	25	glam_bias	1158.37	3.84	0	1	42.29	0	0
51	25	glam_nobias	1452.21	2.67	293.85	0	29.41	28.28	0
52	26	glam_nobias	451.16	3.74	0	1	46.43	0	0
53	26	glam_bias	452.31	3.99	1.15	0	46.15	1.91	0
54	27	glam_bias	867.65	3.52	0	1	42.55	0	0
55	27	glam_nobias	895.33	2.6	27.69	0	43.06	9.67	0
56	28	glam_bias	1175.13	4.43	0	0.98	39.55	0	0
57	28	glam_nobias	1445.24	2.87	270.11	0.02	32.8	29.11	0
58	29	glam_bias	685.52	4.33	0	1	45.94	0	0
59	29	glam_nobias	1024.66	3.09	339.15	0	39.26	27.55	0

Visualising the individual WAIC differences:

[49]:
```
dWAIC = []
for subject in subjects:
    comp_s = comparison_df.loc[comparison_df['subject'] == subject]
    dWAIC_s = comp_s.loc[comp_s['model'] == 'glam_bias', 'WAIC'].values - comp_s.
↪loc[comp_s['model'] == 'glam_nobias', 'WAIC'].values
    dWAIC.append(dWAIC_s[0])
dWAIC = np.array(dWAIC)
np.save('examples/example_1/results/in_sample/model_comparison/dWAIC_in_sample.npy',␣
↪dWAIC)
dWAIC
```

[49]:
```
array([-121.49,  -87.16,  -15.13,    0.72,  -39.12, -412.95,  -61.6 ,
       -311.89, -129.38, -301.49,    1.61, -196.02, -108.13, -121.54,
          1.71, -138.71,  -87.47,    1.47,  -94.38,   -1.07,    1.75,
          0.91,    1.68,  -42.41,    1.95, -293.84,    1.15,  -27.68,
```

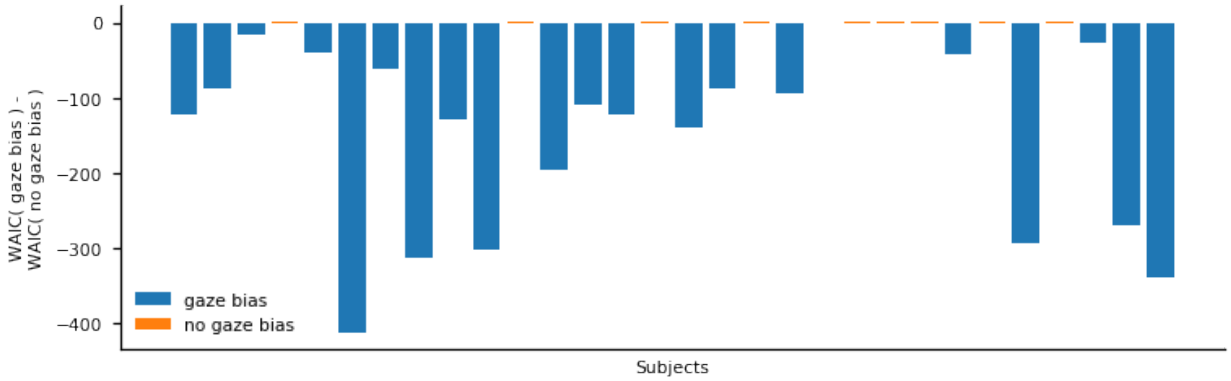
(continues on next page)

(continued from previous page)

```
       -270.11, -339.14])
```

[51]:
```
# identify subjects better described by each model variant
gaze_bias_idx = dWAIC < 0
no_gaze_bias_idx = dWAIC > 0
```

[52]:
```
fig, ax = plt.subplots(1, 1, figsize=gb.plots.cm2inch(18,6), dpi=110)

ax.bar(subjects[gaze_bias_idx], dWAIC[gaze_bias_idx], color='C0', label='gaze bias')
ax.bar(subjects[no_gaze_bias_idx], dWAIC[no_gaze_bias_idx], color='C1', label='no␣
↪gaze bias')
ax.set_xlabel('Subjects', fontsize=7)
ax.set_ylabel('WAIC( gaze bias )'+' -\n'+'WAIC( no gaze bias )', fontsize=7)
ax.legend(loc='lower left', frameon=False, fontsize=7)
ax.set_xticks([])
ax.tick_params(axis='both', which='major', labelsize=7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

fig.tight_layout()

fig.savefig('examples/example_1/figures/relative_model_comparison.png', dpi=330)
```



With this comparison, we are able to identify those participants whose response behaviour matches the assumption of gaze-biased evidence accumulation. In particular, we find that we accurately recover whether an individual has a gaze bias or not for 29 out of 30 individuals.

[53]:
```
no_gaze_bias_subjects
```

[53]:
```
array([ 3, 10, 14, 17, 19, 20, 21, 22, 24, 26])
```

[54]:
```
subjects[no_gaze_bias_idx]
```

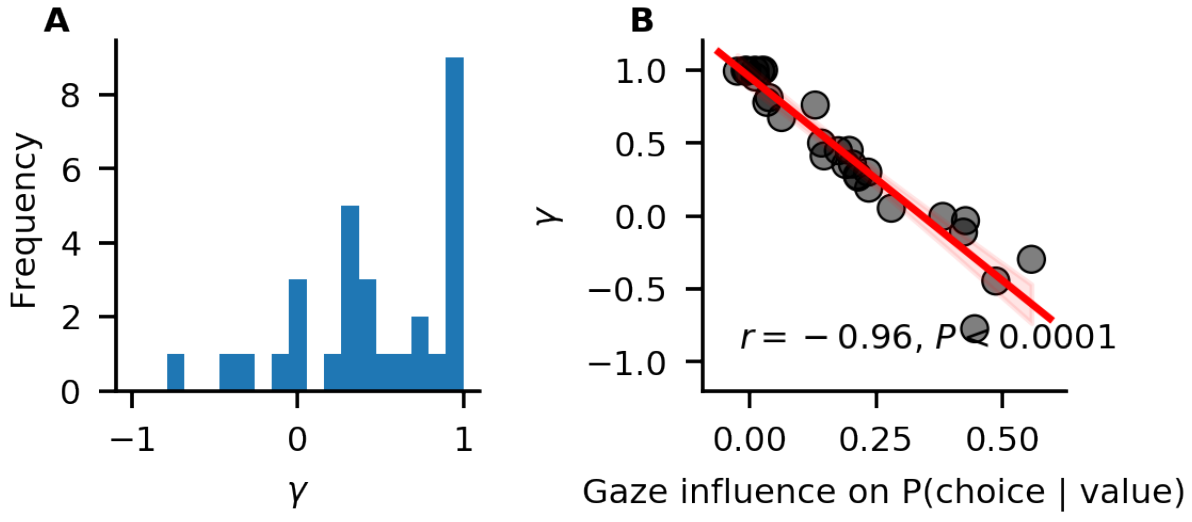
[54]:
```
array([ 3, 10, 14, 17, 20, 21, 22, 24, 26])
```

[56]:
```
[s in subjects[no_gaze_bias_idx] for s in no_gaze_bias_subjects]
```

[56]:
```
[True, True, True, True, False, True, True, True, True, True]
```

Looking at the individual parameter estimates (defined as MAP of the posterior distributions), we find that the individually fitted γ values cover a wide range between -0.8 and 1 (A), indicating strong variability in the strength of individuals' gaze bias. We also find that γ estimates have a strong negative correlation with individuals' scores on the behavioural gaze bias measure (B).

[57]:
```
np.random.seed(10)

fig, axs = plt.subplots(1, 2, figsize=gb.plots.cm2inch(9, 4.5), dpi=330)

axs[0].hist(glam_bias.estimates['gamma'], bins=np.linspace(-1., 1, 20))
gb.plots.plot_correlation(subject_data_summary['gaze_influence'],
                          glam_bias.estimates['gamma'],
                          ax=axs[1],
                          ylim=(-1.2, 1.2))

axs[0].set_ylabel('Frequency', fontsize=7)
axs[0].set_xlabel(r'$\gamma$', fontsize=7)
axs[0].tick_params(axis='both', which='major', labelsize=7)
axs[0].spines['top'].set_visible(False)
axs[0].spines['right'].set_visible(False)
axs[1].set_ylabel(r'$\gamma$', fontsize=7)
axs[1].set_xlabel('Gaze influence on P(choice | value)', fontsize=7)
axs[1].tick_params(axis='both', which='major', labelsize=7)
for ax, label in zip(axs.ravel(), list('AB')):
    ax.text(-0.2,
            1.1,
            label,
            transform=ax.transAxes,
            fontsize=7,
            fontweight='bold',
            va='top')

fig.tight_layout()

fig.savefig('examples/example_1/figures/Figure_3_gaze_bias_estimates.png', dpi=330)
```



4. Out-of-sample prediction

We have identified those participants whose response behaviour is better described by a GLAM variant with gaze-bias than one without. Yet, this analysis does not indicate whether the GLAM is a good model of individuals' response behaviour on an absolute level. To test this, we perform an out-of-sample prediction exercise.

We divide the data of each subject into even- and odd-numbered experiment trials and use the data of the even-numbered trials to fit both GLAM variants:

[58]:
```
data_even = data[(data['trial']%2)==0].copy()
data_odd = data[(data['trial']%2)!=0].copy()
```

[59]:
```
np.random.seed(11)

glam_bias.exchange_data(data_even)
glam_bias.fit(method='MCMC',
              tune=n_tune,
              draws=n_draws,
              chains=n_chains)
```

```
Replaced attached data (9000 trials) with new data (4500 trials)...
Fitting 30 model(s) using MCMC...
  Fitting model 1 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:34<00:00, 1169.99draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 2 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:32<00:00, 1223.64draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 3 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1271.52draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 4 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1279.71draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 5 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1277.87draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 6 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1268.69draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 7 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1286.30draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 8 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1279.16draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 9 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1274.89draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 10 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1281.95draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 11 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1268.59draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 12 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1271.31draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 13 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1287.66draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 14 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1276.58draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 15 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1278.21draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 16 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1279.58draws/s]
The number of effective samples is smaller than 25% for some parameters.
```

```
  Fitting model 17 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1270.47draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 18 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1280.56draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 19 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1286.94draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 20 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1278.56draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 21 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1307.16draws/s]
The number of effective samples is smaller than 25% for some parameters.
```

```
  Fitting model 22 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1286.30draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 23 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1273.71draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 24 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1283.06draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 25 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1292.89draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 26 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1277.71draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 27 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1261.56draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 28 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1281.65draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 29 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1271.98draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 30 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:31<00:00, 1268.38draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
/!\ Automatically setting parameter precision...
```

[60]:
```
np.random.seed(12)

glam_nobias.exchange_data(data_even)
glam_nobias.fit(method='MCMC',
                tune=n_tune,
                draws=n_draws,
                chains=n_chains)
```

```
Replaced attached data (9000 trials) with new data (4500 trials)...
Fitting 30 model(s) using MCMC...
  Fitting model 1 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1916.18draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 2 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1935.55draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 3 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1926.12draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 4 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1923.53draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 5 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1927.32draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 6 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1939.52draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 7 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1928.49draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 8 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1944.04draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 9 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1944.95draws/s]
The number of effective samples is smaller than 25% for some parameters.
  Fitting model 10 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1934.32draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 11 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1914.56draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 12 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1927.79draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 13 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1931.46draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 14 of 30...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1923.61draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 15 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1926.58draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 16 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1925.35draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 17 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1940.44draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 18 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1923.64draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 19 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1919.02draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 20 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1935.94draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 21 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:19<00:00, 2016.08draws/s]
The number of effective samples is smaller than 25% for some parameters.
```

```
  Fitting model 22 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1953.99draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 23 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1944.47draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 24 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1970.87draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 25 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1975.37draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 26 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1992.38draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 27 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
```

(continues on next page)

(continued from previous page)

```
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1933.80draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 28 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1955.43draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 29 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1967.54draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 30 of 30...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:20<00:00, 1968.31draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
/!\ Automatically setting parameter precision...
```

Again, we check all parameter traces for convergence, before proceeding further in the anlaysis:

[61]:
```
np.all([check_convergence(pm.summary(trace), varnames=['v', 's', 'gamma', 'tau'])
        for trace in glam_bias.trace])
```

[61]:
```
True
```

[62]:
```
np.all([check_convergence(pm.summary(trace), varnames=['v', 's', 'tau'])
        for trace in glam_nobias.trace])
```

[62]:
```
True
```

[63]:
```
for subject, subject_trace in enumerate(glam_bias.trace):
    gb.plots.traceplot(subject_trace)
    plt.savefig('examples/example_1/results/out_of_sample/traces/sub_{}_gaze_bias_
↪model_trace.png'.format(subject), dpi=330)
    plt.close()
    pm.trace_to_dataframe(subject_trace).to_csv(
        'examples/example_1/results/out_of_sample/traces/sub_{}_gaze_bias_model_trace.
↪csv'.format(subject))
```

[64]:
```
for subject, subject_trace in enumerate(glam_nobias.trace):
    gb.plots.traceplot(subject_trace)
    plt.savefig('examples/example_1/results/out_of_sample/traces/sub_{}_no_gaze_bias_
↪model_trace.png'.format(subject), dpi=330)
```

(continues on next page)

(continued from previous page)

```
    plt.close()
    pm.trace_to_dataframe(subject_trace).to_csv(
        'examples/example_1/results/out_of_sample/traces/sub_{}_no_gaze_bias_model_
↪trace.csv'.format(subject))
```

We then evaluate the performance of both models in predicting individuals' response behaviour using the MAP estimates and item value and gaze data from the odd-numbered trials. To predict response behaviour for the odd-numbered trials, we use the `predict` method. We repeat every trial 50 times in the prediction (as specified through the `n_repeats` argument) to obtain a stable pattern of predictions:

[65]:
```
n_repeats = 50
```

[66]:
```
np.random.seed(13)

glam_bias.exchange_data(data_odd)
glam_bias.predict(n_repeats=n_repeats)
glam_bias.prediction.to_csv('examples/example_1/results/out_of_sample/predictions/
↪gaze_bias_model_predictions.csv')
```

```
  0%|          | 0/4500 [00:00<?, ?it/s]
```

```
Replaced attached data (4500 trials) with new data (4500 trials)...
Generating predictions for 4500 trials (50 repeats each)...
```

```
100%|| 4500/4500 [57:15<00:00,  1.31it/s]
```

[67]:
```
np.random.seed(14)

glam_nobias.exchange_data(data_odd)
glam_nobias.predict(n_repeats=n_repeats)
glam_nobias.prediction.to_csv('examples/example_1/results/out_of_sample/predictions/
↪no_gaze_bias_model_predictions.csv')
```

```
  0%|          | 0/4500 [00:00<?, ?it/s]
```

```
Replaced attached data (4500 trials) with new data (4500 trials)...
Generating predictions for 4500 trials (50 repeats each)...
```

```
100%|| 4500/4500 [57:29<00:00,  1.30it/s]
```

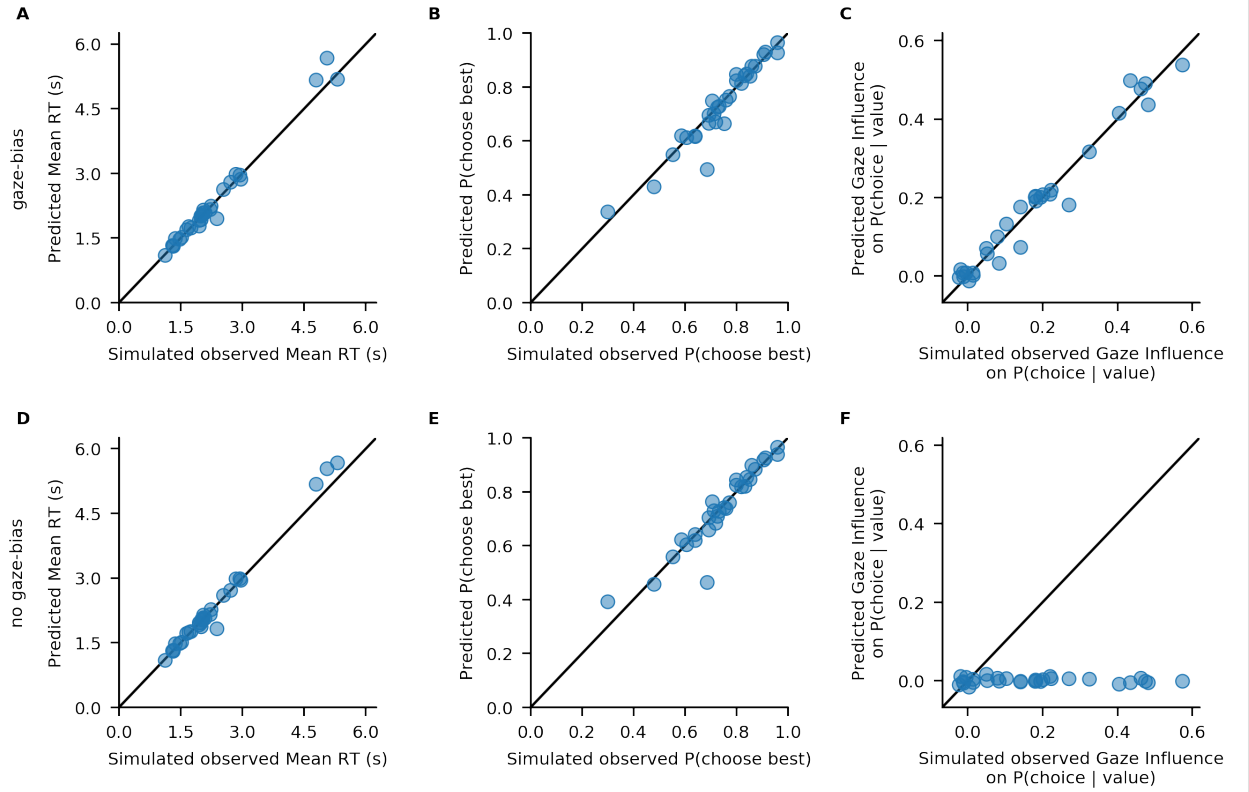
To determine the absolute fit of both model variants to the data, we plot the individually predicted against the individually observed data on all three behavioural metrics. To do this, we use the `plot_individual_fit` function of the `plots` module. This function takes as input the observed data, as well as a list of the predictions of all model variants that ought to be compared. The argument `prediction_labels` specifies the naming used for each model in the resulting figure. For each model variant, the function creates a row of panels, plotting the observed against the predicted data:

[70]:
```
np.random.seed(15)

fig, axs = gb.plots.plot_individual_fit(observed=data_odd,
                                        predictions=[glam_bias.prediction,
                                                     glam_nobias.prediction],
                                        prediction_labels=['gaze-bias', 'no gaze-bias
↪'])

# We'll change the xlabels to "Simulated observed", just to be clear that these are␣
↪simulated data!
for ax in axs.ravel():
```

(continues on next page)

(continued from previous page)

```
    xlabel = ax.get_xlabel()
    ax.set_xlabel('Simulated o' + xlabel[1:])

fig.savefig('examples/example_1/figures/Figure_4_absolute_fit.png', dpi=330)
```



Both model variants perform well in capturing individuals' RTs and probability of choosing the best item (A, D, B, E). Importantly, only the GLAM variant with gaze bias is able to also recover the strength of the association between individuals' choice behaviour and gaze allocation (C).

5. References:

Thomas, A. W., Molter, F., Krajbich, I., Heekeren, H. R., & Mohr, P. N. (2019). Gaze bias differences capture individual choice behaviour. Nature human behaviour, 3(6), 625.

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and computing, 27(5), 1413-1432.

[1]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import numpy as np
import pandas as pd
from scipy.stats import ttest_ind
import pymc3 as pm
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

(continues on next page)

(continued from previous page)

```
sns.set_style('ticks')
sns.set_context('paper')

import glambox as gb
```

3.3.5 Example 2: Hierarchical parameter estimation in cases with limited data

In some research settings, the total amount of data one can collect per individual is limited, conflicting with the large amounts of data required to obtain reliable and precise individual parameter estimates from diffusion models (see for example, Lerche at al. (2017) and Voss et al. (2013)). Hierarchical modeling can offer a solution to this problem. Here, each individual's parameter estimates are assumed to be drawn from a group level distribution. Thereby, during parameter estimation, individual parameter estimates are informed by the data of the entire group. This can greatly improve parameter estimation, especially in the face of limited amounts of data (see for example Ratcliff & Childers (2015) and Wiecki et al. (2013)). In this example, we will simulate a clinical application setting, in which different patient groups are to be compared on the strengths of their gaze biases, during a simple value-based choice task that includes eye tracking. It is reasonable to assume that the amount of data that can be collected in such a setting is limited on at least two accounts:

- The number of patients available for the experiment might be low
- The number of trials that can be performed by each participant might be low, for clinical reasons (e.g., patients feel exhausted more quickly, time to perform tests is limited, etc.)

Therefore, we simulate a dataset with a low number of individuals within each group (between 5 and 15 per group), and a low number of trials per participant (50 trials). We then estimate model parameters in a hierarchical fashion, and compare the group level gaze bias parameter estimates between groups.

1. Simulating Data

We simulate data of three patient groups ($N_1 = 5$, $N_2 = 10$, $N_3 = 15$), with 50 trials per individual, in a simple three item value-based choice task, where participants are instructed to simply choose the item they like the best. These numbers are roughly based on a recent clinical study on the role of the prefrontal cortex in fixation-dependent value representations (Vaidya & Fellows (2015)). Here, the authors found no systematic differences between frontal lobe patients and controls on integration speed or the decision threshold, controlling speed-accuracy trade-offs. Therefore, in our example we only let the gaze bias parameter γ differ systematically between the groups, with means of $\gamma_1 = 0.7$ (weak gaze bias), $\gamma_2 = 0.1$ (moderate gaze bias) and $\gamma_3 = 0.5$ (strong gaze bias), respectively. We do not assume any other systematic differences between the groups and sample all other model parameters from the estimates obtained from fitting the model to the data of Krajbich & Rangel (2011).

[2]:
```
## Load Krajbich & Rangel (2011) dataset estimates from Thomas, Molter et al. (2019)
estimates = pd.read_csv('resources/individual_estimates_sec_nhb2019.csv')
kr2011 = estimates.loc[estimates['dataset'] == 'krajbich2011']
kr2011.head()
```

[2]:

	subject	dataset	v	gamma	s	tau
39	39	krajbich2011	1.207348	0.389116	0.227103	0.354587
40	40	krajbich2011	0.633304	0.353474	0.279450	0.622022
41	41	krajbich2011	0.737201	0.814595	0.282122	0.787967
42	42	krajbich2011	0.800456	-0.154669	0.328268	1.505673
43	43	krajbich2011	0.627823	0.658645	0.219142	0.549583

[3]:
```
np.random.seed(1751)
```

(continues on next page)

(continued from previous page)

```
groups = ['group1', 'group2', 'group3']

# Sample sizes
N = dict(group1=5,
         group2=10,
         group3=15)

# Mean gaze bias parameters for each group
gamma_mu = dict(group1=0.7,      # weak bias
                group2=0.1,      # medium-strong bias
                group3=-0.5)     # very strong bias with leak

# Sample parameter sets from KR2011 estimates
group_idx = {group: np.random.choice(kr2011.index, size=N[group], replace=False)
             for group in groups}
v = {group: kr2011.loc[group_idx[group], 'v'].values
     for group in groups}
s = {group: kr2011.loc[group_idx[group], 's'].values
     for group in groups}
tau = {group: kr2011.loc[group_idx[group], 'tau'].values
       for group in groups}

# Draw normally distributed gaze bias parameters (truncated to be  1)
gamma = dict(group1=np.clip(np.random.normal(loc=gamma_mu['group1'], scale=0.3,␣
↪size=N['group1']), a_min=None, a_max=1),
             group2=np.clip(np.random.normal(loc=gamma_mu['group2'], scale=0.2,␣
↪size=N['group2']), a_min=None, a_max=1),
             group3=np.clip(np.random.normal(loc=gamma_mu['group3'], scale=0.3,␣
↪size=N['group3']), a_min=None, a_max=1))

# Set the number of trials and items in the task
n_trials = 50
n_items  = 3

# Simulate the data using GLAM
glam = gb.GLAM()
for g, group in enumerate(groups):
    glam.simulate_group(kind='individual',
                        n_individuals=N[group],
                        n_trials=n_trials,
                        n_items=n_items,
                        parameters=dict(v=v[group],
                                        gamma=gamma[group],
                                        s=s[group],
                                        tau=tau[group],
                                        t0=np.zeros(N[group])),
                        label=group,
                        seed=g)

data = glam.data.copy()
data.rename({'condition': 'group'}, axis=1, inplace=True)
data.to_csv('examples/example_2/data/data.csv', index=False)
data.head(3)
```

[3]:

	subject	trial	repeat	choice	rt	item_value_0	gaze_0	\
0	0.0	0.0	0.0	0.0	1.266691	5	0.459175	
1	0.0	1.0	0.0	2.0	1.189464	3	0.389115	

(continues on next page)

(continued from previous page)

```
2       0.0     2.0      0.0      1.0  1.709817              3  0.269665

   item_value_1    gaze_1  item_value_2    gaze_2   group
0             0  0.379466             3  0.161359  group1
1             7  0.189588             9  0.421297  group1
2             5  0.409055             2  0.321280  group1
```

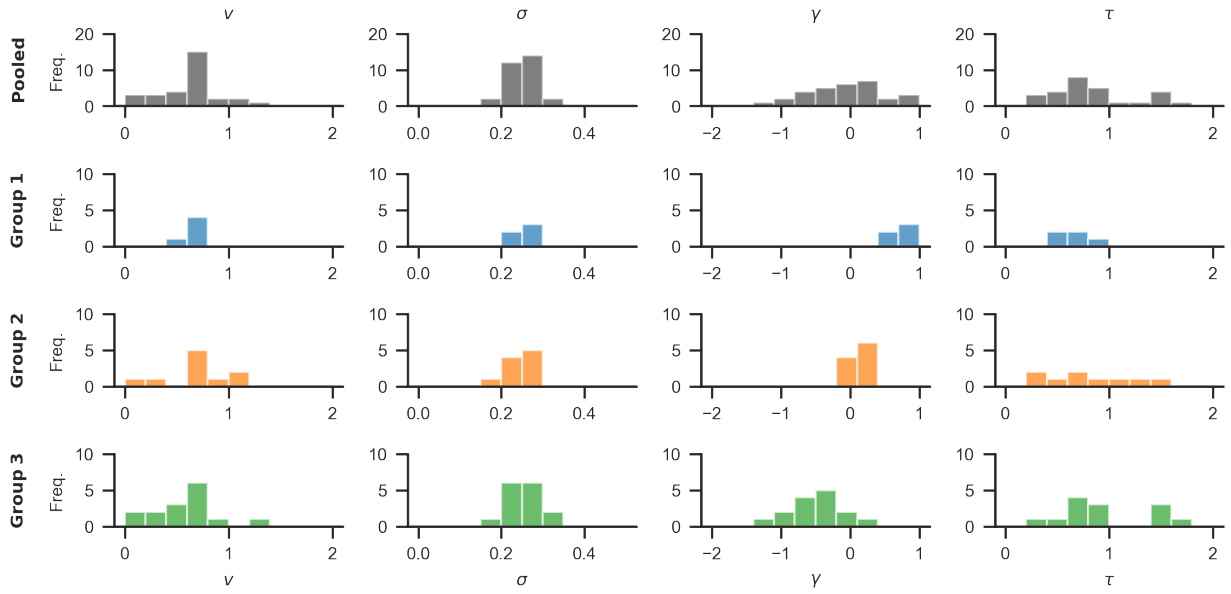
The resulting distribution of data generating parameters looks as follows:

[4]:
```
from string import ascii_uppercase

fontsize = 7
n_bins = 10

fig, axs = plt.subplots(4, 4, figsize=gb.plots.cm2inch(18, 9), sharex='col', dpi=330)

for p, (parameter, parameter_name, bins) in enumerate(
    zip([v, s, gamma, tau],
        [r'$v$', r'$\sigma$', r'$\gamma$', r'$\tau$'],
        [np.linspace(0, 2, n_bins + 1),
         np.linspace(0, 0.5, n_bins + 1),
         np.linspace(-2, 1, n_bins + 1),
         np.linspace(0, 2, n_bins + 1)])):

    # Pooled
    axs[0, p].hist(np.concatenate([parameter[group] for group in groups]),
                   bins=bins,
                   color='black',
                   alpha=0.5)
    axs[0, p].set_ylim(0, 20)
    axs[0, p].set_yticks([0, 10, 20])
    axs[0, p].set_title(parameter_name, fontsize=fontsize, fontweight='bold')
    if p == 0:
        axs[0, p].set_ylabel(r'$\bf{Pooled}$' + '\n\nFreq.', fontsize=fontsize)

    # Groups
    for g, group in enumerate(groups):
        axs[g + 1, p].hist(parameter[group],
                           bins=bins,
                           color='C{}'.format(g),
                           alpha=0.7)
        axs[g + 1, p].set_ylim(0, 1)
        axs[g + 1, p].set_yticks([0, 5, 10])
        if p == 0:
            axs[g + 1, p].set_ylabel(r'$\bf{Group}$ ' + r'$\bf{{{g}}}$'.format(g=(g +␣
↪1)) + '\n\nFreq.',
                                     fontsize=fontsize)

    axs[g + 1, p].set_xlabel(parameter_name, fontsize=fontsize)

for ax, letter in zip(axs.ravel(), ascii_uppercase):
    # Show ticklabels, despite sharey=True
    ax.xaxis.set_tick_params(labelbottom=True)
    ax.yaxis.set_tick_params(labelbottom=True)
    ax.tick_params(axis='both', which='major', labelsize=fontsize)

sns.despine()
```

(continues on next page)

(continued from previous page)

```
fig.tight_layout(pad=1.02)

plt.savefig('examples/example_2/figures/generating_parameters.png', dpi=330)
```



2. Exploring behavioural differences

Behavioural differences between the three groups are plotted below, using the `plot_behaviour_aggregate` function from the `plots` module. Group-level summary tables can be created using the `aggregate_group_level_data` from the `analysis` module. Even though the groups only differ in the gaze bias parameter γ, they also exhibit differences in RT (Group 1 mean $\pm$ s.d. = 1.96 $\pm$ 0.33 s, Group 2 mean $\pm$ s.d. = 2.38 $\pm$ 1.4 s; Group 3 mean $\pm$ s.d. = 2.59 $\pm$ 1.26 ms; A) and choice accuracy (Group 1 mean $\pm$ s.d. = 0.88 $\pm$ 0.06, Group 2 mean $\pm$ s.d. = 0.71 $\pm$ 0.07, Group 3 mean $\pm$ s.d. = 0.50 $\pm$ 0.16; B). As is to be expected, we can also observe behavioural differences in gaze influence measure (Group 1 mean $\pm$ s.d. = 0.08 $\pm$ 0.07, Group 2 mean $\pm$ s.d. = 0.26 $\pm$ 0.11, Group 3 mean $\pm$ s.d. = 0.38 $\pm$ 0.11; C-D, where the choices of Group 3 are driven by gaze more than those of the other groups.

[5]:
```
parameters = ['v', 'gamma', 's', 'tau']
comparisons=[('group1', 'group2'),
             ('group1', 'group3'),
             ('group2', 'group3')]

for p, parameter in zip([v, gamma, s, tau], ['v', 'gamma', 's', 'tau']):
    print(parameter)
    for comparison in comparisons:
        g1, g2 = comparison
        print('{} vs {}\t'.format(g1, g2), ttest_ind(p[g1], p[g2]))
```

```
v
group1 vs group2        Ttest_indResult(statistic=-0.3063071158648262, pvalue=0.
↪7642213460655614)
group1 vs group3        Ttest_indResult(statistic=0.5357661157553092, pvalue=0.
↪5986787412351837)
```

(continues on next page)

(continued from previous page)

```
group2 vs group3         Ttest_indResult(statistic=0.9437792037490204, pvalue=0.
↪35509174773278895)
gamma
group1 vs group2         Ttest_indResult(statistic=6.4021176850791575, pvalue=2.
↪3355862858023058e-05)
group1 vs group3         Ttest_indResult(statistic=6.8208382105890335, pvalue=2.
↪193832426730825e-06)
group2 vs group3         Ttest_indResult(statistic=4.837346716941595, pvalue=6.
↪985175866537608e-05)
s
group1 vs group2         Ttest_indResult(statistic=0.8636132045092535, pvalue=0.
↪4034531813427481)
group1 vs group3         Ttest_indResult(statistic=0.004104731012479496, pvalue=0.
↪9967700581739689)
group2 vs group3         Ttest_indResult(statistic=-0.8970137843731146, pvalue=0.
↪37900526220793296)
tau
group1 vs group2         Ttest_indResult(statistic=-0.8378245121922042, pvalue=0.
↪417267319030197)
group1 vs group3         Ttest_indResult(statistic=-1.6166494748100346, pvalue=0.
↪12334443855331839)
group2 vs group3         Ttest_indResult(statistic=-0.8993267331462613, pvalue=0.
↪3777982033456534)
```

[6]:
```
data.groupby('group').apply(gb.analysis.aggregate_group_level_data, n_items=n_items)
```

[6]:

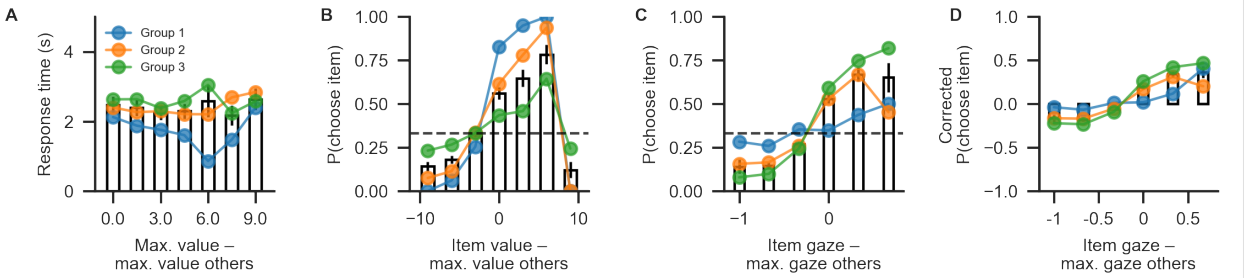
		mean	std	min	max	se	iqr
group							
group1	Mean RT	1.960980	0.326799	1.531284	2.503894	0.163400	0.304160
	P(choose best)	0.876000	0.062482	0.760000	0.920000	0.031241	0.060000
	Gaze Influence	0.077391	0.066525	-0.005919	0.142432	0.033263	0.133686
group2	Mean RT	2.381671	1.402121	1.238400	5.187776	0.467374	0.336601
	P(choose best)	0.706000	0.065146	0.600000	0.800000	0.021715	0.100000
	Gaze Influence	0.255080	0.112799	0.103472	0.440610	0.037600	0.174701
group3	Mean RT	2.588283	1.256759	1.240456	6.318852	0.335883	0.872888
	P(choose best)	0.498667	0.155857	0.280000	0.820000	0.041655	0.200000
	Gaze Influence	0.382977	0.112599	0.144232	0.652829	0.030093	0.122224

[7]:
```
gb.plots.plot_behaviour_aggregate(bar_data=data,
                                  line_data=[data.loc[data['group'] == group] for␣
↪group in groups],
                                  line_labels=['Group 1', 'Group 2', 'Group 3'],
                                  value_bins=7, gaze_bins=7,
                                  limits=dict(rt=(0, 5)));
```

(continues on next page)

(continued from previous page)

```
sns.despine()
plt.savefig('examples/example_2/figures/aggregate_data.png', dp=330)
```



3. Building the hierarchical model

When specifying the hierarchical model, we allow all model parameters to differ between the three groups. This way, we will subsequently be able to address the question whether individuals from different groups differ on one or more model parameters (including the gaze bias parameter γ, which we are mainly interested in here).

As for the individual models, we first initialize the model object using the `GLAM` class and supply it with the behavioural data using the `data` argument. Here, we set the model kind `'hierarchical'` (in contrast to `'individual'`. Further, we specify that each model parameter can vary between groups (referring to a `'group'` variable in the data):

[8]:
```
hglam = gb.GLAM(data=data)
hglam.make_model(kind='hierarchical',
                 depends_on=dict(v='group',
                                 gamma='group',
                                 s='group',
                                 tau='group'))
```

```
Generating hierarchical model for 30 subjects...
```

In this model, each parameter is set up hierarchically within each group, so that individual estimates are informed by other individuals in that group. If the researcher does not expect group differences on a parameter, this parameter can simply be omitted from the `depends_on` dictionary. The resulting model would then have a hierarchical setup of this parameter across groups, so that individual parameter estimates were informed by all other individuals (not only those in the same group).

4. Parameter estimation

After the model is built, the next step is to perform statistical inference over its parameters. As we have done with the individual models, we can use MCMC to approximate the parameters' posterior distributions (see the Methods Section of the manuscript for details). Due to the more complex structure and drastically increased number of parameters, the chains from the hierarchical model usually have higher levels autocorrelation. To still obtain a reasonable number of effective samples (Kurschke (2014), we increase the number of `tuning`- and `draw` steps:

[9]:
```
np.random.seed(1142)
hglam.fit(method='MCMC',
          draws=20000,
          tune=20000,
          chains=4,
          random_seed=1142)
```

```
Fitting 1 model(s) using MCMC...
  Fitting model 1 of 1...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau_group3]
>Metropolis: [tau_group2]
>Metropolis: [tau_group1]
>Metropolis: [tau_group3_sd]
>Metropolis: [tau_group2_sd]
>Metropolis: [tau_group1_sd]
>Metropolis: [tau_group3_mu]
>Metropolis: [tau_group2_mu]
>Metropolis: [tau_group1_mu]
>Metropolis: [s_group3]
>Metropolis: [s_group2]
>Metropolis: [s_group1]
>Metropolis: [s_group3_sd]
>Metropolis: [s_group2_sd]
>Metropolis: [s_group1_sd]
>Metropolis: [s_group3_mu]
>Metropolis: [s_group2_mu]
>Metropolis: [s_group1_mu]
>Metropolis: [gamma_group3]
>Metropolis: [gamma_group2]
>Metropolis: [gamma_group1]
>Metropolis: [gamma_group3_sd]
>Metropolis: [gamma_group2_sd]
>Metropolis: [gamma_group1_sd]
>Metropolis: [gamma_group3_mu]
>Metropolis: [gamma_group2_mu]
>Metropolis: [gamma_group1_mu]
>Metropolis: [v_group3]
>Metropolis: [v_group2]
>Metropolis: [v_group1]
>Metropolis: [v_group3_sd]
>Metropolis: [v_group2_sd]
>Metropolis: [v_group1_sd]
>Metropolis: [v_group3_mu]
>Metropolis: [v_group2_mu]
>Metropolis: [v_group1_mu]
Sampling 4 chains: 100%|| 160000/160000 [54:12<00:00, 49.20draws/s]
The estimated number of effective samples is smaller than 200 for some parameters.
```

```
/!\ Automatically setting parameter precision...
```

[10]:
```
# saving the hierarchical parameter estimates
hglam.estimates.to_csv('examples/example_2/results/estimates.csv', index=False)
```

Check for convergence, using the Rhat measure and number of effective samples:

[11]:
```
variables = [v for v in hglam.trace[0].varnames
             if not v.endswith('__')
             and not v in ['b', 'p_error', 't0']]

pm.summary(hglam.trace[0], var_names=variables).head()
```

[11]:
```
                mean        sd  mc_error   hpd_2.5  hpd_97.5        n_eff  \
```

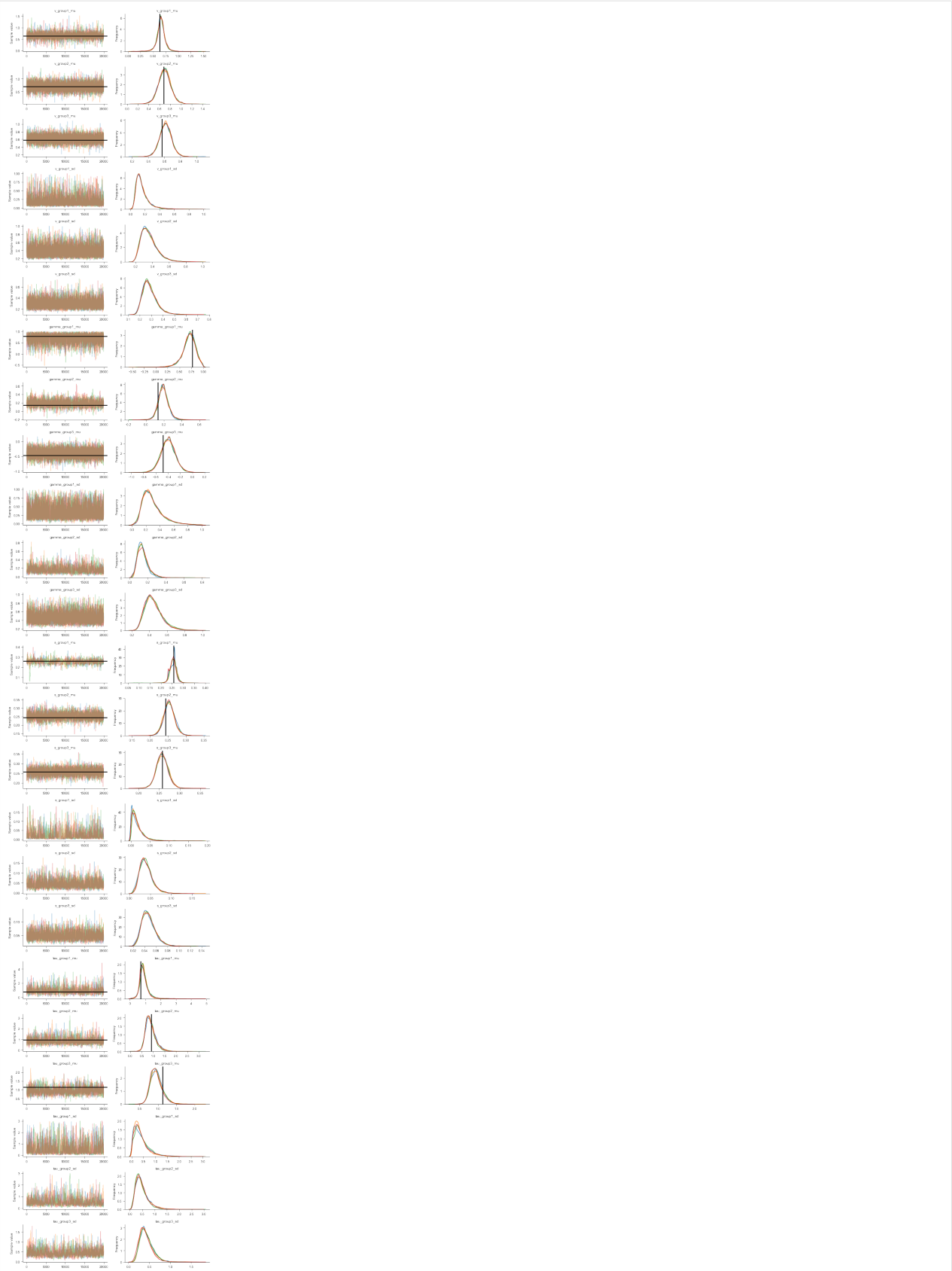
(continues on next page)

(continued from previous page)

```
v_group1_mu  0.655545  0.091426  0.001239  0.471340  0.837252  6536.876639
v_group2_mu  0.695481  0.120024  0.001420  0.456701  0.935323  9496.736263
v_group3_mu  0.614757  0.076902  0.000935  0.468496  0.773918  7983.230317
v_group1_sd  0.167254  0.105907  0.001861  0.041294  0.383618  3369.765623
v_group2_sd  0.364104  0.104120  0.001371  0.198576  0.573141  6578.210532

                 Rhat
v_group1_mu  1.000135
v_group2_mu  1.000189
v_group3_mu  1.000011
v_group1_sd  1.000453
v_group2_sd  1.000039
```

Let's also take a look at the traces for visual inspection:

[12]:
```
variables = [v for v in variables
             if v.endswith('_mu')
             or v.endswith('_sd')]

gb.plots.traceplot(hglam.trace[0],
                   varnames=variables,
                   # specify data generating reference values (vertical lines)
                   ref_val=dict(v_group1_mu=v['group1'].mean(),
                                v_group2_mu=v['group2'].mean(),
                                v_group3_mu=v['group3'].mean(),
                                gamma_group3_mu=gamma['group3'].mean(),
                                gamma_group1_mu=gamma['group1'].mean(),
                                gamma_group2_mu=gamma['group2'].mean(),
                                s_group3_mu=s['group3'].mean(),
                                s_group1_mu=s['group1'].mean(),
                                s_group2_mu=s['group2'].mean(),
                                tau_group3_mu=tau['group3'].mean(),
                                tau_group1_mu=tau['group1'].mean(),
                                tau_group2_mu=tau['group2'].mean()))

plt.savefig('examples/example_2/figures/traceplot.png', dpi=330)
sns.despine()
```



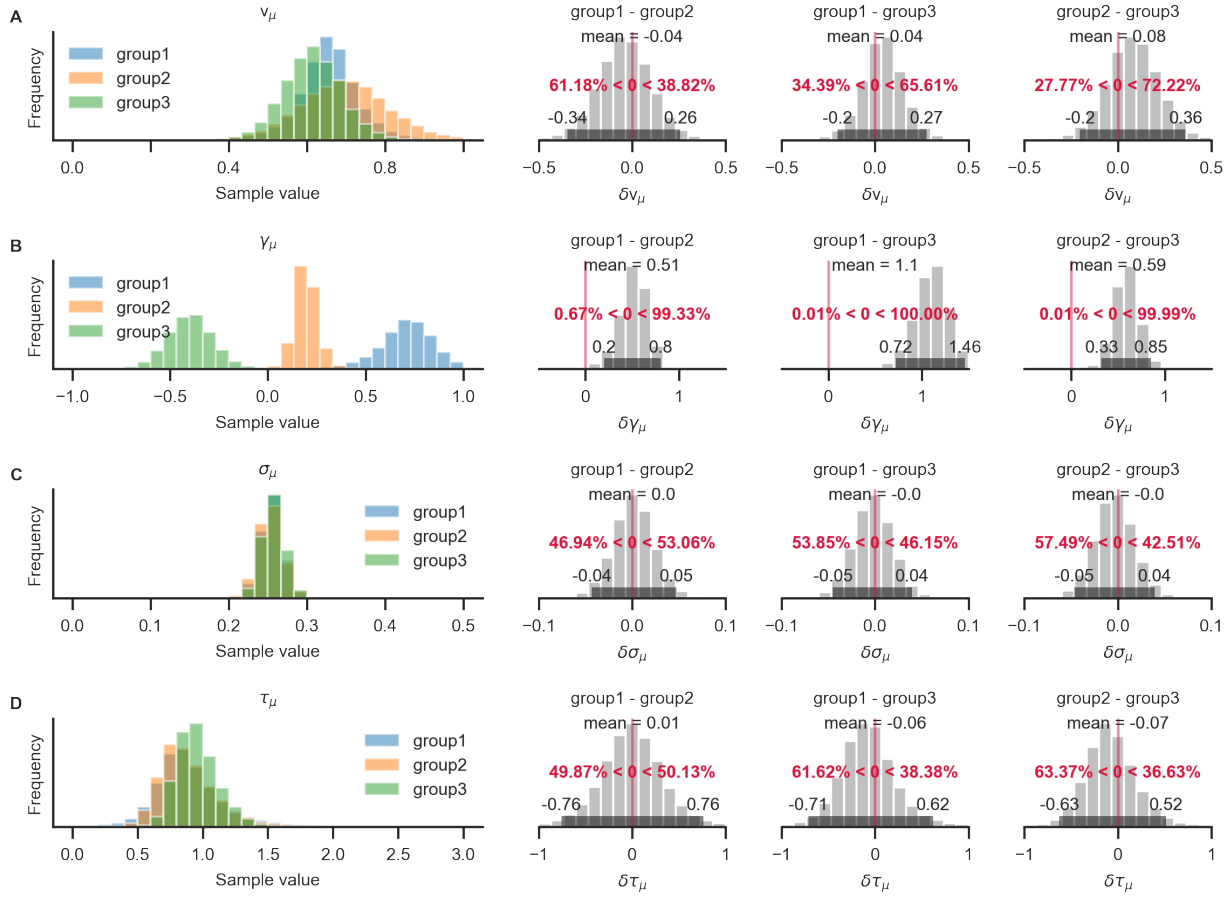
5. Evaluating parameter estimates, interpreting results

After sampling is finished, and the chains were checked for convergence (see above), we can turn back to the research question: *Do the groups differ with respect to their gaze biases?*

Questions about differences between group-level parameters can be addressed by inspecting their posterior distributions. For example, the probability that the mean $\gamma_{1,\mu}$ for Group 1 is larger than the mean $\gamma_{2,\mu}$ of Group 2 is given by the proportion of posterior samples in which this was the case.

GLAMbox includes a `compare_parameters` function that plots posterior distributions of group level parameters. Additionally, the user can specify a list of `comparisons` between groups or conditions. If comparisons are specified, the posterior distributions of their difference and corresponding relevant statistics are added to the figure:

[13]:
```
from glambox.plots import compare_parameters

parameters = ['v', 'gamma', 's', 'tau']
comparisons = [('group1', 'group2'),
               ('group1', 'group3'),
               ('group2', 'group3')]

compare_parameters(model=hglam,
                   parameters=parameters,
                   comparisons=comparisons);

plt.savefig('examples/example_2/figures/node_comparison.png', dpi=330)
```



With the resulting plot (see above), the researcher can infer that the groups did not differ with respect to their mean

velocity parameters $v_{i,\mu}$ (A; top row, pairwise comparisons), mean accumulation noise $\sigma_{i,\mu}$ (C; third row), or scaling parameters $\tau_{i,\mu}$ (D; fourth row). The groups differ, however, in the strength of their mean gaze bias $\gamma_{i,\mu}$ (B; second row): All differences between the groups were statistically meaningful (as inferred by the fact that the corresponding 95% HPD did not contain zero; B, second row, columns 2-4).

6. References

- Krajbich, I., & Rangel, A. (2011). Multialternative drift-diffusion model predicts the relationship between visual fixations and choice in value-based decisions. Proceedings of the National Academy of Sciences, 108(33), 13852-13857.
- Kruschke, J. (2014). Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan. Academic Press.
- Lerche, V., Voss, A., & Nagler, M. (2017). How many trials are required for parameter estimation in diffusion modeling? A comparison of different optimization criteria. Behavior Research Methods, 49(2), 513-537.
- Ratcliff, R., & Childers, R. (2015). Individual differences and fitting methods for the two-choice diffusion model of decision making. Decision, 2(4), 237.
- Vaidya, A. R., & Fellows, L. K. (2015). Testing necessary regional frontal contributions to value assessment and fixation-based updating. Nature communications, 6, 10120.
- Voss, A., Nagler, M., & Lerche, V. (2013). Diffusion models in experimental psychology. Experimental psychology.
- Wiecki, T. V., Sofer, I., & Frank, M. J. (2013). HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in Python. Frontiers in neuroinformatics, 7, 14.

[1]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import numpy as np
import pandas as pd
import pymc3 as pm
from os.path import join, isfile
from os import listdir
from functools import partial
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style('ticks')
sns.set_context('paper')

import glambox as gb
```

3.3.6 Example 3: Parameter Recovery

When performing model-based analyses of behaviour that include the interpretation of parameter estimates, or comparisons of parameter estimates between groups or conditions, the researcher should be confident that the model's parameters are actually identifiable. In particular, the researcher needs to be confident that the set of estimated parameters unambiguously describes the observed data better than any other set of parameters. A straightforward way of testing this is to perform a parameter recovery: The general intuition of a parameter recovery analysis is to first generate a synthetic dataset from a model using a set of known parameters, and then fitting the model to the synthetic data. Finally, the estimated parameters can be compared to the known generating parameters. If they match to a satisfying degree, the parameters were recovered successfully. Previous analyses have already indicated that the GLAM's parameters can be recovered to a satisfying degree (Thomas et al., 2019). Yet, the ability to identify a given

set of parameters always depends on the specific features of a given dataset. The most obvious feature of a dataset that influences recoverability of model parameters is the number of data points included. Usually this quantity refers to the number of trials that participants performed. For hierarchical models, the precision of group-level estimates also depends on the number of individuals per group. Additional features that vary between datasets and that could influence parameter estimation are the observed distribution of gaze, the distribution of item values or the number of items in each trial. For this reason, it is recommended to test whether the estimated parameters of a model can be recovered in the context of a specific dataset.

1. Simulating "observed data"

[2]:
```
np.random.seed(2)
# Simulate a pretend-to-be-collected dataset.
# In the real world, this dataset is collected from participants, so we do not know
# 1) if GLAM is an adequate model for the data
# 2) data-generating parameters

data_model = gb.GLAM()

parameters = dict(v=dict(mu=0.6, sd=0.25, bounds=(0, 1.5)),
                  gamma=dict(mu=0.1, sd=0.4, bounds=(-1, 1)),
                  s=dict(mu=0.25, sd=0.05, bounds=(0.05, 0.75)),
                  tau=dict(mu=1.0, sd=0.3, bounds=(0.1, 2)))

data_model.simulate_group(kind='hierarchical',
                          n_individuals=50,
                          n_trials=200,
                          n_items=3,
                          parameters=parameters,
                          value_range=(1, 10),
                          seed=1)

data = data_model.data
```

[3]:
```
# Save data to file
data.to_csv(join('examples', 'example_3', 'data', 'data.csv'), index=False)
```

2. Parameter estimation

To demonstrate the procedure of a basic parameter recovery analysis using GLAMbox, suppose we have collected and loaded a dataset called `data`. In the first step, we perform parameter estimation as in the previous examples:

[4]:
```
np.random.seed(4)
glam = gb.GLAM(data=data)
glam.make_model(kind='individual')
glam.fit(method='MCMC',
         draws=5000,
         tune=5000,
         chains=4)
```

```
Generating single subject models for 50 subjects...
Fitting 50 model(s) using MCMC...
  Fitting model 1 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:28<00:00, 1392.86draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 2 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1351.54draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 3 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1362.03draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 4 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1352.12draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 5 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1343.94draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 6 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1356.30draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 7 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1344.96draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 8 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1368.29draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 9 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1362.10draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 10 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1372.75draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 11 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1348.38draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 12 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1352.57draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 13 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1352.82draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 14 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1358.10draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 15 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1359.78draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 16 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1336.29draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 17 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1329.44draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 18 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1311.59draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 19 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1342.75draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 20 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:30<00:00, 1333.05draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 21 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1336.53draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 22 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:28<00:00, 1399.04draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 23 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1453.70draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 24 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1461.93draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 25 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1472.69draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 26 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1470.75draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 27 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1483.43draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 28 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1471.93draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 29 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1481.82draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 30 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1483.44draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 31 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1478.25draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 32 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1470.72draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 33 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1446.27draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 34 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1487.18draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 35 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1491.35draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 36 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1460.41draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 37 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1489.48draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 38 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1483.08draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 39 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1465.77draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 40 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1474.64draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 41 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1468.09draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 42 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1476.85draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 43 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1467.98draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 44 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1469.22draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 45 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1466.65draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 46 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1485.67draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 47 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1497.46draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 48 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1475.57draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 49 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1482.59draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 50 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1472.00draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
/!\ Automatically setting parameter precision...
```

3. Simulation of synthetic data

The next step is to create a synthetic, model-generated dataset using the model parameters estimated from the empirical data, together with the empirically observed stimulus and gaze data using the `predict` method. Setting `n_repeats` to 1 results in a dataset of the same size as the observed one:

[5]:
```
np.random.seed(5)
glam.predict(n_repeats=1)
synthetic = glam.prediction
```

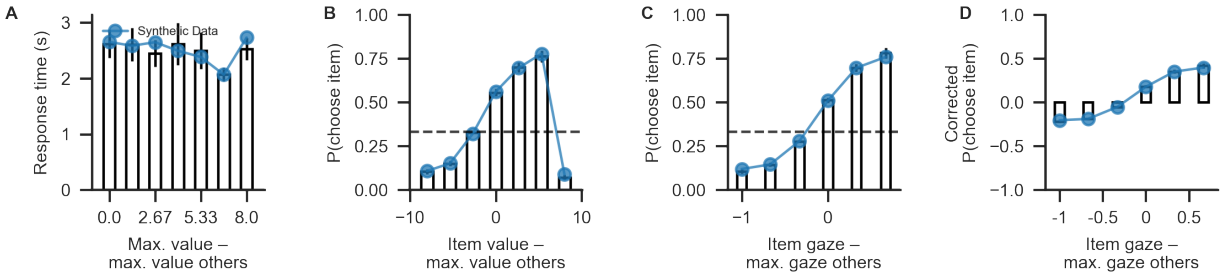
```
  0%|          | 8/10000 [00:00<02:13, 74.93it/s]
Generating predictions for 10000 trials (1 repeats each)...
100%|| 10000/10000 [02:14<00:00, 74.53it/s]
```

[6]:
```
# save synthetic data to file
synthetic.to_csv(join('examples', 'example_3', 'data', 'synthetic.csv'), index=False)
```

[7]:
```
# For this synthetic dataset, we know the generating parameters:
true_parameters = {parameter: glam.estimates[parameter].values
                   for parameter in ['v', 'gamma', 's', 'tau']}

# Save these generating parameters
true_param_df = pd.DataFrame(true_parameters)
true_param_df['subject'] = range(50)
true_param_df.to_csv(join('examples', 'example_3', 'results', 'true_parameters.csv'),␣
↪index=False)
```

The synthetic dataset should resemble the empirically observed data closely. If there are major discrepancies between the synthetic and observed data, this indicates that GLAM might not be a good candidate model for the data at hand.

[8]:
```
# visualize match between "observed" and synthetic datasets
gb.plots.plot_behaviour_aggregate(data, line_data=[synthetic], line_labels=[
↪'Synthetic Data'])
plt.savefig(join('examples', 'example_3', 'figures', 'data-vs-synthetic.png'),␣
↪dpi=330)
```



4. Parameter re-estimation

Next, we create a new model instance, attach the synthetic data, build a model and re-estimate its parameters:

[9]:
```
np.random.seed(9)
glam_rec = gb.GLAM(data=synthetic)
glam_rec.make_model(kind='individual')
glam_rec.fit(method='MCMC',
             draws=5000,
             tune=5000,
             chains=4)
```

```
Generating single subject models for 50 subjects...
Fitting 50 model(s) using MCMC...
  Fitting model 1 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1478.20draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 2 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1493.62draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 3 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1483.86draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 4 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1474.95draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 5 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1459.54draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 6 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:26<00:00, 1491.87draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 7 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1345.92draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 8 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:29<00:00, 1351.26draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 9 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1456.77draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 10 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1474.75draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 11 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1470.08draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 12 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1466.79draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 13 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1475.07draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 14 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1472.22draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 15 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1474.12draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 16 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1474.44draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 17 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1458.41draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 18 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1472.54draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 19 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1471.98draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 20 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1472.00draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 21 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1471.14draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 22 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1463.96draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 23 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1470.77draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 24 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1459.38draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 25 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1457.87draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 26 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1466.27draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 27 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1466.23draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 28 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1467.44draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 29 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1466.37draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 30 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1463.82draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 31 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1468.31draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 32 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1467.64draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 33 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1471.74draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 34 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1475.38draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 35 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1467.14draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 36 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1463.10draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 37 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1469.14draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 38 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1467.60draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 39 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1471.39draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 40 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1466.50draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 41 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1465.58draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 42 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1470.44draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 43 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1469.98draws/s]
The number of effective samples is smaller than 25% for some parameters.
  Fitting model 44 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1472.19draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 45 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1465.69draws/s]
The number of effective samples is smaller than 10% for some parameters.
  Fitting model 46 of 50...
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
```

(continues on next page)

(continued from previous page)

```
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1473.98draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 47 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1466.34draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 48 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1464.09draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 49 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1468.90draws/s]
The number of effective samples is smaller than 10% for some parameters.
```

```
  Fitting model 50 of 50...
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [tau]
>Metropolis: [s]
>Metropolis: [gamma]
>Metropolis: [v]
Sampling 4 chains: 100%|| 40000/40000 [00:27<00:00, 1463.18draws/s]
The number of effective samples is smaller than 25% for some parameters.
```

```
/!\ Automatically setting parameter precision...
```

[10]:
```
# save recovered estimates to file
glam_rec.estimates.to_csv(join('examples', 'example_3', 'results', 'glam_rec_
↪estimates.csv'), index=False)
```

5. Comparison of generating and recovered estimates

Finally, the recovered and generating parameters can be compared. If the recovered parameters do not match the generating parameters, the parameters cannot be identified given this specific dataset. In this case, parameter estimates should not be interpreted.

If, on the other hand, generating and recovered parameters do align, the parameters have been recovered successfully. This indicates that the model's parameters can be identified unambiguously given the general characteristics of the dataset and thereby increases confidence that the parameters obtained from the empirical data are valid and can be interpreted.

[11]:
```
def plot_recovery_individual(model, generating_parameters,
                             parameters=['v', 'gamma', 's', 'tau'],
                             xlimits=dict(v=[-0.1, 0.1],
                                          gamma=[-0.25, 0.25],
                                          s=[-0.075, 0.075],
                                          tau=[-1, 1]),
                             figsize=gb.plots.cm2inch(18, 6),
                             fontsize=7):
    """
    Plot parameter recovery results from individually fitted models.

    Args:
        model: Fitted GLAM model of type 'individual'
        generating_parameters (dict): Dictionary of data generating parameters
        parameters (list, optional): List of parameters to include
        figsize (tuple, optional): Figure size


    """
    parameter_names = {'v': 'v',
                       'gamma': r'$\gamma$',
                       's': r'$\sigma$',
                       'tau': r'$\tau$'}

    n_individuals = len(generating_parameters[parameters[0]])
    n_parameters = len(parameters)

    # Construct long dataframe,
    # every row is one parameter of one subject
    recovery = []
    for parameter in parameters:
        recovery_p = model.estimates[['subject', parameter, parameter + '_hpd_2.5',␣
↪parameter + '_hpd_97.5']].copy()
        recovery_p.rename({parameter: 'recovered',
                           parameter + '_hpd_2.5': 'recovered_hpd_lower',
                           parameter + '_hpd_97.5': 'recovered_hpd_upper'},
                          axis=1, inplace=True)
        recovery_p['parameter'] = parameter
        recovery_p['generating'] = generating_parameters[parameter]
        recovery.append(recovery_p)
    recovery = pd.concat(recovery)
    recovery['success'] = ((recovery['generating'] > recovery['recovered_hpd_lower'])␣
↪&
                           (recovery['generating'] < recovery['recovered_hpd_upper
↪'])).values

    # Plot
    fig = plt.figure(figsize=figsize, dpi=330)
    axs = {}
    for p, parameter in enumerate(parameters):
        axs[(0, p)] = plt.subplot2grid((5, 4), (0, p), rowspan=1)
        axs[(1, p)] = plt.subplot2grid((5, 4), (2, p), rowspan=4, sharex=axs[(0, p)])
```

(continues on next page)

(continued from previous page)

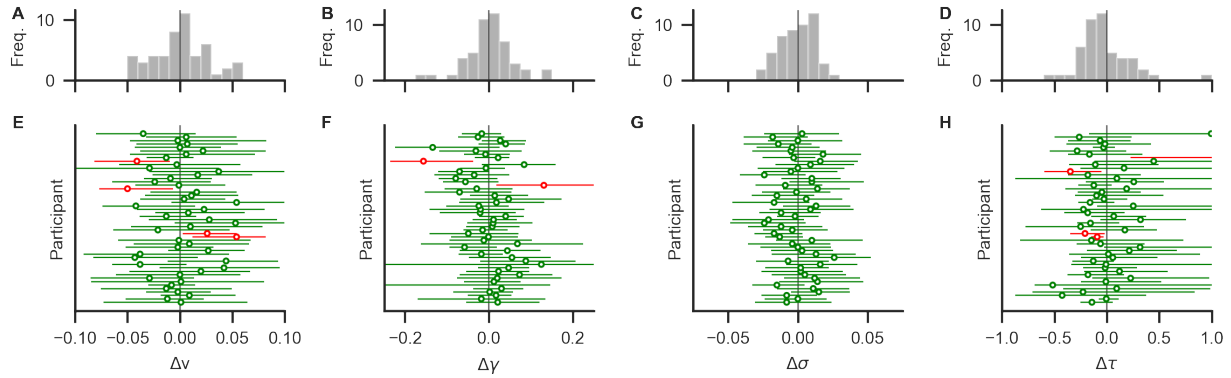
```
    for p, parameter in enumerate(parameters):
        parameter_df = recovery.loc[recovery['parameter'] == parameter]

        # Histogram of differences
        delta = (parameter_df['recovered'] - parameter_df['generating']).values
        axs[(0, p)].hist(delta,
                         color='black', alpha=0.3,
                         bins=np.linspace(*xlimits[parameter], 21))
        axs[(0, p)].axvline(0,
                            color='black', linewidth=0.5, alpha=0.7)
        axs[(0, p)].set_ylabel('Freq.',
                               fontsize=fontsize)
        for label in axs[(0, p)].get_xticklabels():
            label.set_visible(False)

        # Individual HPDs around true value
        ## Success Color coding
        color = np.array(['red', 'green'])[parameter_df['success'].values.astype(int)]

        ## Vertical, indicating zero difference
        axs[(1, p)].axvline(0, color='black', zorder=-1, linewidth=0.5, alpha=0.7)
        ## Difference posterior mean - generating
        axs[(1, p)].scatter(x=delta,
                    y=range(n_individuals),
                    color=color,
                    s=4,
                    marker='o', facecolor='white',
                    zorder=2)
        ## HPD
        axs[(1, p)].hlines(y=range(n_individuals),
                           xmin=parameter_df['recovered_hpd_lower'].values -␣
↪parameter_df['generating'].values,
                           xmax=parameter_df['recovered_hpd_upper'].values -␣
↪parameter_df['generating'].values,
                           linewidth=0.5,
                           zorder=1,
                           color=color)

        ## Labels
        axs[(1, p)].set_xlabel(r'$\Delta$' + parameter_names[parameter],
                       fontsize=fontsize)
        axs[(1, p)].set_ylabel('Participant',
                       fontsize=fontsize)

        ## Limits
        axs[(1, p)].set_xlim(*xlimits[parameter])

        ## Panel Labels
        from string import ascii_uppercase
        for label, ax in zip(list(ascii_uppercase),
                             [axs[i, p]
                              for i in [0, 1]
                              for p in range(len(parameters))]):
            ax.tick_params(axis='both', which='major', labelsize=fontsize)
            ax.text(-0.3, 1.05, label, transform=ax.transAxes,
                    fontsize=fontsize, fontweight='bold', va='top')
        sns.despine()
```

(continues on next page)

(continued from previous page)

```
        fig.tight_layout(h_pad=-1)

        for ax in [axs[(1, p)]
                   for p in range(len(parameters))]:
            ax.set_yticks([])

    return fig, axs
```

[12]:
```
fig, axs = plot_recovery_individual(glam_rec, true_parameters);
plt.savefig(join('examples', 'example_3', 'figures', 'deltaGenRec.png'), dpi=330)
```



The lower row (**E-H**) shows deviations between known generating parameter values and recovered MAP estimates (circles) and their 95% HPDs (horizontal error bars) for each participant. Green (red) colour indicates that the true value is within (outside) the 95% HPD. Most parameters were recovered with small deviations. Panels **A-D** show distributions of deviations across individuals. Distributions are mostly centered around zero, indicating no systematic under- or overestimation (bias) across individuals.

Here, all parameters could be recovered as illustrated in the figure. For most individuals, the MAP estimates and their 95% HPDs are close to the known generating parameters. Across individuals, no systematic biases in the estimation can be identified.

6. Conclusion

In this example, we demonstrated how to perform a basic parameter recovery for a given dataset. When successful, this increases confidence that the parameters can be identified with the given dataset.

7. References

- Thomas, A. W., Molter, F., Krajbich, I., Heekeren, H. R., & Mohr, P. N. (2019). Gaze bias differences capture individual choice behaviour. Nature human behaviour, 3(6), 625. (Link to publisher)

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

g

INDEX