rlssm

Release 0.1.dev

Laura Fontanesi

Aug 02, 2022

GETTING STARTED:

rlssm is a Python package for fitting **reinforcement learning** (RL) models, **sequential sampling models** (DDM, RDM, LBA, ALBA, and ARDM), and **combinations of the two**, using **Bayesian parameter estimation**.

Parameter estimation is done at an individual or hierarchical level using PyStan, the Python Interface to Stan. Stan performs Bayesian inference using the No-U-Turn sampler, a variant of Hamiltonian Monte Carlo.

CHAPTER

ONE

INSTALLATION

You can install the rlssm package using `pip install rlssm`, or get it directly from GitHub.

Make sure you have the dependecies installed first.

1.1 Dependencies

- pystan=2.19
- pandas
- scipy
- seaborn

1.2 Conda environment (suggested)

If you have Andaconda or miniconda installed and you would like to create a separate environment for the rlssm package, do the following:

```
conda create --n stanenv python=3 pandas scipy seaborn pystan=2.19
conda activate stanenv
pip install rlssm
```

Check if the compiler is working:

```
python tests/test_compiler.py
```

On MacOS, if you encounter a compilation error, you can try the following:

```
conda create -n stanenv python=3.7 pandas cython seaborn scipy
conda activate stanenv
conda install clang_osx-64 clangxx_osx-64 -c anaconda
conda info
```

Copy the "active env location" and substitute into:

```
ls ACTIVE_ENV_LOCATION/bin/ | grep clang | grep -i 'apple'
```

Copy the two clangs and modify the following:

```
export CC=x86_64-apple-darwin13.4.0-clang
export CXX=x86_64-apple-darwin13.4.0-clang++
```

Install pystan and rlssm and test again whether the compiler works:

```
conda install -c conda-forge pystan=2.19
pip install rlssm
python tests/test_compiler.py
```

If you are experiencing other issues with the compiler, check the pystan documentation.

CHAPTER

TWO

REFERENCES

The likelihood of the diffusion decision model, or DDM, (the Wiener first passage time distribution) is already implemented in stan (see stan's manual).

The likelihood functions of the race models are implemented by us in stan, and are based on the following papers:

- Race diffusion model (RDM): Tillman, G., Van Zandtc, T., & Loganb, G. D. Psychon Bull Rev (2020), RDM paper
- Linear ballistic accumulator (LBA): Brown, S. D., & Heathcote, A. Cognitive psychology (2008), LBA paper
- Advantage LBA: van Ravenzwaaij, D., Brown, S. D., Marley, A. A. J., & Heathcote, A. Psychological review (2020), ALBA paper
- Advantage RDM: Miletić, S., Boag, R. J., Trutti, A. C., Stevenson, N., Forstmann, B. U., & Heathcote, A. (2021), ARDM paper

The RLDDMs (combinations of RL and DDM, diffusion decision model) are based on the following papers:

- Fontanesi, L., Gluth, S., Spektor, M.S. & Rieskamp, J. Psychon Bull Rev (2019), RLDDM paper 1
- Fontanesi, L., Palminteri, S. & Lebreton, M. Cogn Affect Behav Neurosci (2019), RLDDM paper 2

For a more in-depth explanation of the model's cognitive mechanisms and theories, please refer to https://osf.io/fbpmh/.

CHAPTER

THREE

CREDITS

This package was developed by me, **Laura Fontanesi**, with the help of **Amir Hosein Hadian Rasanan**. It is part of ongoing scientific work at the University of Basel and University of Hamburg, in collaboration with Prof. Jörg Rieskamp and Prof. Sebastian Gluth.

When using this package or part of the code for your own research, I ask you to **cite us (using the Zenodo DOI below)** and to always perform a parameter recovery, to make sure that everything is working as it should. Even though we tested all functions, I cannot assure that it will work perfectly on all platforms and data formats. Therefore, feedback on usage and issues is especially welcomed at this stage :)

If you have any questions or would like to contribute, you can write me at laura.fontanesi@unibas.ch.

CHAPTER

FOUR

HOW TO INITIALIZE A MODEL

To initialize a model, you can use one of the following model classes:

1. For simple reinforcement learning models: `RLModel_2A`
2. For diffusion decision models: `DDModel`
3. For reinforcement learning diffusion decision models: `RLDDModel`
4. For race models: `RDModel_2A`, `LBAModel_2A`, `ARDModel_2A`, `ALBAModel_2A`
5. For reinforcement learning race models: `RLRDModel_2A`, `RLLBAModel_2A`, `RLARDModel_2A`, `RLALBAModel_2A`

All these classes have 1 non-default argument: `hierarchical_levels`. This should be set to 1 for model fits on individual data, and 2 for model fits on group data.

Additional arguments can be specified in order to "turn on and off" different model mechanisms that are implemented.

For example, let's say I want to specify a **RLDDM for 1 subject**:

[1]:
```
import rlssm
```

[2]:
```
model = rlssm.RLDDModel(hierarchical_levels=1)
```

```
Using cached StanModel
```

After initialization, you can inspect the model's default priors. You can change these when you fit the model.

[3]:
```
model.priors
```

[3]:
```
{'alpha_priors': {'mu': 0, 'sd': 1},
 'drift_scaling_priors': {'mu': 1, 'sd': 50},
 'threshold_priors': {'mu': 1, 'sd': 5},
 'ndt_priors': {'mu': 1, 'sd': 1}}
```

Note that, if this is the first time that you initialize this type of model, it's going to take some time to compile it. Otherwise, the cashed model will be automatically loaded.

By default, all mechanisms are "off", meaning that the simplest model is fit, so you need to set alternative mechanisms to `True` to have them included in your model:

[4]:
```
model_complex = rlssm.DDModel(hierarchical_levels=1,
                              drift_variability = True,
                              starting_point_variability=True)
```

```
Using cached StanModel
```

[5]:
```
model_complex.priors
```

[5]:
```
{'threshold_priors': {'mu': 0, 'sd': 5},
 'ndt_priors': {'mu': 0, 'sd': 5},
 'drift_trialmu_priors': {'mu': 1, 'sd': 5},
 'drift_trialsd_priors': {'mu': 0, 'sd': 5},
 'rel_sp_trialmu_priors': {'mu': 0, 'sd': 0.8},
 'rel_sp_trialsd_priors': {'mu': 0, 'sd': 0.5}}
```

You can check what are the possible mechanisms for each class in the API reference guide, or by typing:

[6]:
```
rlssm.DDModel?
```

```
Init signature:
rlssm.DDModel(
    hierarchical_levels,
    starting_point_bias=False,
    drift_variability=False,
    starting_point_variability=False,
    drift_starting_point_correlation=False,
    drift_starting_point_beta_correlation=False,
    drift_starting_point_regression=False,
)
Docstring:
DDModel allows to specify a diffusion decision model.

When initializing the model, you should specify whether the model is hierarchical or not.
Additionally, you can specify the mechanisms that you wish to include or exclude.

The underlying stan model will be compiled if no previously compiled model is found.
After initializing the model, it can be fitted to a particular dataset using pystan.
Init docstring:
Initialize a DDModel object.

Note
----
This model is restricted to two options per trial (coded as correct and incorrect).

Parameters
----------

hierarchical_levels : int
    Set to 1 for individual data and to 2 for grouped data.

starting_point_bias : bool, default False
    By default, there is no starting point bias.
    If set to True, the starting point bias is estimated.

drift_variability : bool, default False
    By default, there is no drift-rate variability across trials.
    If set to True, the standard deviation of the drift-rate across trials is estimated.

starting_point_variability : bool, default False
    By default, there is no starting point bias variability across trials.
```

(continues on next page)

(continued from previous page)

```
    If set to True, the standard deviation of the starting point bias across trials
    is estimated.

drift_starting_point_correlation : bool, default False
    By default, the correlation between these 2 parameters is not estimated.
    If set to True, the 2 parameters are assumed to come
    from a multinormal distribution.
    Only relevant when drift_variability and starting_point_variability are True.

drift_starting_point_beta_correlation : bool, default False
    If True, trial-by-trial drift-rate, rel_sp and an external
    variable beta are assumed to come from a multinormal distribution.
         Only relevant when drift_variability and starting_point_variability are True.

drift_starting_point_regression : bool, default False
    If True, two regression coefficients are estimated, for trial drift
    and relative starting point, and an external variable beta.
    Only relevant when drift_variability and starting_point_variability are True.

Attributes
----------
model_label : str
    The label of the fully specified model.

n_parameters_individual : int
    The number of individual parameters of the fully specified model.

n_parameters_trial : int
    The number of parameters that are estimated at a trial level.

stan_model_path : str
    The location of the stan model code.

compiled_model : pystan.StanModel
    The compiled stan model.
File:           ~/git/rlssm/rlssm/models_DDM.py
Type:           type
Subclasses:
```

CHAPTER

FIVE

HOW TO FIT A MODEL

To fit a model, there are 3 main things to specify:

1. Data: The `data`, which should be in the form of a pandas DataFrame.

Different model classes might require different columns in the data. You should check in the API Reference of each model class (or using `model.fit?`) what the required data columns are.

2. The priors (optional): You can decide whether to use the default priors (which you can see after initializing the model) or whether you want to change the mean or SD of the prior or hyper-prior distributions. Whether you changed the priors or not, they are always printed out when the model starts fitting.

3. Sampling parameters: The sampling parameters **(number of chains, iterations, warmups, thinning, etc.)** are the arguments to the `pystan.StanModel.sampling()` function, and we simply refer to the pystan documentation for a better overview.

Additional learning parameters: While all sequential sampling models (DDM and race models) **without a learning component** only require a `data` argument, all models with a learning components (RL models, RLDDMs, and RL+race models) also require a `K` argument, which is the total number of different options in a learning block (note that this can be different from the number of options presented in each trial), and `initial_value_learning`, which is the initial Q value (before learning).

[1]:
```
import rlssm
```

5.1 Non-learning example (non-hierarchical, simulated data)

[2]:
```
model_ddm = rlssm.DDModel(hierarchical_levels=1)
```

```
Using cached StanModel
```

[3]:
```
# simulate some DDM data:
from rlssm.random import simulate_ddm
data_ddm = simulate_ddm(
    n_trials=400,
    gen_drift=.8,
    gen_threshold=1.3,
    gen_ndt=.23)
```

For the simple, non-hierarchical DDM, it is only necessary to have `rt` and `accuracy` columns:

- *rt*, response times in seconds.
- *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

[4]:
```
data_ddm.head()
```

[4]:
```
                    drift  rel_sp  threshold   ndt     rt  accuracy
participant trial
1           1         0.8     0.5        1.3  0.23  1.134       0.0
            2         0.8     0.5        1.3  0.23  0.433       1.0
            3         0.8     0.5        1.3  0.23  0.702       1.0
            4         0.8     0.5        1.3  0.23  0.695       1.0
            5         0.8     0.5        1.3  0.23  0.351       1.0
```

[5]:
```
# Run 2 chains, with 2000 samples each, 1000 of which warmup, with custom priors:
model_fit_ddm = model_ddm.fit(
    data_ddm,
    drift_priors={'mu':.5, 'sd':1},
    threshold_priors={'mu':0, 'sd':.5},
    ndt_priors={'mu':0, 'sd':.1},
    chains=2,
    iter=2000,
    warmup=1000,
    thin=1)
```

```
Fitting the model using the priors:
drift_priors {'mu': 0.5, 'sd': 1}
threshold_priors {'mu': 0, 'sd': 0.5}
ndt_priors {'mu': 0, 'sd': 0.1}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
```

```
Checks MCMC diagnostics:
n_eff / iter looks reasonable for all parameters
0.0 of 2000 iterations ended with a divergence (0.0%)
0 of 2000 iterations saturated the maximum tree depth of 10 (0.0%)
E-BFMI indicated no pathological behavior
```

5.2 Learning example (hierarchical, real data)

[6]:
```
model_rl = rlssm.RLModel_2A(hierarchical_levels = 2)
```

```
Using cached StanModel
```

[7]:
```
# import some example data:
data_rl = rlssm.load_example_dataset(hierarchical_levels = 2)

data_rl.head()
```

[7]:
```
   participant  block_label  trial_block  f_cor  f_inc  cor_option  \
0            1            1            1     43     39           2
1            1            1            2     60     50           4
2            1            1            3     44     36           4
3            1            1            4     55     55           4
4            1            1            5     52     49           4
```

(continues on next page)

(continued from previous page)

```
   inc_option  times_seen        rt  accuracy
0           1           1  1.244082         0
1           3           1  1.101821         1
2           2           2  1.029923         0
3           3           2  1.368007         0
4           3           3  1.039329         1
```

Since this learning model is only fit on choices, `rt` are not required.

Other columns/indexes that should be included are:

- *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.
- *trial_block*, the number of trial in a learning session. Should be integers starting from 1.
- *f_cor*, the output from the correct option in the presented pair (the option with higher outcome on average).
- *f_inc*, the output from the incorrect option in the presented pair (the option with lower outcome on average).
- *cor_option*, the number identifying the correct option in the presented pair (the option with higher outcome on average).
- *inc_option*, the number identifying the incorrect option in the presented pair(the option with lower outcome on average).
- *block_label*, the number identifying the learning session. Should be integers starting from 1. Set to 1 in case there is only one learning session.

If the model is hierarchical, also include:

- *participant*, the participant number. Should be integers starting from 1.

If increasing_sensitivity is True, also include:

- *times_seen*, average number of times the presented options have been seen in a learning session.

[8]:
```
data_rl.head()
```

[8]:
```
   participant  block_label  trial_block  f_cor  f_inc  cor_option  \
0            1            1            1     43     39           2
1            1            1            2     60     50           4
2            1            1            3     44     36           4
3            1            1            4     55     55           4
4            1            1            5     52     49           4

   inc_option  times_seen        rt  accuracy
0           1           1  1.244082         0
1           3           1  1.101821         1
2           2           2  1.029923         0
3           3           2  1.368007         0
4           3           3  1.039329         1
```

[9]:
```
# Run 2 chains, with 3000 samples each, 1000 of which warmup, with thinning and custom␣
↪priors:
model_fit_rl = model_rl.fit(
    data_rl,
    K=4,
```

(continues on next page)

(continued from previous page)

```
    initial_value_learning=27.5,
    alpha_priors={'mu_mu':-.3, 'sd_mu':.1, 'mu_sd':0, 'sd_sd':.1},
    sensitivity_priors={'mu_mu':-.1, 'sd_mu':.1, 'mu_sd':0, 'sd_sd':.1},
    chains=2,
    iter=3000,
    warmup=1000,
    print_diagnostics=False, # (not suggested, see below)
    thin=2)
```

```
Fitting the model using the priors:
alpha_priors {'mu_mu': -0.3, 'sd_mu': 0.1, 'mu_sd': 0, 'sd_sd': 0.1}
sensitivity_priors {'mu_mu': -0.1, 'sd_mu': 0.1, 'mu_sd': 0, 'sd_sd': 0.1}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
```

5.3 Diagnostics

As you can see, the MCMC diagnostics are already printed by default (if you do not want this, you can set `print_diagnostics` to `False`). I refer to https://mc-stan.org/users/documentation/case-studies/divergences_and_bias.html for an excellent explanation of what these diagnostics actually mean and how to assess them.

On top of these, you can also check the convergence of the chains and the WAIC:

[10]:
```
model_fit_ddm.rhat
```

[10]:
```
       rhat   variable
0  1.000630      drift
1  0.999847  threshold
2  0.999237        ndt
```

[11]:
```
model_fit_rl.rhat.describe()
```

[11]:
```
            rhat
count  58.000000
mean    1.000025
std     0.000724
min     0.999071
25%     0.999492
50%     0.999797
75%     1.000427
max     1.001968
```

[12]:
```
model_fit_ddm.waic
```

[12]:
```
{'lppd': -201.4814621446239,
 'p_waic': 3.304311118929816,
 'waic': 409.5715465271074,
 'waic_se': 45.969396831225254}
```

[13]:
```
model_fit_rl.waic
```

[13]:
```
{'lppd': -2632.8620973496854,
 'p_waic': 53.04664551553182,
 'waic': 5371.817485730435,
 'waic_se': 94.06444827215812}
```

If you want to also see the point-wise WAIC, you can set `pointwise_waic` to `True`.

5.4 Save the results

By default, the model fit results are saved in the same folder, using the `model_label` as filename. you can specify a different location using the `filename` argument.

[14]:
```
model_fit_ddm.to_pickle()
```

```
Saving file as: /Users/laurafontanesi/git/rlssm/docs/notebooks/DDM.pkl
```

[15]:
```
model_fit_rl.to_pickle()
```

```
Saving file as: /Users/laurafontanesi/git/rlssm/docs/notebooks/hierRL_2A.pkl
```

5.5 Re-load previously saved results

[16]:
```
model_fit_ddm = rlssm.load_model_results('/Users/laurafontanesi/git/rlssm/docs/notebooks/
↪DDM.pkl')
```

[17]:
```
model_fit_rl = rlssm.load_model_results('/Users/laurafontanesi/git/rlssm/docs/notebooks/
↪hierRL_2A.pkl')
```

The data the model was fit on are stored in `data_info`:

[18]:
```
model_fit_rl.data_info['data']
```

[18]:

	index	participant	block_label	trial_block	f_cor	f_inc	cor_option	inc_option	times_seen	rt	accuracy
0	0	1	1	1	43	39	2	1	1	1.244082	0
1	1	1	1	2	60	50	4	3	1	1.101821	1
2	2	1	1	3	44	36	4				
3	3	1	1	4	55	55	4				
4	4	1	1	5	52	49	4				
...	...	...	...	...	...	...	...				
6459	6459	27	3	76	37	36	2				
6460	6460	27	3	77	58	41	4				
6461	6461	27	3	78	64	49	4				
6462	6462	27	3	79	44	37	3				
6463	6463	27	3	80	51	51	4				

(continues on next page)

(continued from previous page)

```
2              2           2  1.029923         0
3              3           2  1.368007         0
4              3           3  1.039329         1
...          ...         ...       ...       ...
6459           1          39  1.875327         1
6460           2          39  1.696957         1
6461           3          38  2.059956         1
6462           1          39  1.623731         1
6463           3          40  1.115363         1

[6464 rows x 11 columns]
```

The priors are stored in `priors`:

[19]:
```
model_fit_ddm.priors
```

[19]:
```
{'drift_priors': {'mu': 0.5, 'sd': 1},
 'threshold_priors': {'mu': 0, 'sd': 0.5},
 'ndt_priors': {'mu': 0, 'sd': 0.1}}
```

[20]:
```
model_fit_rl.priors
```

[20]:
```
{'alpha_priors': {'mu_mu': -0.3, 'sd_mu': 0.1, 'mu_sd': 0, 'sd_sd': 0.1},
 'sensitivity_priors': {'mu_mu': -0.1, 'sd_mu': 0.1, 'mu_sd': 0, 'sd_sd': 0.1}}
```

And different parameter information are stored in `parameter_info`:

[21]:
```
model_fit_ddm.parameters_info
```

[21]:
```
{'hierarchical_levels': 1,
 'n_parameters_individual': 3,
 'n_parameters_trial': 0,
 'n_posterior_samples': 2000,
 'parameters_names': ['drift', 'threshold', 'ndt'],
 'parameters_names_transf': ['transf_drift', 'transf_threshold', 'transf_ndt'],
 'parameters_names_all': ['drift', 'threshold', 'ndt']}
```

[22]:
```
model_fit_rl.waic
```

[22]:
```
{'lppd': -2632.8620973496854,
 'p_waic': 53.04664551553182,
 'waic': 5371.817485730435,
 'waic_se': 94.06444827215812}
```

And, of course, you can inspect the model's posteriors, see How to inspect a model for more details on this.

CHAPTER

SIX

HOW TO INSPECT MODEL FIT RESULTS

[1]:
```
import rlssm

# load non-hierarchical DDM fit:
model_fit_ddm = rlssm.load_model_results('/Users/laurafontanesi/git/rlssm/docs/notebooks/
↪DDM.pkl')

# load non-hierarchical LBA fit:
model_fit_lba = rlssm.load_model_results('/Users/laurafontanesi/git/rlssm/docs/notebooks/
↪LBA_2A.pkl')

# load hierarchical RL fit:
model_fit_rl = rlssm.load_model_results('/Users/laurafontanesi/git/rlssm/docs/notebooks/
↪hierRL_2A.pkl')
```

6.1 Posteriors

The posterior samples are stored in `samples`:

[2]:
```
model_fit_ddm.samples
```

[2]:
```
      chain  draw  transf_drift  transf_threshold  transf_ndt
0         0   339       0.603124          1.290092    0.239343
1         0   784       0.795615          1.308542    0.240124
2         0   250       0.766659          1.299046    0.238052
3         0   498       0.909609          1.293544    0.242449
4         0   854       0.894703          1.304357    0.235697
...     ...   ...            ...               ...         ...
1995      1   482       0.893670          1.245259    0.245646
1996      1   882       0.862621          1.228198    0.248502
1997      1   571       0.732467          1.281621    0.235845
1998      1   403       0.767313          1.278530    0.243468
1999      1   998       0.812562          1.285636    0.242826

[2000 rows x 5 columns]
```

[3]:
```
model_fit_rl.samples.describe()
```

[3]:
```
             chain         draw  transf_mu_alpha  transf_mu_sensitivity  \
count  2000.000000  2000.000000      2000.000000            2000.000000
```

(continues on next page)

(continued from previous page)

```
mean      0.500000   499.500000         0.215143               0.459187
std       0.500125   288.747186         0.026105               0.031864
min       0.000000     0.000000         0.137208               0.371670
25%       0.000000   249.750000         0.197125               0.437836
50%       0.500000   499.500000         0.214402               0.457982
75%       1.000000   749.250000         0.231833               0.478926
max       1.000000   999.000000         0.322323               0.602965

       alpha_sbj[1]  alpha_sbj[2]  alpha_sbj[3]  alpha_sbj[4]  alpha_sbj[5]  \
count   2000.000000   2000.000000   2000.000000   2000.000000   2000.000000
mean       0.118446      0.038598      0.153991      0.125686      0.228167
std        0.049484      0.046423      0.070357      0.053255      0.081044
min        0.013977      0.004837      0.016532      0.021659      0.030638
25%        0.081938      0.017878      0.103897      0.087518      0.172536
50%        0.111285      0.027213      0.142031      0.118346      0.222287
75%        0.146284      0.043262      0.191607      0.154902      0.278837
max        0.386474      0.812427      0.538177      0.378176      0.540157

       alpha_sbj[6]  ...  sensitivity_sbj[18]  sensitivity_sbj[19]  \
count   2000.000000  ...          2000.000000          2000.000000
mean       0.176434  ...             0.186011             0.222761
std        0.086367  ...             0.142018             0.035984
min        0.016090  ...             0.033029             0.134701
25%        0.111644  ...             0.074628             0.197551
50%        0.162954  ...             0.127833             0.218959
75%        0.228109  ...             0.266526             0.242577
max        0.527946  ...             0.879778             0.582695

       sensitivity_sbj[20]  sensitivity_sbj[21]  sensitivity_sbj[22]  \
count          2000.000000          2000.000000          2000.000000
mean              0.345018             0.675645             0.594302
std               0.051357             0.129893             0.112983
min               0.226670             0.383286             0.293426
25%               0.308241             0.585866             0.514762
50%               0.340830             0.661773             0.582411
75%               0.375535             0.752205             0.657723
max               0.697668             1.244169             1.158730

       sensitivity_sbj[23]  sensitivity_sbj[24]  sensitivity_sbj[25]  \
count          2000.000000          2000.000000          2000.000000
mean              0.733052             0.380368             0.400283
std               0.162488             0.062239             0.142786
min               0.361626             0.216227             0.165044
25%               0.616664             0.336826             0.297846
50%               0.714372             0.373471             0.366054
75%               0.831164             0.416691             0.473252
max               1.626704             0.705096             1.150375

       sensitivity_sbj[26]  sensitivity_sbj[27]
count          2000.000000          2000.000000
mean              0.207491             0.255721
std               0.119114             0.064612
```

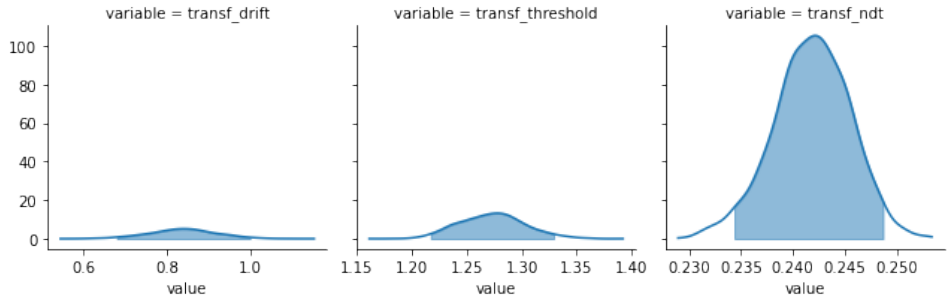
(continues on next page)

(continued from previous page)

```
min             0.050049            0.129987
25%             0.116860            0.211426
50%             0.177753            0.242781
75%             0.266407            0.286889
max             0.990639            0.711207

[8 rows x 58 columns]
```

You can simply plot the model's posteriors using `plot_posteriors`:

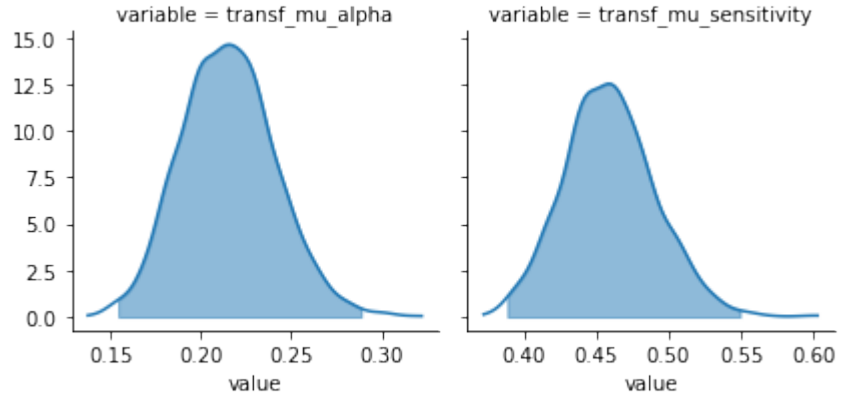
[4]:
```
model_fit_ddm.plot_posteriors();
```



nbsphinx-code-borderwhite

By default, 95% HDIs are shown, but you can also choose to have the posteriors without intervals or BCIs, and change the alpha level:

[5]:
```
model_fit_rl.plot_posteriors(show_intervals='BCI', alpha_intervals=.01);
```



nbsphinx-code-borderwhite

6.1.1 Trial-level

Depending on the model specification, you can also extract certain trial-level parameters as numpy ordered dictionaries of n_samples X n_trials shape:

[6]:
```
model_fit_ddm.trial_samples['drift_t'].shape
```

[6]:
```
(2000, 400)
```

[7]:
```
model_fit_ddm.trial_samples.keys()
```

[7]:
```
odict_keys(['drift_t', 'threshold_t', 'ndt_t'])
```

[8]:
```
model_fit_lba.trial_samples.keys() # for the LBA
```

[8]:
```
odict_keys(['k_t', 'A_t', 'tau_t', 'drift_cor_t', 'drift_inc_t'])
```

In the case of a RL model fit on choices alone, you can extract the log probability of accuracy=1 for each trial:

[9]:
```
model_fit_rl.trial_samples.keys()
```

[9]:
```
odict_keys(['log_p_t'])
```

[10]:
```
model_fit_rl.trial_samples['log_p_t'].shape
```

[10]:
```
(2000, 6464)
```

6.2 Posterior predictives

With `get_posterior_predictives_df` you get posterior predictives as pandas DataFrames of `n_posterior_predictives` X `n_trials` shape:

[11]:
```
pp = model_fit_rl.get_posterior_predictives_df(n_posterior_predictives=1000)
pp
```

[11]:
```
variable accuracy                                                 ...            \
trial           1    2    3    4    5    6    7    8    9    10  ... 6455 6456
sample                                                            ...
1               1    1    1    1    1    1    0    0    1    0  ...    0    1
2               0    1    1    1    1    1    0    1    1    1  ...    0    0
3               1    1    0    0    0    1    1    0    1    1  ...    0    1
4               0    1    0    1    0    1    1    1    1    1  ...    0    0
5               1    1    1    0    1    1    0    0    1    1  ...    0    1
...           ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
996             1    1    0    0    0    1    0    0    1    1  ...    1    0
997             1    0    0    1    1    0    1    0    1    1  ...    0    0
998             0    0    0    1    1    1    1    0    1    1  ...    1    1
999             1    1    0    0    1    1    1    1    1    1  ...    0    0
1000            0    0    1    1    0    1    0    1    1    1  ...    0    0

variable
trial    6457 6458 6459 6460 6461 6462 6463 6464
sample
1           1    1    1    1    1    1    1    1
2           1    1    1    1    1    1    1    1
3           1    0    0    0    1    1    1    1
4           1    1    1    1    1    1    1    1
5           1    1    1    1    1    1    1    0
...       ...  ...  ...  ...  ...  ...  ...  ...
996         1    1    1    1    1    1    1    1
997         1    0    0    0    1    0    0    1
998         1    0    1    0    1    1    1    0
999         0    1    1    1    1    0    1    1
```

(continues on next page)

(continued from previous page)

```
1000        1    1    1    1    1    1    1    1

[1000 rows x 6464 columns]
```

For the DDM, you have additional parameters to tweak the DDM simulations, and you get a DataFrame with a hierarchical column index, for RTs and for accuracy:

[12]:
```
pp = model_fit_ddm.get_posterior_predictives_df(n_posterior_predictives=100, dt=.001)
pp
```

[12]:
```
variable        rt                                                    \
trial          1         2         3         4         5         6
sample
1         0.383343  0.662343  0.639343  0.748343  0.775343  1.179343
2         0.994124  0.333124  0.549124  0.848124  1.027124  0.310124
3         1.307052  0.686052  0.457052  0.343052  0.367052  0.612052
4         0.642449  0.535449  0.388449  0.470449  0.649449  0.523449
5         0.842697  1.535697  0.721697  0.316697  0.708697  1.516697
...            ...       ...       ...       ...       ...       ...
96        0.496667  0.334667  0.872667  0.771667  0.389667  0.434667
97        0.328090  0.558090  0.444090  0.516090  1.467090  0.736090
98        0.424736  0.395736  0.576736  0.411736  0.324736  0.497736
99        1.000664  0.364664  0.521664  0.370664  0.387664  0.732664
100       0.956275  0.396275  0.968275  0.416275  0.418275  1.048275

variable                                          ... accuracy                \
trial          7         8         9         10   ...      391  392  393  394
sample                                            ...
1         1.619343  0.398343  1.047343  0.329343  ...      0.0  1.0  1.0  0.0
2         0.425124  0.601124  0.476124  0.536124  ...      1.0  0.0  1.0  1.0
3         0.284052  0.650052  0.381052  0.615052  ...      1.0  1.0  1.0  1.0
4         0.338449  0.557449  0.581449  0.606449  ...      1.0  1.0  0.0  0.0
5         0.363697  0.751697  0.417697  0.544697  ...      0.0  1.0  1.0  1.0
...            ...       ...       ...       ...  ...      ...  ...  ...  ...
96        0.425667  1.027667  0.433667  0.358667  ...      1.0  1.0  1.0  1.0
97        1.013090  0.312090  0.405090  0.328090  ...      1.0  0.0  1.0  1.0
98        0.520736  0.741736  0.510736  1.300736  ...      0.0  1.0  1.0  1.0
99        0.794664  0.369664  0.780664  0.353664  ...      1.0  1.0  0.0  1.0
100       1.015275  0.706275  0.383275  0.395275  ...      1.0  1.0  1.0  1.0

variable
trial     395  396  397  398  399  400
sample
1         1.0  1.0  1.0  0.0  1.0  1.0
2         1.0  1.0  1.0  0.0  1.0  0.0
3         0.0  1.0  0.0  1.0  0.0  1.0
4         0.0  1.0  1.0  1.0  0.0  1.0
5         0.0  1.0  1.0  1.0  1.0  1.0
...       ...  ...  ...  ...  ...  ...
96        1.0  0.0  1.0  1.0  1.0  1.0
97        1.0  1.0  1.0  1.0  0.0  0.0
98        1.0  0.0  1.0  1.0  1.0  0.0
99        0.0  1.0  1.0  0.0  1.0  0.0
```

(continues on next page)

(continued from previous page)

```
100       1.0  0.0  1.0  1.0  1.0  1.0

[100 rows x 800 columns]
```

You can also have posterior predictive summaries with `get_posterior_predictives_summary`.

Only mean accuracy for RL models fit on choices alone, and also mean RTs, skewness and quantiles for lower and upper boundaries for models fitted on RTs as well.

[13]:
```
model_fit_rl.get_posterior_predictives_summary()
```

[13]:
```
        mean_accuracy
sample
1            0.799814
2            0.802754
3            0.803373
4            0.800743
5            0.806312
...               ...
496          0.804301
497          0.809561
498          0.807550
499          0.800588
500          0.793472

[500 rows x 1 columns]
```

[14]:
```
model_fit_ddm.get_posterior_predictives_summary()
```

[14]:
```
        mean_accuracy   mean_rt  skewness  quant_10_rt_low  quant_30_rt_low  \
sample
1              0.6650  0.670093  2.045784         0.346243         0.460143
2              0.7500  0.658724  1.576743         0.392724         0.471524
3              0.7025  0.669302  2.188532         0.363052         0.480652
4              0.7875  0.640931  2.890019         0.361849         0.446049
5              0.7625  0.632817  1.456125         0.338897         0.389897
...               ...       ...       ...              ...              ...
496            0.6950  0.615908  1.971778         0.352350         0.408850
497            0.8025  0.651565  2.103909         0.376220         0.435420
498            0.7525  0.604910  1.602822         0.328585         0.448785
499            0.7425  0.624212  1.663869         0.346202         0.433202
500            0.7625  0.636005  2.907804         0.344582         0.429582

        quant_50_rt_low  quant_70_rt_low  quant_90_rt_low  quant_10_rt_up  \
sample
1              0.568843         0.761643         1.152343        0.351843
2              0.565624         0.725624         1.049824        0.348024
3              0.589052         0.779852         1.210852        0.348052
4              0.526449         0.699049         1.110649        0.363849
5              0.481697         0.626897         1.060897        0.361097
...                 ...              ...              ...             ...
496            0.505750         0.665450         0.982550        0.337250
497            0.567020         0.728620         1.113820        0.352020
```

(continues on next page)

(continued from previous page)

```
498            0.573785         0.717585         0.966385        0.340785
499            0.536402         0.619802         0.957202        0.347602
500            0.522382         0.681182         1.085382        0.350382

        quant_30_rt_up  quant_50_rt_up  quant_70_rt_up  quant_90_rt_up
sample
1             0.434843        0.550343        0.720843        1.103843
2             0.423524        0.567624        0.703324        1.168924
3             0.455052        0.546052        0.723052        1.068052
4             0.432649        0.543449        0.695449        0.987649
5             0.442697        0.559697        0.754097        1.007497
...                ...             ...             ...             ...
496           0.435350        0.543750        0.687050        1.004750
497           0.434020        0.536020        0.710020        1.085020
498           0.429785        0.519785        0.662785        0.956785
499           0.451402        0.552402        0.722202        1.003002
500           0.440582        0.537382        0.698582        1.021982

[500 rows x 13 columns]
```

You can also specify which quantiles you are interested in:

[15]:
```
model_fit_lba.get_posterior_predictives_summary(n_posterior_predictives=200, quantiles=[.
↪1, .5, .9])
```

[15]:
```
        mean_accuracy     mean_rt   skewness  quant_10_rt_incorrect  \
sample
1            0.858333    1.721248   1.746695               1.477207
2            0.850000    1.711749   2.070015               1.353934
3            0.812500    1.737245   1.874562               1.516550
4            0.875000    1.629739   1.314279               1.337575
5            0.841667    1.648190   2.309592               1.304721
...               ...         ...        ...                    ...
196          0.883333    1.719413   2.057477               1.576292
197          0.908333    1.623029   1.669531               1.286818
198          0.891667  167.991497  15.491932               1.290854
199          0.904167    1.672464   4.330050               1.579026
200          0.891667    1.662455   1.442103               1.374941

        quant_50_rt_incorrect  quant_90_rt_incorrect  quant_10_rt_correct  \
sample
1                    1.835347               2.437212             1.289665
2                    1.743663               2.418908             1.249635
3                    1.909714               2.711531             1.215915
4                    1.594347               2.075680             1.219060
5                    1.557302               2.144594             1.228440
...                       ...                    ...                  ...
196                  1.939482               2.982563             1.218782
197                  1.684597               2.031900             1.211736
198                  1.626846               3.500035             1.209622
199                  2.011868               2.988423             1.209593
200                  1.719873               2.295363             1.249559
```

(continues on next page)

(continued from previous page)

```
        quant_50_rt_correct  quant_90_rt_correct
sample
1                  1.589248             2.239182
2                  1.564929             2.175558
3                  1.550399             2.272305
4                  1.536317             2.125802
5                  1.531487             2.100375
...                     ...                  ...
196                1.544592             2.283167
197                1.512554             2.172847
198                1.526332             2.445680
199                1.535032             1.992702
200                1.578828             2.195988

[200 rows x 9 columns]
```

Finally, you can get summary for grouping variables (e.g., experimental conditions, trial blocks, etc.) in your data:

[16]:
```
model_fit_lba.get_grouped_posterior_predictives_summary(n_posterior_predictives=200,
                                                        grouping_vars=['block_label'],
                                                        quantiles=[.3, .5, .7])
```

[16]:
```
                    mean_accuracy   mean_rt  skewness  quant_30_rt_incorrect  \
block_label sample
1           1              0.8875  1.742559  0.838222               1.958646
            2              0.8250  1.667581  2.036401               1.490616
            3              0.7875  1.727859  1.586878               1.890115
            4              0.8500  1.683817  6.404095               1.440239
            5              0.9000  1.625880  2.086927               1.694815
...                           ...       ...       ...                    ...
3           196            0.9000  1.725673  1.756656               1.776924
            197            0.8500  1.699036  2.265547               1.630694
            198            0.9125  1.772827  3.916904               1.515939
            199            0.8375  1.717850  2.060856               1.609815
            200            0.8500  1.685803  0.829624               1.469060

                    quant_30_rt_correct  quant_50_rt_incorrect  \
block_label sample
1           1                  1.402651               2.265617
            2                  1.386556               1.690442
            3                  1.314495               2.045244
            4                  1.336986               1.638169
            5                  1.334574               1.804016
...                                 ...                    ...
3           196                1.402891               1.912637
            197                1.350169               1.696332
            198                1.358824               1.594244
            199                1.404633               1.786825
            200                1.422103               1.654015

                    quant_50_rt_correct  quant_70_rt_incorrect  \
block_label sample
1           1                  1.637518               2.414924
```

(continues on next page)

(continued from previous page)

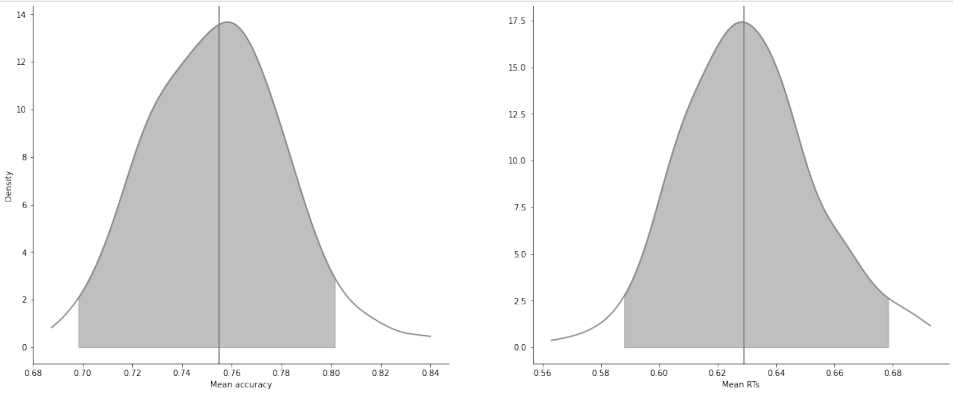
```
            2                   1.539333                1.996045
            3                   1.469067                2.167337
            4                   1.467766                1.874202
            5                   1.481553                1.877418
...                                  ...                     ...
3           196                 1.556986                1.987094
            197                 1.519048                2.194706
            198                 1.478403                2.743708
            199                 1.567322                1.928092
            200                 1.567095                1.986281

                      quant_70_rt_correct
block_label sample
1           1                   1.812356
            2                   1.740201
            3                   1.671310
            4                   1.667109
            5                   1.693119
...                                  ...
3           196                 1.743893
            197                 1.645663
            198                 1.674592
            199                 1.769137
            200                 1.827734

[600 rows x 9 columns]
```

6.3 Plot posterior predictives

You can plot posterior predictives similarly, both **ungrouped** (across all trials) or **grouped** (across conditions, trial blocks, etc.plot_mean_posterior_predictives).

For RT models, you have both **mean plots**, and **quantile plots**:

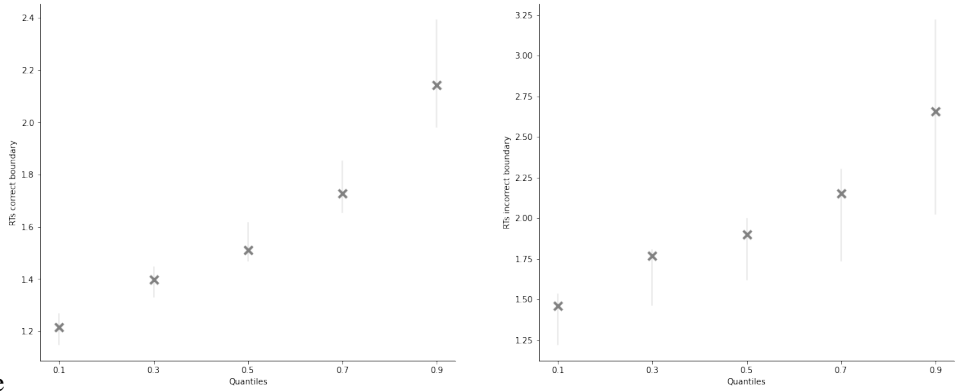
[17]:
```
model_fit_ddm.plot_mean_posterior_predictives(n_posterior_predictives=200);
```



nbsphinx-code-borderwhite

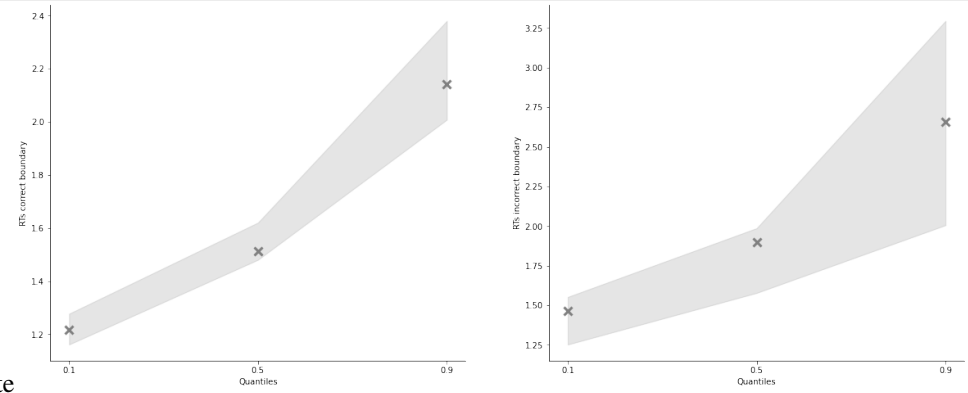
Quantile plots have 2 main visualization options, “shades” and “lines”, and you can specify again which quantiles you want, which in tervals and alpha levels:

[18]:
```
model_fit_lba.plot_quantiles_posterior_predictives(n_posterior_predictives=200);
```



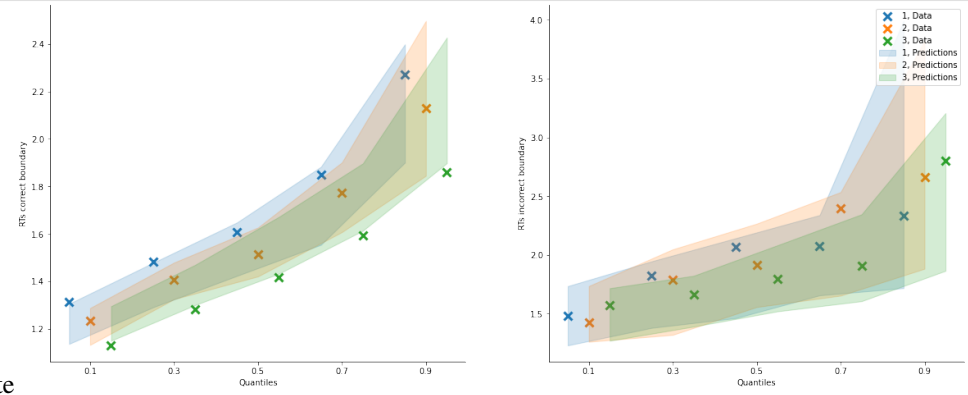
nbsphinx-code-borderwhite

[19]:
```
model_fit_lba.plot_quantiles_posterior_predictives(n_posterior_predictives=200,
                                                   kind='shades',
                                                   quantiles=[.1, .5, .9]);
```



nbsphinx-code-borderwhite

[20]:
```
model_fit_lba.plot_quantiles_grouped_posterior_predictives(
    n_posterior_predictives=100,
    grouping_var='block_label',
    kind='shades',
    quantiles=[.1, .3, .5, .7, .9]);
```



nbsphinx-code-borderwhite

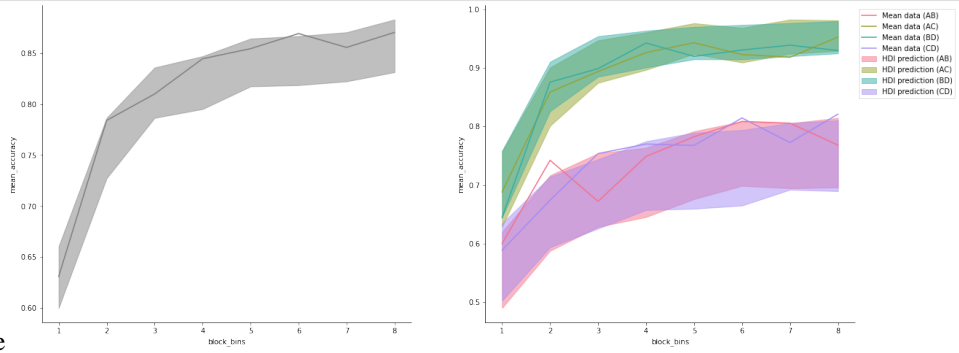
[21]:
```
# Define new grouping variables:
```

(continues on next page)

(continued from previous page)

```
import pandas as pd
import numpy as np

data = model_fit_rl.data_info['data']

# add a column to the data to group trials across learning blocks
data['block_bins'] = pd.cut(data.trial_block, 8, labels=np.arange(1, 9))

# add a column to define which choice pair is shown in that trial
data['choice_pair'] = 'AB'
data.loc[(data.cor_option == 3) & (data.inc_option == 1), 'choice_pair'] = 'AC'
data.loc[(data.cor_option == 4) & (data.inc_option == 2), 'choice_pair'] = 'BD'
data.loc[(data.cor_option == 4) & (data.inc_option == 3), 'choice_pair'] = 'CD'
```

[22]:
```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(1, 2, figsize=(20,8))

model_fit_rl.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins'], n_
↪posterior_predictives=500, ax=axes[0])

model_fit_rl.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins',
↪'choice_pair'],
                                                     n_posterior_predictives=500,␣
↪ax=axes[1])

sns.despine()
```



nbsphinx-code-borderwhite

CHAPTER

SEVEN

FIT THE DDM ON INDIVIDUAL DATA

[1]:
```
import rlssm
import pandas as pd
import os
```

7.1 Import the data

[2]:
```
data = rlssm.load_example_dataset(hierarchical_levels = 1)

data.head()
```

[2]:
```
   participant  block_label  trial_block  f_cor  f_inc  cor_option  \
0           15            1            1     50     28           3
1           15            1            2     52     44           3
2           15            1            3     30     38           2
3           15            1            4     64     45           4
4           15            1            5     48     26           3

   inc_option  times_seen        rt  accuracy
0           1           1  2.630658         1
1           1           2  2.718299         1
2           1           2  2.382882         1
3           2           1  2.167205         1
4           1           3  2.748257         0
```

7.2 Initialize the model

[3]:
```
model = rlssm.DDModel(hierarchical_levels = 1)
```

```
Using cached StanModel
```

7.3 Fit

[4]:
```
# sampling parameters
n_iter = 1000
n_chains = 2
n_thin = 1
```

[5]:
```
model_fit = model.fit(
    data,
    thin = n_thin,
    iter = n_iter,
    chains = n_chains,
    pointwise_waic=False,
    verbose = False)
```

```
Fitting the model using the priors:
drift_priors {'mu': 1, 'sd': 5}
threshold_priors {'mu': 0, 'sd': 5}
ndt_priors {'mu': 0, 'sd': 5}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
```

```
Checks MCMC diagnostics:
n_eff / iter looks reasonable for all parameters
0.0 of 1000 iterations ended with a divergence (0.0%)
0 of 1000 iterations saturated the maximum tree depth of 10 (0.0%)
E-BFMI indicated no pathological behavior
```

7.3.1 get Rhat

[6]:
```
model_fit.rhat
```

[6]:
```
       rhat   variable
0  1.000810      drift
1  1.006943  threshold
2  1.007549        ndt
```

7.3.2 get wAIC

[7]:
```
model_fit.waic
```

[7]:
```
{'lppd': -249.31901167684737,
 'p_waic': 3.0587189056624244,
 'waic': 504.7554611650196,
 'waic_se': 34.26010966730786}
```

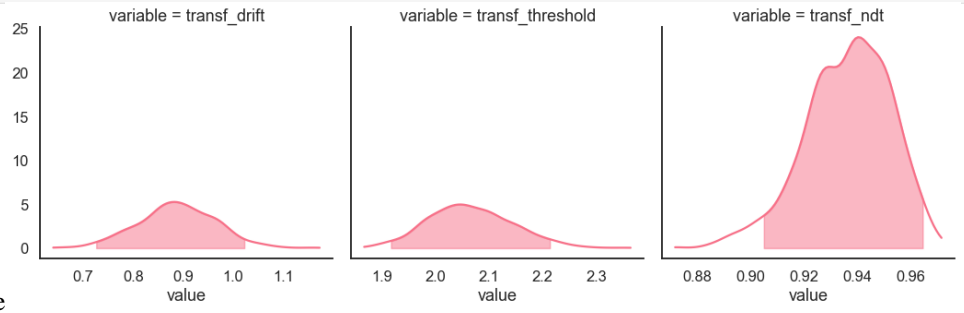
7.4 Posteriors

[8]:
```
model_fit.samples.describe()
```

[8]:
```
            chain         draw  transf_drift  transf_threshold   transf_ndt
count  1000.00000  1000.000000   1000.000000       1000.000000  1000.000000
mean      0.50000   249.500000      0.887978          2.066264     0.936474
std       0.50025   144.409501      0.077742          0.078720     0.016016
min       0.00000     0.000000      0.639946          1.866531     0.871589
25%       0.00000   124.750000      0.838996          2.012143     0.926123
50%       0.50000   249.500000      0.887060          2.060566     0.938202
75%       1.00000   374.250000      0.939397          2.117163     0.948530
max       1.00000   499.000000      1.172651          2.364307     0.971738
```

[9]:
```
import seaborn as sns
sns.set(context = "talk",
        style = "white",
        palette = "husl",
        rc={'figure.figsize':(15, 8)})
```

[10]:
```
model_fit.plot_posteriors(height=5, show_intervals="HDI", alpha_intervals=.05);
```



nbsphinx-code-borderwhite

7.5 Posterior predictives

7.5.1 Ungrouped

[11]:
```
pp = model_fit.get_posterior_predictives_df(n_posterior_predictives=100)
pp
```

[11]:
```
variable        rt                                                  \
trial            1         2         3         4         5         6
sample
1         1.223334  2.259334  1.509334  1.212334  2.187334  2.319334
2         1.305570  1.567570  1.483570  1.874570  1.748570  1.159570
3         1.400020  1.402020  3.388020  1.057020  2.142020  1.223020
4         4.672817  1.878817  1.780817  1.547817  2.339817  1.494817
5         3.312260  2.279260  2.675260  1.332260  1.425260  2.339260
...            ...       ...       ...       ...       ...       ...
96        2.024093  1.262093  2.446093  4.031093  1.437093  1.193093
97        1.300279  1.493279  1.214279  1.889279  1.611279  1.160279
```

(continues on next page)

(continued from previous page)

```
98        1.342920  1.468920  2.064920  1.563920  3.481920  1.511920
99        1.473277  2.308277  1.750277  1.995277  1.345277  1.712277
100       1.485624  1.479624  1.893624  2.016624  1.354624  2.434624

variable                                          ... accuracy                \
trial          7         8         9         10   ...      230  231  232  233
sample                                            ...
1         1.228334  2.348334  1.991334  1.349334  ...      1.0  1.0  1.0  0.0
2         1.312570  1.802570  1.703570  1.095570  ...      1.0  1.0  1.0  1.0
3         2.474020  1.597020  1.396020  1.279020  ...      1.0  1.0  1.0  1.0
4         1.835817  1.222817  2.578817  3.279817  ...      0.0  1.0  1.0  1.0
5         1.814260  1.268260  1.372260  1.549260  ...      0.0  1.0  0.0  1.0
...            ...       ...       ...       ...  ...      ...  ...  ...  ...
96        2.746093  1.450093  1.216093  2.919093  ...      1.0  1.0  1.0  1.0
97        1.265279  1.881279  3.004279  2.891279  ...      1.0  1.0  0.0  1.0
98        1.863920  1.289920  1.685920  1.471920  ...      1.0  1.0  1.0  1.0
99        1.291277  1.392277  1.486277  1.976277  ...      1.0  1.0  1.0  1.0
100       1.665624  2.441624  1.143624  1.129624  ...      1.0  1.0  0.0  1.0

variable
trial     234  235  236  237  238  239
sample
1         1.0  1.0  0.0  1.0  1.0  0.0
2         1.0  1.0  1.0  1.0  1.0  1.0
3         1.0  1.0  0.0  1.0  0.0  1.0
4         1.0  1.0  1.0  1.0  1.0  1.0
5         1.0  1.0  1.0  1.0  1.0  1.0
...       ...  ...  ...  ...  ...  ...
96        0.0  1.0  1.0  0.0  0.0  1.0
97        1.0  0.0  1.0  1.0  1.0  1.0
98        1.0  1.0  1.0  1.0  0.0  1.0
99        1.0  0.0  1.0  1.0  1.0  1.0
100       1.0  1.0  1.0  1.0  1.0  1.0

[100 rows x 478 columns]
```

[12]:
```
pp_summary = model_fit.get_posterior_predictives_summary(n_posterior_predictives=100)
pp_summary
```

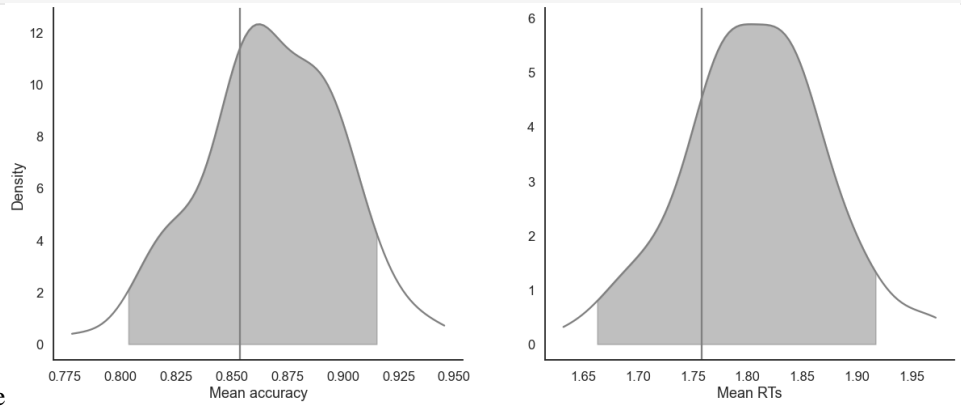
[12]:
```
        mean_accuracy   mean_rt  skewness  quant_10_rt_low  quant_30_rt_low  \
sample
1            0.878661  1.892451  2.078444         1.378734         1.538534
2            0.832636  1.769654  3.328429         1.184570         1.293470
3            0.924686  1.782321  1.752938         1.219020         1.453120
4            0.803347  1.767152  1.387015         1.225617         1.505617
5            0.895397  1.745301  2.373022         1.250060         1.413660
...               ...       ...       ...              ...              ...
96           0.849372  1.825122  2.236298         1.277593         1.400593
97           0.874477  1.747479  1.793196         1.177179         1.235679
98           0.836820  1.865740  1.573733         1.221320         1.324320
99           0.887029  1.832696  1.897374         1.208077         1.358877
100          0.870293  1.697068  2.067900         1.206624         1.279624
```

(continues on next page)

(continued from previous page)

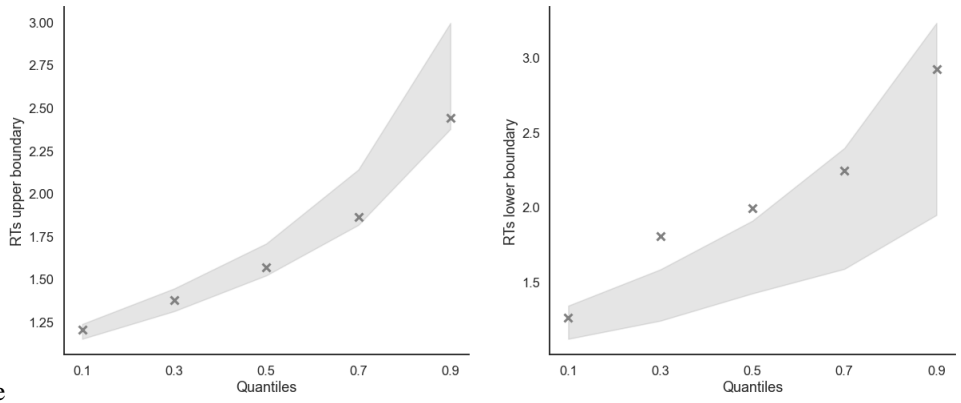
```
        quant_50_rt_low  quant_70_rt_low  quant_90_rt_low  quant_10_rt_up  \
sample
1              1.742334         2.037534         3.194934        1.212934
2              1.441070         1.742370         2.202970        1.185370
3              1.737520         1.887220         2.282020        1.230020
4              1.725817         1.867217         2.265217        1.175917
5              1.624260         1.853460         2.186860        1.179860
...                 ...              ...              ...             ...
96             1.548593         1.712593         2.730093        1.198093
97             1.505779         1.718079         2.450379        1.171479
98             1.529920         1.825520         2.247520        1.200220
99             1.609277         1.928877         2.915877        1.184377
100            1.573624         1.868624         2.586624        1.143124

        quant_30_rt_up  quant_50_rt_up  quant_70_rt_up  quant_90_rt_up
sample
1             1.432634        1.656334        2.048934        2.963734
2             1.381770        1.575570        1.923370        2.626970
3             1.360020        1.584020        1.929020        2.615020
4             1.353117        1.589317        1.994917        2.578117
5             1.352760        1.582260        1.840360        2.588060
...                ...             ...             ...             ...
96            1.389893        1.613093        2.038493        2.657093
97            1.301479        1.504279        1.795079        2.846879
98            1.433020        1.740920        2.032520        2.920520
99            1.371877        1.601277        2.000277        2.812677
100           1.328724        1.525124        1.824924        2.440624

[100 rows x 13 columns]
```

[13]:
```
model_fit.plot_mean_posterior_predictives(n_posterior_predictives=100, figsize=(20,8),␣
↪show_intervals='HDI');
```



nbsphinx-code-borderwhite

[14]:
```
model_fit.plot_quantiles_posterior_predictives(n_posterior_predictives=100, kind='shades
↪');
```



nbsphinx-code-borderwhite

7.5.2 Grouped

[15]:
```
import numpy as np
```

[16]:
```
# Define new grouping variables, in this case, for the different choice pairs, but any␣
↪grouping var can do
data['choice_pair'] = 'AB'
data.loc[(data.cor_option == 3) & (data.inc_option == 1), 'choice_pair'] = 'AC'
data.loc[(data.cor_option == 4) & (data.inc_option == 2), 'choice_pair'] = 'BD'
data.loc[(data.cor_option == 4) & (data.inc_option == 3), 'choice_pair'] = 'CD'

data['block_bins'] = pd.cut(data.trial_block, 8, labels=np.arange(1, 9))
```

[17]:
```
model_fit.get_grouped_posterior_predictives_summary(
                grouping_vars=['block_label', 'choice_pair'],
                quantiles=[.3, .5, .7],
                n_posterior_predictives=100)
```

[17]:

block_label	choice_pair	sample	mean_accuracy	mean_rt	skewness	quant_30_rt_low	quant_30_rt_up
1	AB	1	0.85	2.110784	1.247609	1.394134	1.324534
		2	0.85	1.624170	0.995058	1.831370	1.357170
		3	0.95	1.735570	1.643538	1.350020	1.374820
		4	0.75	1.557317	1.365500	1.221417	1.320417
		5	0.85	1.756110	1.313400		
...			...	...	...		
3	CD	96	0.75	1.501643	0.760884		
		97	1.00	1.679479	1.992939		
		98	1.00	1.809320	2.061042		
		99	0.95	1.526577	1.452011		
		100	0.85	1.690874	2.286446		

(continues on next page)

(continued from previous page)

```
                         5              1.313460        1.244660
...                                          ...             ...
3           CD           96             1.369893        1.277693
                         97                  NaN        1.343079
                         98                  NaN        1.290220
                         99             1.913277        1.293877
                         100            1.749624        1.329624

                                  quant_50_rt_low  quant_50_rt_up  \
block_label choice_pair sample
1           AB           1             1.427334        1.898334
                         2             2.268570        1.439570
                         3             1.350020        1.546020
                         4             1.239817        1.503817
                         5             1.338260        1.565260
...                                          ...             ...
3           CD           96            1.553093        1.382093
                         97                  NaN        1.527279
                         98                  NaN        1.469920
                         99             1.913277        1.353277
                         100            1.943624        1.404624

                                  quant_70_rt_low  quant_70_rt_up
block_label choice_pair sample
1           AB           1             1.492934        2.750934
                         2             2.271770        1.612570
                         3             1.350020        1.817820
                         4             1.509417        1.694217
                         5             1.588660        2.090260
...                                          ...             ...
3           CD           96            1.657093        1.625293
                         97                  NaN        1.597579
                         98                  NaN        1.932820
                         99             1.913277        1.658677
                         100            1.964424        1.777024

[1200 rows x 9 columns]
```

[18]:
```
model_fit.get_grouped_posterior_predictives_summary(
                grouping_vars=['block_bins'],
                quantiles=[.3, .5, .7],
                n_posterior_predictives=100)
```

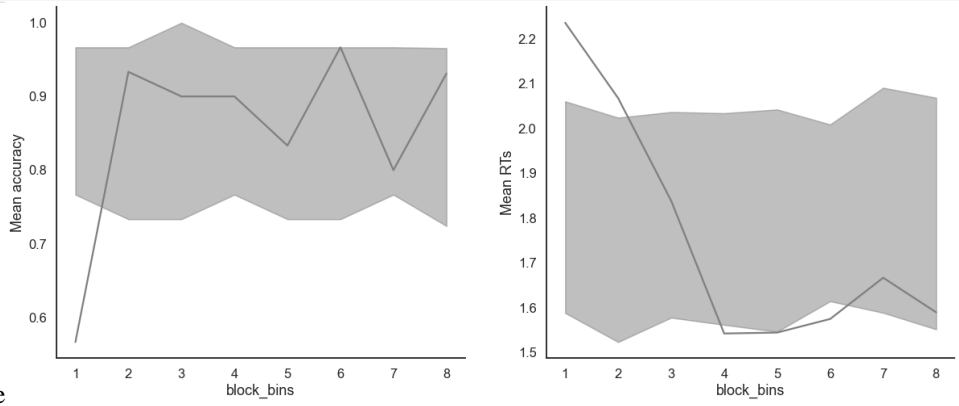
[18]:
```
                    mean_accuracy   mean_rt  skewness  quant_30_rt_low  \
block_bins sample
1          1             0.933333  1.963768  1.036143         1.866334
           2             0.833333  1.588937  2.913381         1.240770
           3             0.933333  1.896486  2.235551         1.538420
           4             0.833333  1.723484  1.037996         1.675017
           5             0.966667  1.668193  2.396271         2.047260
...                           ...       ...       ...              ...
8          96            0.689655  1.988265  1.413794         1.474693
```

(continues on next page)

(continued from previous page)

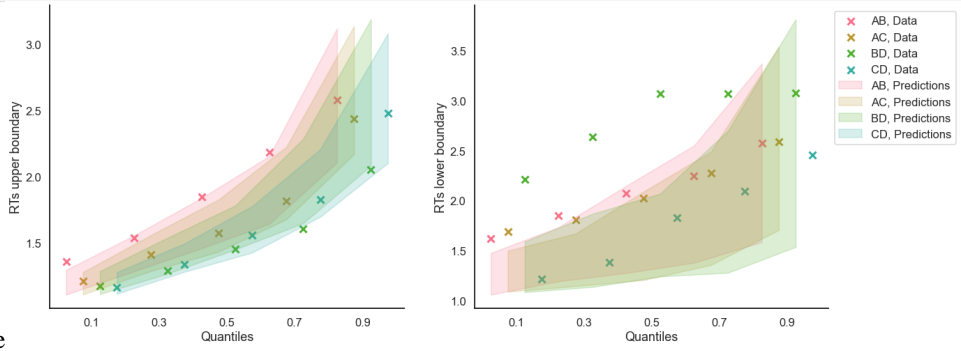
```
           97           0.793103  1.831796  1.107177         1.467779
           98           0.793103  1.898368  2.873183         1.320920
           99           0.896552  2.043208  2.130396         1.331477
           100          0.965517  1.748003  1.149147         1.250624

                    quant_30_rt_up  quant_50_rt_low  quant_50_rt_up  \
block_bins sample
1          1              1.523834         2.112334        1.723834
           2              1.281170         1.549570        1.446570
           3              1.590420         1.672020        1.839520
           4              1.328017         1.723817        1.482817
           5              1.285660         2.047260        1.446260
...                            ...              ...             ...
8          96             1.529093         1.514093        1.859093
           97             1.469279         1.586779        1.549279
           98             1.476520         1.478920        1.701920
           99             1.459777         1.346277        1.714277
           100            1.506824         1.250624        1.692624

                    quant_70_rt_low  quant_70_rt_up
block_bins sample
1          1              2.358334        2.232334
           2              1.569570        1.629170
           3              1.805620        2.031920
           4              1.995017        1.927617
           5              2.047260        1.650460
...                            ...             ...
8          96             2.040893        2.171193
           97             1.791779        2.077879
           98             1.585420        2.138520
           99             1.890677        2.075277
           100            1.250624        1.907424

[800 rows x 9 columns]
```

[19]:
```
model_fit.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins'],
                                                  n_posterior_predictives=100,
                                                  figsize=(20,8));
```



nbsphinx-code-borderwhite

[20]:
```
model_fit.plot_quantiles_grouped_posterior_predictives(
    n_posterior_predictives=100,
    grouping_var='choice_pair',
    kind='shades',
    quantiles=[.1, .3, .5, .7, .9]);
```



nbsphinx-code-borderwhite

CHAPTER

EIGHT

FIT THE DDM ON HIERARCHICAL DATA

[1]:
```
import rlssm
import pandas as pd
import os
```

8.1 Import the data

[2]:
```
data = rlssm.load_example_dataset(hierarchical_levels = 2)

data.head()
```

[2]:
```
   participant  block_label  trial_block  f_cor  f_inc  cor_option  \
0            1            1            1     43     39           2
1            1            1            2     60     50           4
2            1            1            3     44     36           4
3            1            1            4     55     55           4
4            1            1            5     52     49           4

   inc_option  times_seen        rt  accuracy
0           1           1  1.244082         0
1           3           1  1.101821         1
2           2           2  1.029923         0
3           3           2  1.368007         0
4           3           3  1.039329         1
```

8.2 Initialize the model

[3]:
```
model = rlssm.DDModel(hierarchical_levels = 2)
```

```
Using cached StanModel
```

8.3 Fit

[4]:
```
# sampling parameters
n_iter = 3000
n_warmup = 1000
n_chains = 2
n_thin = 1

# bayesian model, change default priors:
drift_priors = {'mu_mu':1, 'sd_mu':1, 'mu_sd':0, 'sd_sd':1}
threshold_priors = {'mu_mu':-1, 'sd_mu':.5, 'mu_sd':0, 'sd_sd':1}
```

[5]:
```
model_fit = model.fit(
    data,
    drift_priors=drift_priors,
    threshold_priors=threshold_priors,
    warmup = n_warmup,
    iter = n_iter,
    chains = n_chains,
    verbose = False)
```

```
Fitting the model using the priors:
drift_priors {'mu_mu': 1, 'sd_mu': 1, 'mu_sd': 0, 'sd_sd': 1}
threshold_priors {'mu_mu': -1, 'sd_mu': 0.5, 'mu_sd': 0, 'sd_sd': 1}
ndt_priors {'mu_mu': 1, 'sd_mu': 1, 'mu_sd': 0, 'sd_sd': 1}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
```

```
Checks MCMC diagnostics:
n_eff / iter looks reasonable for all parameters
0.0 of 4000 iterations ended with a divergence (0.0%)
0 of 4000 iterations saturated the maximum tree depth of 10 (0.0%)
E-BFMI indicated no pathological behavior
```

8.3.1 get Rhat

[6]:
```
model_fit.rhat.describe()
```

[6]:
```
            rhat
count  87.000000
mean    1.002063
std     0.001658
min     0.999537
25%     1.000544
50%     1.001811
75%     1.003711
max     1.005368
```

8.3.2 calculate wAIC

[7]:
```
model_fit.waic
```

[7]:
```
{'lppd': -5411.232334273516,
 'p_waic': 100.49218768903286,
 'waic': 11023.449043925099,
 'waic_se': 176.56475211992282}
```

8.4 Posteriors

[8]:
```
model_fit.samples.describe()
```

[8]:
```
             chain         draw  transf_mu_drift  transf_mu_threshold  \
count  4000.000000  4000.000000      4000.000000          4000.000000
mean      0.500000   999.500000         0.902761             1.807539
std       0.500063   577.422379         0.059822             0.054908
min       0.000000     0.000000         0.674924             1.595346
25%       0.000000   499.750000         0.863994             1.772357
50%       0.500000   999.500000         0.903058             1.808784
75%       1.000000  1499.250000         0.940805             1.842290
max       1.000000  1999.000000         1.141893             2.033432

       transf_mu_ndt  drift_sbj[1]  drift_sbj[2]  drift_sbj[3]  drift_sbj[4]  \
count    4000.000000   4000.000000   4000.000000   4000.000000   4000.000000
mean        0.741604      1.130521      0.796869      0.987316      0.722244
std         0.026017      0.092552      0.086190      0.091410      0.070307
min         0.647802      0.819075      0.466966      0.655683      0.454312
25%         0.723955      1.067871      0.738409      0.926912      0.674731
50%         0.740936      1.130679      0.795897      0.986899      0.722107
75%         0.758722      1.192259      0.855532      1.047859      0.769020
max         0.852444      1.422928      1.123298      1.392205      0.987737

       drift_sbj[5]  ...  ndt_sbj[18]  ndt_sbj[19]  ndt_sbj[20]  ndt_sbj[21]  \
count   4000.000000  ...  4000.000000  4000.000000  4000.000000  4000.000000
mean       0.788625  ...     0.730397     0.387217     0.917420     0.746415
std        0.080458  ...     0.010526     0.013373     0.012058     0.016956
min        0.505320  ...     0.676340     0.326232     0.861828     0.663290
25%        0.733996  ...     0.723725     0.378700     0.909887     0.735585
50%        0.788834  ...     0.731266     0.388043     0.918484     0.747791
75%        0.842797  ...     0.738054     0.396858     0.925692     0.758399
max        1.074931  ...     0.757969     0.422315     0.950412     0.794461

       ndt_sbj[22]  ndt_sbj[23]  ndt_sbj[24]  ndt_sbj[25]  ndt_sbj[26]  \
count  4000.000000  4000.000000  4000.000000  4000.000000  4000.000000
mean      0.742279     0.582877     0.738693     0.853545     0.465463
std       0.012844     0.015244     0.011544     0.009388     0.005858
min       0.685504     0.512557     0.690101     0.785180     0.433751
25%       0.734626     0.573140     0.731390     0.847871     0.461699
50%       0.743200     0.583825     0.739453     0.854248     0.465894
75%       0.751337     0.593646     0.746683     0.860238     0.469833
```

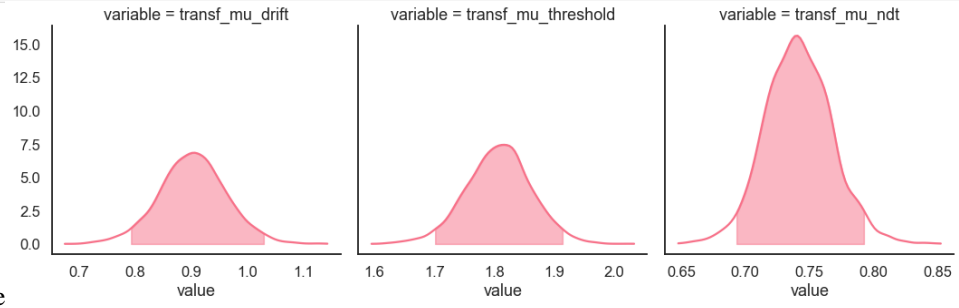
(continues on next page)

(continued from previous page)

```
max       0.777875     0.625252     0.774845     0.880258     0.479346

       ndt_sbj[27]
count  4000.000000
mean      0.863420
std       0.013867
min       0.801070
25%       0.854582
50%       0.864309
75%       0.873388
max       0.901650

[8 rows x 86 columns]
```

[9]:
```
import seaborn as sns
sns.set(context = "talk",
        style = "white",
        palette = "husl",
        rc={'figure.figsize':(15, 8)})
```

[10]:
```
model_fit.plot_posteriors(height=5, show_intervals='HDI');
```



nbsphinx-code-borderwhite

8.5 Posterior predictives

8.5.1 Ungrouped

[11]:
```
pp_summary = model_fit.get_posterior_predictives_summary(n_posterior_predictives=100)
pp_summary
```

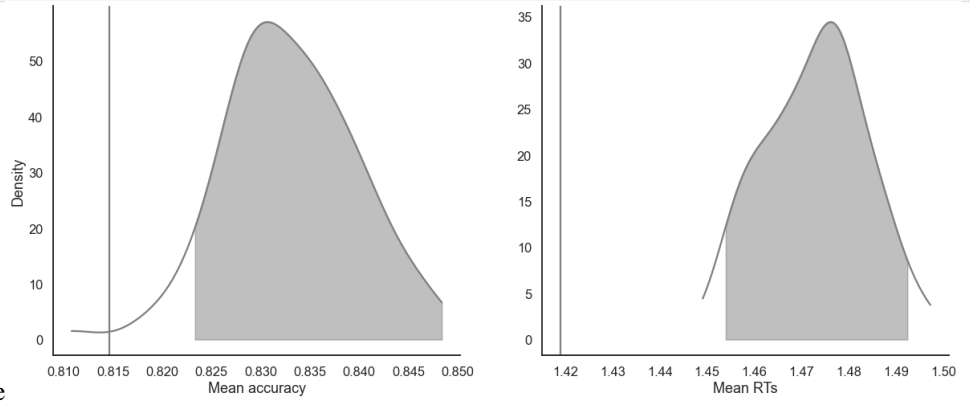
[11]:
```
        mean_accuracy   mean_rt  skewness  quant_10_rt_low  quant_30_rt_low  \
sample
1            0.833075  1.465869  1.924414         0.883730         1.093301
2            0.836170  1.463162  2.103788         0.892418         1.078721
3            0.846999  1.483644  2.208010         0.905064         1.090908
4            0.814047  1.484810  2.045138         0.923073         1.095670
5            0.837407  1.465835  2.270638         0.922109         1.093109
...               ...       ...       ...              ...              ...
96           0.829517  1.469527  2.031207         0.921734         1.093067
97           0.834004  1.482221  2.272835         0.884389         1.096907
```

(continues on next page)

(continued from previous page)

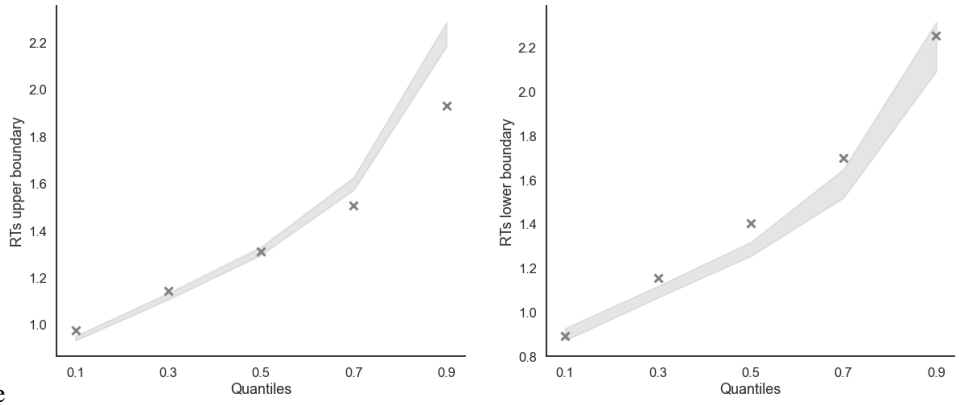
```
98          0.825959  1.478508  2.217464         0.900590         1.089122
99          0.828899  1.468675  2.238373         0.900192         1.104152
100         0.827042  1.463481  2.305245         0.882360         1.074106

        quant_50_rt_low  quant_70_rt_low  quant_90_rt_low  quant_10_rt_up  \
sample
1              1.272597         1.568961         2.198967        0.941062
2              1.299354         1.572111         2.172324        0.931466
3              1.315067         1.576960         2.248927        0.940345
4              1.295465         1.581610         2.274091        0.947493
5              1.314665         1.643438         2.282154        0.937676
...                 ...              ...              ...             ...
96             1.290120         1.542060         2.201835        0.940105
97             1.286055         1.565813         2.301758        0.940277
98             1.268247         1.549817         2.153399        0.941824
99             1.303109         1.600759         2.304936        0.932834
100            1.280296         1.583718         2.223310        0.945467

        quant_30_rt_up  quant_50_rt_up  quant_70_rt_up  quant_90_rt_up
sample
1             1.119104        1.321842        1.611800        2.195939
2             1.112196        1.298555        1.591849        2.215945
3             1.116395        1.311158        1.604061        2.271693
4             1.129157        1.314156        1.610896        2.241961
5             1.112792        1.289724        1.571149        2.203733
...                ...             ...             ...             ...
96            1.114216        1.305271        1.599589        2.227620
97            1.119118        1.309424        1.595480        2.277529
98            1.118637        1.309469        1.600631        2.277239
99            1.107752        1.296001        1.570841        2.213527
100           1.117998        1.303640        1.580593        2.195555

[100 rows x 13 columns]
```

[12]:
```
model_fit.plot_mean_posterior_predictives(n_posterior_predictives=100, figsize=(20,8),␣
↪show_intervals='HDI');
```



nbsphinx-code-borderwhite

[13]:
```
model_fit.plot_quantiles_posterior_predictives(n_posterior_predictives=100, kind='shades
↪');
```



nbsphinx-code-borderwhite

8.5.2 Grouped

[14]:
```
import numpy as np
```

[15]:
```
# Define new grouping variables, in this case, for the different choice pairs, but any␣
↪grouping var can do
data['choice_pair'] = 'AB'
data.loc[(data.cor_option == 3) & (data.inc_option == 1), 'choice_pair'] = 'AC'
data.loc[(data.cor_option == 4) & (data.inc_option == 2), 'choice_pair'] = 'BD'
data.loc[(data.cor_option == 4) & (data.inc_option == 3), 'choice_pair'] = 'CD'

data['block_bins'] = pd.cut(data.trial_block, 8, labels=np.arange(1, 9))
```

[16]:
```
model_fit.get_grouped_posterior_predictives_summary(
                grouping_vars=['block_label', 'choice_pair'],
                quantiles=[.3, .5, .7],
                n_posterior_predictives=100)
```

[16]:
```
                                  mean_accuracy   mean_rt  skewness  \
block_label choice_pair sample
1           AB          1              0.836127  1.424913  1.829896
                        2              0.810056  1.489926  3.021606
                        3              0.834264  1.522572  1.795817
                        4              0.826816  1.504265  2.000594
                        5              0.856611  1.455028  2.136628
...                                         ...       ...       ...
3           CD          96             0.822222  1.508209  1.729570
                        97             0.829630  1.469977  1.739914
                        98             0.838889  1.519894  1.752790
                        99             0.844444  1.462611  1.526917
                        100            0.838889  1.442204  2.094030

                                  quant_30_rt_low  quant_30_rt_up  \
block_label choice_pair sample
1           AB          1              1.132202        1.090361
                        2              1.085730        1.115178
                        3              1.114961        1.123546
```

(continues on next page)

(continued from previous page)

```
                        4              1.127252        1.137665
                        5              1.016709        1.092805
...                                         ...             ...
3           CD          96             1.070761        1.113506
                        97             1.114338        1.112565
                        98             1.191110        1.155950
                        99             1.011587        1.141000
                        100            1.070715        1.115111

                                  quant_50_rt_low  quant_50_rt_up  \
block_label choice_pair sample
1           AB          1              1.302380        1.287831
                        2              1.290275        1.313489
                        3              1.348158        1.338346
                        4              1.289395        1.347707
                        5              1.266724        1.297497
...                                         ...             ...
3           CD          96             1.194467        1.393401
                        97             1.333333        1.292743
                        98             1.393306        1.349036
                        99             1.217973        1.333668
                        100            1.201510        1.322007

                                  quant_70_rt_low  quant_70_rt_up
block_label choice_pair sample
1           AB          1              1.640282        1.555307
                        2              1.660278        1.574471
                        3              1.773007        1.607617
                        4              1.620425        1.639692
                        5              1.535460        1.598402
...                                         ...             ...
3           CD          96             1.542280        1.692783
                        97             1.630972        1.608350
                        98             1.715621        1.666124
                        99             1.496050        1.607953
                        100            1.432134        1.583041

[1200 rows x 9 columns]
```

[ ]:
```
model_fit.get_grouped_posterior_predictives_summary(
                grouping_vars=['block_bins'],
                quantiles=[.3, .5, .7],
                n_posterior_predictives=100)
```

[ ]:
```
model_fit.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins'],
                                                  n_posterior_predictives=100,
                                                  figsize=(20,8));
```

[ ]:
```
model_fit.plot_quantiles_grouped_posterior_predictives(n_posterior_predictives=100,
                                                       grouping_var='choice_pair',
                                                       kind='shades',
```

(continues on next page)

(continued from previous page)

```
                                                    quantiles=[.1, .3, .5, .7, .9]);
```

CHAPTER

NINE

PARAMETER RECOVERY OF THE DDM WITH STARTING POINT BIAS

[1]:
```
import rlssm
import pandas as pd
```

9.1 Simulate individual data

[2]:
```
from rlssm.random import simulate_ddm
```

[3]:
```
data = simulate_ddm(
    n_trials=400,
    gen_drift=.8,
    gen_threshold=1.3,
    gen_ndt=.23,
    gen_rel_sp=.6)
```

[4]:
```
data.describe()[['rt', 'accuracy']]
```

[4]:
```
                rt   accuracy
count  400.000000  400.00000
mean     0.583140    0.81750
std      0.296395    0.38674
min      0.257000    0.00000
25%      0.366000    1.00000
50%      0.493500    1.00000
75%      0.706500    1.00000
max      1.916000    1.00000
```

9.2 Initialize the model

[5]:
```
model = rlssm.DDModel(hierarchical_levels = 1, starting_point_bias=True)
```

```
Using cached StanModel
```

9.3 Fit

[6]:
```
# sampling parameters
n_iter = 3000
n_chains = 2
n_thin = 1

# bayesian model, change default priors:
drift_priors = {'mu':1, 'sd':3}
threshold_priors = {'mu':-1, 'sd':3}
ndt_priors = {'mu':-1, 'sd':1}
```

[7]:
```
model_fit = model.fit(
    data,
    drift_priors=drift_priors,
    threshold_priors=threshold_priors,
    ndt_priors=ndt_priors,
    thin = n_thin,
    iter = n_iter,
    chains = n_chains,
    verbose = False)
```

```
Fitting the model using the priors:
drift_priors {'mu': 1, 'sd': 3}
threshold_priors {'mu': -1, 'sd': 3}
ndt_priors {'mu': -1, 'sd': 1}
rel_sp_priors {'mu': 0, 'sd': 0.8}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
```

```
Checks MCMC diagnostics:
n_eff / iter looks reasonable for all parameters
0.0 of 3000 iterations ended with a divergence (0.0%)
0 of 3000 iterations saturated the maximum tree depth of 10 (0.0%)
E-BFMI indicated no pathological behavior
```

9.3.1 get Rhat

[8]:
```
model_fit.rhat
```

[8]:

	rhat	variable
0	1.002813	drift
1	1.000353	threshold
2	0.999921	ndt
3	1.003156	rel_sp

9.3.2 calculate wAIC

[9]:
```
model_fit.waic
```

[9]:
```
{'lppd': -122.26126045695726,
 'p_waic': 3.682425753566376,
 'waic': 251.88737242104727,
 'waic_se': 47.269086540763105}
```

9.4 Posteriors

[10]:
```
model_fit.samples.describe()
```

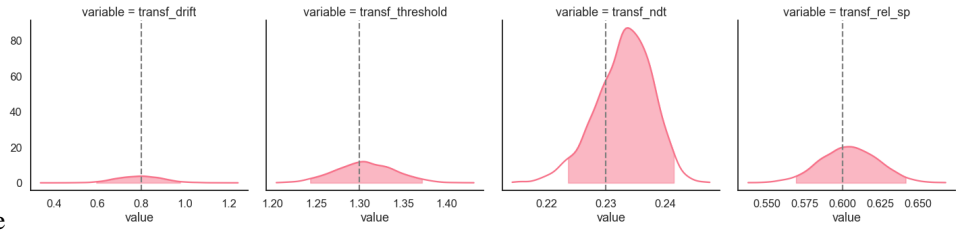
[10]:
```
             chain         draw  transf_drift  transf_threshold   transf_ndt  \
count  3000.000000  3000.000000   3000.000000       3000.000000  3000.000000
mean      0.500000   749.500000      0.792928          1.306639     0.233079
std       0.500083   433.084792      0.103006          0.033952     0.004736
min       0.000000     0.000000      0.338697          1.204520     0.214251
25%       0.000000   374.750000      0.721867          1.283811     0.230113
50%       0.500000   749.500000      0.793180          1.306182     0.233424
75%       1.000000  1124.250000      0.863523          1.329765     0.236436
max       1.000000  1499.000000      1.242643          1.431171     0.247481

       transf_rel_sp
count    3000.000000
mean        0.604580
std         0.019013
min         0.537698
25%         0.591341
50%         0.604716
75%         0.617648
max         0.668216
```

[11]:
```
import seaborn as sns
sns.set(context = "talk",
        style = "white",
        palette = "husl",
        rc={'figure.figsize':(15, 8)})
```

Here we plot the estimated posterior distributions against the generating parameters, to see whether the model parameters are recovering well:

[12]:
```
g = model_fit.plot_posteriors(height=5, show_intervals='HDI')

for i, ax in enumerate(g.axes.flatten()):
    ax.axvline(data[['drift', 'threshold', 'ndt', 'rel_sp']].mean().values[i], color=
↪'grey', linestyle='--')
```

nbsphinx-code-borderwhite



CHAPTER

TEN

PARAMETER RECOVERY OF THE HIERARCHICAL DDM WITH STARTING POINT BIAS

[1]:
```
import rlssm
import pandas as pd
```

10.1 Simulate group data

[2]:
```
from rlssm.random import simulate_hier_ddm
```

[3]:
```
data = simulate_hier_ddm(n_trials=200,
                         n_participants=10,
                         gen_mu_drift=.6, gen_sd_drift=.1,
                         gen_mu_threshold=.5, gen_sd_threshold=.1,
                         gen_mu_ndt=0, gen_sd_ndt=.01,
                         gen_mu_rel_sp=.1, gen_sd_rel_sp=.01)
```

[4]:
```
data.head()
```

[4]:

participant	trial	drift	threshold	ndt	rel_sp	rt	accuracy
1	1	0.623365	0.997041	0.692471	0.539151	0.716471	1.0
	1	0.623365	0.997041	0.692471	0.539151	0.902471	1.0
	1	0.623365	0.997041	0.692471	0.539151	0.793471	1.0
	1	0.623365	0.997041	0.692471	0.539151	0.980471	0.0
	1	0.623365	0.997041	0.692471	0.539151	1.739471	1.0

[5]:
```
data.groupby('participant').describe()[['rt', 'accuracy']]
```

[5]:

	rt						\
	count	mean	std	min	25%	50%	
participant							
1	200.0	0.951961	0.228372	0.710471	0.796221	0.888471	
2	200.0	0.977915	0.226304	0.730590	0.816340	0.919590	
3	200.0	0.999711	0.246408	0.728586	0.829836	0.935586	
4	200.0	0.926358	0.171244	0.720133	0.794883	0.892133	
5	200.0	0.885120	0.165347	0.708605	0.774355	0.831605	
6	200.0	0.939961	0.198498	0.711296	0.809296	0.900296	
7	200.0	0.909842	0.185368	0.703557	0.785057	0.844057	

(continues on next page)

(continued from previous page)

```
8            200.0  0.902757  0.165986  0.710047  0.790047  0.854547
9            200.0  0.892931  0.151139  0.711606  0.777356  0.852106
10           200.0  0.955701  0.230286  0.726306  0.810306  0.901306

                           accuracy                                    \
                75%       max    count   mean       std  min  25%  50%  75%
participant
1           1.012721  2.130471    200.0  0.680  0.467647  0.0  0.0  1.0  1.0
2           1.074590  2.046590    200.0  0.650  0.478167  0.0  0.0  1.0  1.0
3           1.111336  2.373586    200.0  0.675  0.469550  0.0  0.0  1.0  1.0
4           1.008383  1.610133    200.0  0.695  0.461563  0.0  0.0  1.0  1.0
5           0.923105  1.585605    200.0  0.625  0.485338  0.0  0.0  1.0  1.0
6           1.028546  2.424296    200.0  0.655  0.476561  0.0  0.0  1.0  1.0
7           0.984807  2.023557    200.0  0.720  0.450126  0.0  0.0  1.0  1.0
8           0.967047  1.645047    200.0  0.635  0.482638  0.0  0.0  1.0  1.0
9           0.968856  1.604606    200.0  0.735  0.442441  0.0  0.0  1.0  1.0
10          1.023556  2.345306    200.0  0.705  0.457187  0.0  0.0  1.0  1.0


            max
participant
1           1.0
2           1.0
3           1.0
4           1.0
5           1.0
6           1.0
7           1.0
8           1.0
9           1.0
10          1.0
```

10.2 Initialize the model

[6]:
```
model = rlssm.DDModel(hierarchical_levels = 2, starting_point_bias=True)
```

```
Using cached StanModel
```

10.3 Fit

[7]:
```
# sampling parameters
n_iter = 5000
n_chains = 2
n_thin = 1
```

[8]:
```
model_fit = model.fit(
    data,
```

(continues on next page)

(continued from previous page)

```
    thin = n_thin,
    iter = n_iter,
    chains = n_chains,
    verbose = False)
```

```
Fitting the model using the priors:
drift_priors {'mu_mu': 1, 'sd_mu': 5, 'mu_sd': 0, 'sd_sd': 5}
threshold_priors {'mu_mu': 1, 'sd_mu': 3, 'mu_sd': 0, 'sd_sd': 3}
ndt_priors {'mu_mu': 1, 'sd_mu': 1, 'mu_sd': 0, 'sd_sd': 1}
rel_sp_priors {'mu_mu': 0, 'sd_mu': 1, 'mu_sd': 0, 'sd_sd': 1}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
WARNING:pystan:3 of 5000 iterations ended with a divergence (0.06 %).
WARNING:pystan:Try running with adapt_delta larger than 0.8 to remove the divergences.
```

```
Checks MCMC diagnostics:
n_eff / iter looks reasonable for all parameters
3.0 of 5000 iterations ended with a divergence (0.06%)
  Try running with larger adapt_delta to remove the divergences
0 of 5000 iterations saturated the maximum tree depth of 10 (0.0%)
E-BFMI indicated no pathological behavior
```

10.3.1 get Rhat

[9]:
```
model_fit.rhat.describe()
```

[9]:
```
            rhat
count  48.000000
mean    1.000329
std     0.000791
min     0.999604
25%     0.999880
50%     1.000038
75%     1.000444
max     1.003618
```

10.3.2 calculate wAIC

[10]:
```
model_fit.waic
```

[10]:
```
{'lppd': -138.87336841598818,
 'p_waic': 23.687615971055777,
 'waic': 325.1219687740879,
 'waic_se': 92.20297977041197}
```

10.4 Posteriors

[11]:
```
model_fit.samples.describe()
```

[11]:
```
            chain         draw  transf_mu_drift  transf_mu_threshold  \
count  5000.00000  5000.000000      5000.000000          5000.000000
mean      0.50000  1249.500000         0.615952             1.009498
std       0.50005   721.759958         0.065815             0.030910
min       0.00000     0.000000         0.350688             0.860374
25%       0.00000   624.750000         0.572878             0.990568
50%       0.50000  1249.500000         0.615384             1.009284
75%       1.00000  1874.250000         0.659705             1.028144
max       1.00000  2499.000000         0.961835             1.160558

       transf_mu_ndt  transf_mu_rel_sp  drift_sbj[1]  drift_sbj[2]  \
count    5000.000000       5000.000000   5000.000000   5000.000000
mean        0.691726          0.530285      0.603799      0.576879
std         0.002199          0.012904      0.090243      0.093931
min         0.681198          0.475725      0.093595      0.157550
25%         0.690373          0.522065      0.549763      0.520853
50%         0.691617          0.530185      0.604913      0.584693
75%         0.693001          0.538358      0.660774      0.638983
max         0.704329          0.589211      0.934712      0.990759

       drift_sbj[3]  drift_sbj[4]  ...  rel_sp_sbj[1]  rel_sp_sbj[2]  \
count   5000.000000   5000.000000  ...    5000.000000    5000.000000
mean       0.599734      0.642007  ...       0.536195       0.526231
std        0.088017      0.093087  ...       0.018094       0.017883
min        0.190691      0.282644  ...       0.469905       0.460770
25%        0.544895      0.581498  ...       0.524433       0.514626
50%        0.601593      0.633981  ...       0.535846       0.526326
75%        0.656278      0.694858  ...       0.548069       0.537657
max        0.994084      1.178273  ...       0.602393       0.590122

       rel_sp_sbj[3]  rel_sp_sbj[4]  rel_sp_sbj[5]  rel_sp_sbj[6]  \
count    5000.000000    5000.000000    5000.000000    5000.000000
mean        0.522930       0.526576       0.520650       0.492087
std         0.018320       0.017435       0.017763       0.020995
min         0.455520       0.465381       0.450294       0.414656
25%         0.510905       0.515375       0.509454       0.477564
50%         0.523374       0.527026       0.521320       0.492551
75%         0.535127       0.538225       0.532493       0.506852
max         0.599665       0.592212       0.595451       0.551932

       rel_sp_sbj[7]  rel_sp_sbj[8]  rel_sp_sbj[9]  rel_sp_sbj[10]
count    5000.000000    5000.000000    5000.000000     5000.000000
mean        0.568101       0.520123       0.549857        0.539313
std         0.021740       0.017393       0.019104        0.018017
min         0.506434       0.461902       0.484085        0.466391
25%         0.552796       0.508784       0.536325        0.527094
50%         0.567758       0.520566       0.549263        0.538623
75%         0.582966       0.531734       0.562528        0.551173
max         0.638467       0.586136       0.617997        0.610974
```

(continues on next page)

(continued from previous page)

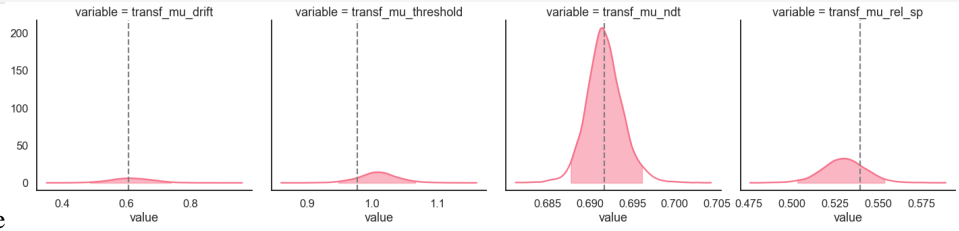
```
[8 rows x 46 columns]
```

[12]:
```
import seaborn as sns
sns.set(context = "talk",
        style = "white",
        palette = "husl",
        rc={'figure.figsize':(15, 8)})
```

Here we plot the estimated posterior distributions against the generating parameters, to see whether the model parameters are recovering well:

[13]:
```
g = model_fit.plot_posteriors(height=5, show_intervals='HDI')

for i, ax in enumerate(g.axes.flatten()):
    ax.axvline(data[['drift', 'threshold', 'ndt', 'rel_sp']].mean().values[i], color=
↪'grey', linestyle='--')
```

nbsphinx-code-borderwhite



CHAPTER

ELEVEN

FIT A RL MODEL ON INDIVIDUAL DATA

[1]:
```
import rlssm
import pandas as pd
import os
```

11.1 Import individual data

[2]:
```
# import some example data:
data = rlssm.load_example_dataset(hierarchical_levels = 1)

data.head()
```

[2]:

	participant	block_label	trial_block	f_cor	f_inc	cor_option	\
0	20	1	1	46	46	4	
1	20	1	2	60	33	4	
2	20	1	3	32	44	2	
3	20	1	4	56	40	4	
4	20	1	5	34	32	2	

	inc_option	times_seen	rt	accuracy
0	2	1	2.574407	1
1	2	2	1.952774	1
2	1	2	2.074999	0
3	2	3	2.320916	0
4	1	3	1.471107	1

11.2 Initialize the model

[3]:
```
# you can "turn on and off" different mechanisms:
model = rlssm.RLModel_2A(hierarchical_levels = 1,
                         increasing_sensitivity = False,
                         separate_learning_rates = True)
```

```
Using cached StanModel
```

[4]:
```
model.priors
```

[4]:
```
{'sensitivity_priors': {'mu': 1, 'sd': 50},
 'alpha_pos_priors': {'mu': 0, 'sd': 1},
 'alpha_neg_priors': {'mu': 0, 'sd': 1}}
```

11.3 Fit

[5]:
```
# sampling parameters
n_iter = 2000
n_chains = 2
n_thin = 1

# learning parameters
K = 4 # n options in a learning block (participants see 2 at a time)
initial_value_learning = 27.5 # intitial learning value (Q0)
```

[6]:
```
model_fit = model.fit(
    data,
    K,
    initial_value_learning,
    sensitivity_priors={'mu': 0, 'sd': 5},
    thin = n_thin,
    iter = n_iter,
    chains = n_chains,
    verbose = False)
```

```
Fitting the model using the priors:
sensitivity_priors {'mu': 0, 'sd': 5}
alpha_pos_priors {'mu': 0, 'sd': 1}
alpha_neg_priors {'mu': 0, 'sd': 1}
```

```
WARNING:pystan:n_eff / iter below 0.001 indicates that the effective sample size has␣
↪likely been overestimated
```

```
Checks MCMC diagnostics:
n_eff / iter for parameter log_p_t[1] is 0.0005136779702589292!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2] is 0.0005090424204222284!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[81] is 0.0006137657752379577!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[161] is 0.0006167516503960009!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1] is 0.0005161261210126754!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2] is 0.0005103493063252117!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[81] is 0.0009513954236082592!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter below 0.001 indicates that the effective sample size has likely been␣
↪overestimated
0.0 of 2000 iterations ended with a divergence (0.0%)
```

(continues on next page)

(continued from previous page)

```
0 of 2000 iterations saturated the maximum tree depth of 10 (0.0%)
E-BFMI indicated no pathological behavior
```

11.4 get Rhat

[7]:
```
model_fit.rhat
```

[7]:
```
       rhat     variable
0  0.999890    alpha_pos
1  1.000694    alpha_neg
2  1.000057  sensitivity
```

11.5 get wAIC

[8]:
```
model_fit.waic
```

[8]:
```
{'lppd': -76.0013341137295,
 'p_waic': 2.584147075766782,
 'waic': 157.17096237899256,
 'waic_se': 15.860333370240344}
```

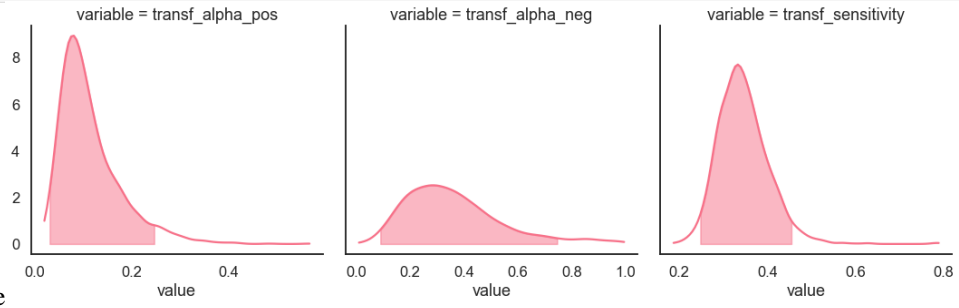
11.6 Posteriors

[9]:
```
model_fit.samples.describe()
```

[9]:
```
             chain         draw  transf_alpha_pos  transf_alpha_neg  \
count  2000.000000  2000.000000       2000.000000       2000.000000
mean      0.500000   499.500000          0.114611          0.362081
std       0.500125   288.747186          0.063520          0.180011
min       0.000000     0.000000          0.020200          0.010799
25%       0.000000   249.750000          0.070945          0.230730
50%       0.500000   499.500000          0.097523          0.329723
75%       1.000000   749.250000          0.141440          0.449876
max       1.000000   999.000000          0.567216          0.995145

       transf_sensitivity
count         2000.000000
mean             0.345694
std              0.057914
min              0.187258
25%              0.306195
50%              0.339720
75%              0.376828
max              0.789082
```

[10]:
```
import seaborn as sns
sns.set(context = "talk",
        style = "white",
        palette = "husl",
        rc={'figure.figsize':(15, 8)})
```

[11]:
```
model_fit.plot_posteriors(height=5, show_intervals="HDI", alpha_intervals=.05);
```



nbsphinx-code-borderwhite

11.7 Posterior predictives

11.7.1 Ungrouped

[12]:
```
pp = model_fit.get_posterior_predictives_df(n_posterior_predictives=1000)
pp
```

[12]:
```
variable accuracy                                    ...                   \
trial         1   2   3   4   5   6   7   8   9   10  ... 231 232 233 234
sample                                               ...
1               0   1   0   1   1   0   1   1   0   1  ...   1   1   1   1
2               1   1   0   1   1   1   1   1   1   1  ...   0   1   1   1
3               1   1   0   1   1   1   1   0   0   1  ...   0   0   1   0
4               0   0   1   1   0   1   0   1   0   1  ...   0   1   1   1
5               0   1   1   1   1   1   1   1   0   1  ...   0   0   1   1
...           ...  ..  ..  ..  ..  ..  ..  ..  ..  ..  ...  ..  ..  ..  ..
996             1   1   1   1   1   0   1   0   1   1  ...   1   1   1   1
997             1   0   1   1   1   0   1   1   0   0  ...   1   1   1   0
998             0   1   1   1   1   1   0   0   0   1  ...   0   0   1   1
999             1   1   1   1   1   1   1   0   0   0  ...   1   0   1   1
1000            0   0   1   0   1   1   1   1   0   1  ...   0   1   1   1

variable
trial    235 236 237 238 239 240
sample
1          1   1   1   0   1   1
2          1   1   1   0   1   1
3          1   0   1   1   0   1
4          1   1   1   1   1   1
5          1   1   1   1   1   1
...       ..  ..  ..  ..  ..  ..
996        1   0   0   0   0   1
```

(continues on next page)

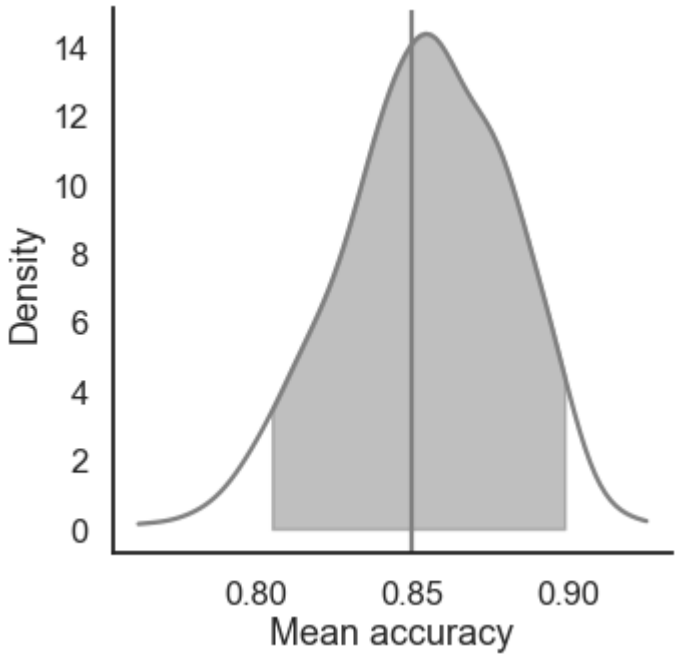
(continued from previous page)

```
997         1   1   1   1   1   1
998         1   0   1   1   1   1
999         1   0   1   1   0   1
1000        1   0   1   1   1   1

[1000 rows x 240 columns]
```

[13]:
```
pp_summary = model_fit.get_posterior_predictives_summary(n_posterior_predictives=1000)
pp_summary
```

[13]:
```
        mean_accuracy
sample
1            0.804167
2            0.804167
3            0.800000
4            0.883333
5            0.825000
...               ...
996          0.833333
997          0.908333
998          0.887500
999          0.845833
1000         0.900000

[1000 rows x 1 columns]
```

[14]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(5, 5))

model_fit.plot_mean_posterior_predictives(n_posterior_predictives=500, ax=ax, show_
↪intervals='HDI')

ax.set_ylabel('Density')
ax.set_xlabel('Mean accuracy')

sns.despine()
```



nbsphinx-code-borderwhite

11.7.2 Grouped

[15]:
```
import numpy as np
```

[16]:
```
# Define new grouping variables, in this case, for the different choice pairs, but any␣
↪grouping var can do
data['choice_pair'] = 'AB'
data.loc[(data.cor_option == 3) & (data.inc_option == 1), 'choice_pair'] = 'AC'
data.loc[(data.cor_option == 4) & (data.inc_option == 2), 'choice_pair'] = 'BD'
data.loc[(data.cor_option == 4) & (data.inc_option == 3), 'choice_pair'] = 'CD'

data['block_bins'] = pd.cut(data.trial_block, 8, labels=np.arange(1, 9))
```

[17]:
```
model_fit.get_grouped_posterior_predictives_summary(grouping_vars=['block_label', 'block_
↪bins', 'choice_pair'],
                                                    n_posterior_predictives=500)
```

[17]:

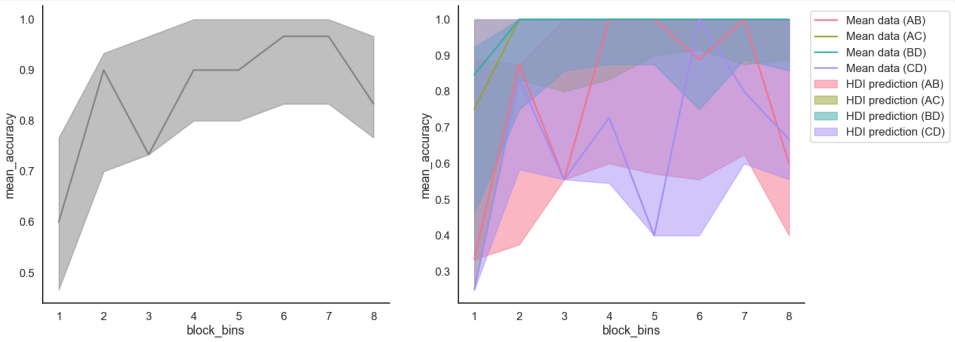
				mean_accuracy
block_label	block_bins	choice_pair	sample	
1	1	AB	1	0.800000
			2	0.200000
			3	0.600000
			4	0.800000
			5	0.800000
...				...
3	8	CD	496	1.000000
			497	0.666667
			498	0.333333
			499	0.333333

(continues on next page)

(continued from previous page)

```
                                    500          0.333333

[46000 rows x 1 columns]
```

[18]:
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(20,8))

model_fit.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins'], n_
↪posterior_predictives=500, ax=axes[0])

model_fit.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins', 'choice_
↪pair'], n_posterior_predictives=500, ax=axes[1])

sns.despine()
```



nbsphinx-code-borderwhite

CHAPTER

TWELVE

FIT A RL MODEL ON HIERARCHICAL DATA

[1]:
```
import rlssm
import pandas as pd
import os
```

12.1 Import the data

[2]:
```
# import some example data:
data = rlssm.load_example_dataset(hierarchical_levels = 2)

data.head()
```

[2]:

	participant	block_label	trial_block	f_cor	f_inc	cor_option	\
0	1	1	1	43	39	2	
1	1	1	2	60	50	4	
2	1	1	3	44	36	4	
3	1	1	4	55	55	4	
4	1	1	5	52	49	4	

	inc_option	times_seen	rt	accuracy
0	1	1	1.244082	0
1	3	1	1.101821	1
2	2	2	1.029923	0
3	3	2	1.368007	0
4	3	3	1.039329	1

12.2 Initialize the model

[3]:
```
# you can "turn on and off" different mechanisms:
model = rlssm.RLModel_2A(hierarchical_levels = 2,
                         increasing_sensitivity = False,
                         separate_learning_rates = True)
```

```
Using cached StanModel
```

12.3 Fit

[4]:
```
# sampling parameters
n_iter = 3000
n_warmup = 1000
n_chains = 2

# learning parameters
K = 4 # n options in a learning block (participants see 2 at a time)
initial_value_learning = 27.5 # intitial learning value (Q0)
```

[5]:
```
model_fit = model.fit(
    data,
    K,
    initial_value_learning,
    warmup = n_warmup,
    iter = n_iter,
    chains = n_chains)
```

```
Fitting the model using the priors:
sensitivity_priors {'mu_mu': 1, 'sd_mu': 30, 'mu_sd': 0, 'sd_sd': 30}
alpha_pos_priors {'mu_mu': 0, 'sd_mu': 1, 'mu_sd': 0, 'sd_sd': 0.1}
alpha_neg_priors {'mu_mu': 0, 'sd_mu': 1, 'mu_sd': 0, 'sd_sd': 0.1}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
```

```
Checks MCMC diagnostics:
n_eff / iter for parameter log_p_t[1] is 0.0002799233707527157!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2] is 0.0002799233707527157!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[81] is 0.0002819023696361508!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[161] is 0.0002843338482336077 36!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[162] is 0.0002843338482336077 36!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[241] is 0.0002933653734114046!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[242] is 0.0002933653734114046!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[320] is 0.0002928140782404197!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[400] is 0.0002889810526741195 7!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[480] is 0.0002790395611894542!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[482] is 0.0002790395611894542!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[560] is 0.0002871753452678353 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[561] is 0.0002871753452678353 5!
```

(continues on next page)

(continued from previous page)

```
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[640] is 0.0002880099310742576!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[720] is 0.00025765706943268 76!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[723] is 0.00025765706943268 76!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[800] is 0.00025343033893801493!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[801] is 0.00025343033893801493!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[880] is 0.00025399162211278 28!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[881] is 0.00025399162211278 28!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[959] is 0.00025466693948108 81!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1039] is 0.00025222570276379 23!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1041] is 0.00025222570276379 23!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1119] is 0.00025264916607589 764!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1199] is 0.00026321879484829 39!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1279] is 0.00028573669736374 69!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1281] is 0.00028573669736374 69!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1359] is 0.00028678470684977 82!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1439] is 0.00026986409994502706!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1519] is 0.00027615916806837725!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1520] is 0.00027615916806837725!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1599] is 0.00027570650884196 81!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1679] is 0.00025656750207598834!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1681] is 0.00025656750207598834!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1682] is 0.00025787044956306 75!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1759] is 0.00025730907447247 67!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1839] is 0.00026016390719811827!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1841] is 0.00026016390719811827!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1919] is 0.00026006908560515305!
```

(continues on next page)

(continued from previous page)

```
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[1998] is 0.0002538633169948887!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2078] is 0.0002546920004679544!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2158] is 0.00026447678259718045!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2237] is 0.00028661549004448225!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2317] is 0.00029329505695239484!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2318] is 0.0002932950569523948!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2397] is 0.00025230405014892165!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2477] is 0.0002581993475683 08!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2556] is 0.00025732370268245514!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2557] is 0.0002573237026824551 4!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2636] is 0.00025649225036419247!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2637] is 0.0002564922503641924 7!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2714] is 0.00028922441288894 84!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2715] is 0.0002892244128889484!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2794] is 0.000291084360326 35!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2796] is 0.00029108436032635!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2874] is 0.00025163474046227075!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2876] is 0.0002516347404622707 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[2954] is 0.00025423860534105853!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3034] is 0.00025463877044963834!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3113] is 0.00027726965694474894!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3192] is 0.00026046147813494 28!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3193] is 0.0002604614781349428!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3272] is 0.00026039040814173 31!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3352] is 0.00025397962075149 04!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3431] is 0.00028289585253427047!
```

(continues on next page)

(continued from previous page)

```
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3511] is 0.0002793804305194759!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3591] is 0.0002943408635524366!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3671] is 0.00030998559877916523!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3751] is 0.0003260944054773388 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3754] is 0.0003260944054773388 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3831] is 0.0002715928044771485 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3833] is 0.0002715928044771485 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3911] is 0.0002883905004669475!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3912] is 0.0002883905004669475!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[3990] is 0.0002950602408930058 3!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4070] is 0.0002523347258812193!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4148] is 0.0002534585361800773!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4149] is 0.0002534585361800773!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4228] is 0.0002533718970521620 6!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4308] is 0.0002566227505211013!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4309] is 0.0002566227505211013!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4388] is 0.0002785959197108895 6!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4389] is 0.0002785959197108895 6!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4467] is 0.0002851160588422376 4!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4469] is 0.0002851160588422376 4!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4547] is 0.0002534676283603465!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4548] is 0.0002536835417713571!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4627] is 0.0002923034262924831!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4707] is 0.0002888884958040385!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4787] is 0.0002745665559964785 4!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4866] is 0.0005843846163678089!
```

(continues on next page)

(continued from previous page)

```
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[4946] is 0.00061849097358796631!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5026] is 0.00029052292610679144!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5106] is 0.00028236989629140801!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5107] is 0.00028236989629140801!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5186] is 0.00028852482411352571!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5266] is 0.00029973650879762083!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5506] is 0.00026448368461714854!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5586] is 0.00027537123443344871!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5666] is 0.00027551023324608351!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5667] is 0.00027551023324608351!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5746] is 0.00028881669499148977!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5825] is 0.00026916320961329964!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5826] is 0.00026916320961329964!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5905] is 0.00028162259159749027!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[5985] is 0.00025124528470199231!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6065] is 0.00025333026676788491!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6066] is 0.00025333026676788491!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6145] is 0.00025347081830268261!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6225] is 0.00025579461334213591!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6226] is 0.00025579461334213591!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6305] is 0.00025491131163259851!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6306] is 0.00025491131163259851!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6385] is 0.00025705477454476831!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_p_t[6386] is 0.00025705477454476831!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1] is 0.00028954626014017541!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2] is 0.00028309417836931125!
```

(continues on next page)

(continued from previous page)

```
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[81] is 0.000850863969263963!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[161] is 0.00029897267157371867!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[241] is 0.00035856856817346 63!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[242] is 0.00030580313220260 83!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[320] is 0.00058837906213720 06!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[400] is 0.00086138295937411 61!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[480] is 0.00028519140669647957!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[482] is 0.00028519140669647957!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[560] is 0.00029284292241305525!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[561] is 0.00029284292241305525!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[640] is 0.00030244697624787 03!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[720] is 0.00025941825937871005!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[723] is 0.00025941825937871005!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[800] is 0.00025387584369178503!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[801] is 0.00025387584369178503!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[880] is 0.00025447692499797 55!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[881] is 0.00025470601484466 16!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[959] is 0.00025573409421526 61!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1039] is 0.00025302801201130534!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1041] is 0.00025267876592158 34!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1119] is 0.00025316527735717 12!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1199] is 0.00026625950701907904!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1279] is 0.00078368977416244 83!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1281] is 0.00030501559721526 26!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1359] is 0.00030449284919127607!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1439] is 0.00027252032585876 28!
```

(continues on next page)

(continued from previous page)

```
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1520] is 0.0002886069902727606!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1679] is 0.0002578426703585 39!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1681] is 0.0002581374179697 3383!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1682] is 0.0002598242200925619!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1759] is 0.0002595655856155212!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1839] is 0.0002624037797201 1435!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1841] is 0.0002624031109106839!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1919] is 0.0002620960926322 6424!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[1998] is 0.0002559544936961811!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2078] is 0.0002552205600586139!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2158] is 0.0002660971807973 0375!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2237] is 0.0003126885306053587!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2317] is 0.0003456679465261 6514!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2318] is 0.0003456679465261 6514!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2397] is 0.0002530465912392863!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2477] is 0.0002595076377372196!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2556] is 0.0002591993818883274!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2557] is 0.0002595933872333637!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2636] is 0.0002570696848124 7974!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2637] is 0.0002578714136522 4956!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2714] is 0.0003189332008750418!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2715] is 0.0002931052248891832!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2794] is 0.0002950417038878458!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2796] is 0.0003266333883847532!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2874] is 0.0002519591560546632!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2876] is 0.0002519591560546632!
```

(continues on next page)

(continued from previous page)

```
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[2954] is 0.0002550831587724221 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3034] is 0.0002560459911414795 4!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3113] is 0.0002801614575517462 4!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3192] is 0.0002663646936520726 3!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3272] is 0.0002656589602931239!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3352] is 0.0002544861941071475 7!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3431] is 0.0002950856015950594!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3591] is 0.0003196928626686715!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3671] is 0.0003427468988188433 6!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3751] is 0.0003751029011653509!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3754] is 0.0003751029011653509!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3831] is 0.0002783827420896341!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3833] is 0.0002741995011948103 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3911] is 0.0003161764276414001 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3912] is 0.0003161764276414001 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[3990] is 0.0003080420357715197 7!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4070] is 0.0002536193865679765!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4148] is 0.0002547248131675416 7!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4149] is 0.0002547248131675416 7!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4228] is 0.0002540357632675728!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4308] is 0.0002579540101053987!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4309] is 0.0002585273495782583 6!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4388] is 0.0002841926030560907 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4389] is 0.0002841926030560907 5!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4467] is 0.0003074900797024581!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4469] is 0.0002886856114533897 7!
```

(continues on next page)

(continued from previous page)

```
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4547] is 0.00025425243986929914!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4548] is 0.00025441068124249796!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4627] is 0.00055813286573467 97!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4707] is 0.000640233469688095!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4866] is 0.00073018996089533 68!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[4946] is 0.00074862257253040 9!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5026] is 0.0006601898278964!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5186] is 0.00030745490873915183!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5346] is 0.00045597346673577663!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5506] is 0.00026734840139891 23!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5586] is 0.00028833228677768 42!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5746] is 0.00030776131025889733!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5825] is 0.0002799511307667!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5905] is 0.00029768095390328013!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[5985] is 0.00025221730930267 84!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6065] is 0.00025472268218372 75!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6066] is 0.00025472268218372 75!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6145] is 0.00025503260257997 12!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6225] is 0.00025701382458317594!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6226] is 0.00025701382458317594!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6305] is 0.00025555814397782251!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6306] is 0.00025555814397782251!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6385] is 0.00025896680029555 81!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter for parameter log_lik[6386] is 0.00025913330121181665!
E-BFMI below 0.2 indicates you may need to reparameterize your model
n_eff / iter below 0.001 indicates that the effective sample size has likely been␣
↪overestimated
0.0 of 4000 iterations ended with a divergence (0.0%)
```

(continues on next page)

(continued from previous page)

```
0 of 4000 iterations saturated the maximum tree depth of 10 (0.0%)
E-BFMI indicated no pathological behavior
```

12.3.1 get Rhat

[6]:
```
model_fit.rhat.describe()
```

[6]:
```
            rhat
count  87.000000
mean    1.000126
std     0.000572
min     0.999518
25%     0.999702
50%     0.999932
75%     1.000346
max     1.001978
```

12.3.2 get wAIC

[7]:
```
model_fit.waic
```

[7]:
```
{'lppd': -2585.9186822018128,
 'p_waic': 49.06740318618709,
 'waic': 5269.972170776,
 'waic_se': 90.34698762773861}
```

12.4 Posteriors

[8]:
```
model_fit.samples.describe()
```

[8]:
```
             chain         draw  transf_mu_alpha_pos  transf_mu_alpha_neg  \
count  4000.000000  4000.000000          4000.000000          4000.000000
mean      0.500000   999.500000             0.059391             0.240970
std       0.500063   577.422379             0.011054             0.034860
min       0.000000     0.000000             0.029233             0.130741
25%       0.000000   499.750000             0.051704             0.216983
50%       0.500000   999.500000             0.058487             0.238924
75%       1.000000  1499.250000             0.066049             0.263020
max       1.000000  1999.000000             0.112444             0.375415

       transf_mu_sensitivity  alpha_pos_sbj[1]  alpha_pos_sbj[2]  \
count            4000.000000       4000.000000       4000.000000
mean                0.337773          0.042806          0.012892
std                 0.049276          0.021579          0.005417
min                 0.208916          0.007275          0.002957
25%                 0.303607          0.028721          0.009236
50%                 0.333174          0.037555          0.011515
75%                 0.366492          0.051156          0.015136
```

(continues on next page)

(continued from previous page)

```
max                0.593196          0.202651          0.050444

       alpha_pos_sbj[3]  alpha_pos_sbj[4]  alpha_pos_sbj[5]  ...  \
count       4000.000000       4000.000000       4000.000000  ...
mean           0.070399          0.101971          0.102206  ...
std            0.041293          0.045829          0.042308  ...
min            0.001847          0.023123          0.023433  ...
25%            0.046246          0.069185          0.072545  ...
50%            0.066096          0.093490          0.094744  ...
75%            0.091792          0.123847          0.124971  ...
max            0.455474          0.370728          0.396222  ...

       sensitivity_sbj[18]  sensitivity_sbj[19]  sensitivity_sbj[20]  \
count          4000.000000          4000.000000          4000.000000
mean              0.084429             0.312524             0.372647
std               0.054155             0.074492             0.058519
min               0.020625             0.133664             0.207507
25%               0.059548             0.258165             0.331700
50%               0.072125             0.302820             0.366789
75%               0.088098             0.354502             0.407251
max               0.720152             0.625586             0.699467

       sensitivity_sbj[21]  sensitivity_sbj[22]  sensitivity_sbj[23]  \
count          4000.000000          4000.000000          4000.000000
mean              0.767712             0.588902             0.901913
std               0.176125             0.110758             0.197530
min               0.357291             0.300730             0.443730
25%               0.651809             0.509775             0.760999
50%               0.740668             0.575615             0.876022
75%               0.849611             0.651033             1.016270
max               2.335755             1.317175             2.048247

       sensitivity_sbj[24]  sensitivity_sbj[25]  sensitivity_sbj[26]  \
count          4000.000000          4000.000000          4000.000000
mean              0.426767             0.606579             0.083444
std               0.076603             0.147913             0.020596
min               0.240316             0.218562             0.023674
25%               0.375488             0.501393             0.069679
50%               0.417441             0.592633             0.081530
75%               0.469168             0.698397             0.095357
max               0.975525             1.537778             0.264362

       sensitivity_sbj[27]
count          4000.000000
mean              0.250021
std               0.050787
min               0.138294
25%               0.216377
50%               0.243665
75%               0.274589
max               0.748396
```

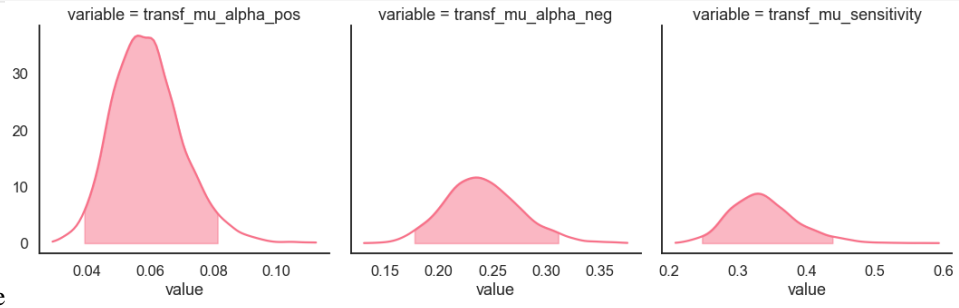
(continues on next page)

(continued from previous page)

```
[8 rows x 86 columns]
```

[9]:
```
import seaborn as sns
sns.set(context = "talk",
        style = "white",
        palette = "husl",
        rc={'figure.figsize':(15, 8)})
```

[10]:
```
model_fit.plot_posteriors(height=5, show_intervals="HDI", alpha_intervals=.05);
```



nbsphinx-code-borderwhite

12.5 Posterior predictives

12.5.1 Ungrouped

[11]:
```
pp = model_fit.get_posterior_predictives_df(n_posterior_predictives=500)
pp
```

[11]:
```
variable accuracy                                                 ...          \
trial           1    2    3    4    5    6    7    8    9    10  ... 6455 6456
sample                                                            ...
1               0    0    1    1    0    1    1    0    1    1  ...    0    0
2               1    1    1    0    0    0    1    1    1    1  ...    1    0
3               1    0    0    1    1    0    0    1    1    1  ...    1    1
4               1    0    1    0    1    1    0    0    1    1  ...    1    1
5               0    0    0    1    1    0    0    0    1    1  ...    0    1
...           ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
496             1    0    1    1    1    0    1    0    1    1  ...    1    0
497             1    0    0    1    0    1    1    0    0    0  ...    1    0
498             0    0    1    0    0    1    0    0    0    0  ...    1    1
499             0    0    1    1    0    1    1    1    1    1  ...    1    1
500             1    1    0    1    0    1    1    1    1    1  ...    1    1

variable
trial    6457 6458 6459 6460 6461 6462 6463 6464
sample
1           1    1    1    0    1    0    1    0
2           1    1    0    1    1    1    1    0
3           1    1    1    1    1    0    1    0
4           1    0    1    0    1    0    1    1
```

(continues on next page)

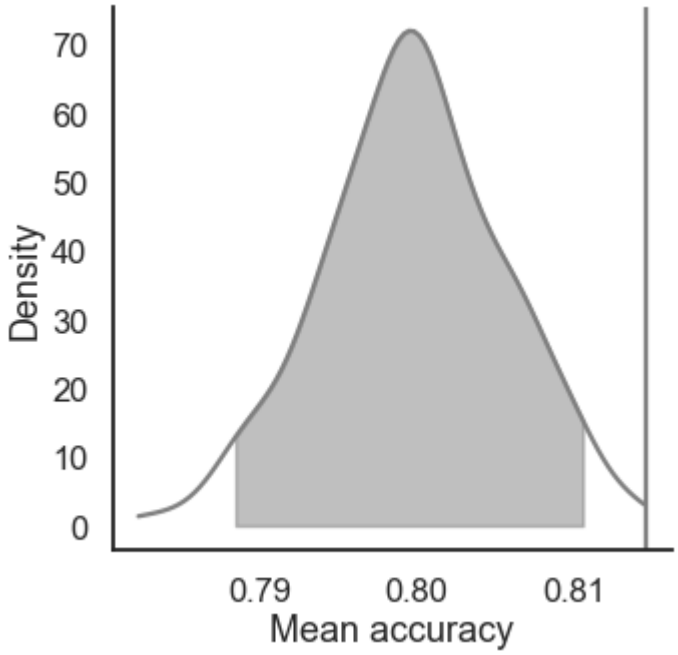
(continued from previous page)

```
5           1    1    1    1    1    1    0    1
...       ...  ...  ...  ...  ...  ...  ...  ...
496         1    0    1    0    1    1    1    1
497         1    1    1    1    1    1    1    1
498         1    1    0    1    1    1    1    1
499         1    0    1    1    0    1    1    0
500         1    1    1    1    1    0    1    1

[500 rows x 6464 columns]
```

[12]:
```
pp_summary = model_fit.get_posterior_predictives_summary(n_posterior_predictives=500)
pp_summary
```

[12]:
```
        mean_accuracy
sample
1            0.793162
2            0.798886
3            0.796101
4            0.796411
5            0.795173
...               ...
496          0.799041
497          0.802599
498          0.803837
499          0.797494
500          0.799350

[500 rows x 1 columns]
```

[13]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1, figsize=(5, 5))

model_fit.plot_mean_posterior_predictives(n_posterior_predictives=500, ax=ax, show_
↪intervals='HDI')

ax.set_ylabel('Density')
ax.set_xlabel('Mean accuracy')

sns.despine()
```



nbsphinx-code-borderwhite

12.5.2 Grouped

[14]:
```
import numpy as np
```

[15]:
```
# Define new grouping variables, in this case, for the different choice pairs, but any␣
↪grouping var can do
data['choice_pair'] = 'AB'
data.loc[(data.cor_option == 3) & (data.inc_option == 1), 'choice_pair'] = 'AC'
data.loc[(data.cor_option == 4) & (data.inc_option == 2), 'choice_pair'] = 'BD'
data.loc[(data.cor_option == 4) & (data.inc_option == 3), 'choice_pair'] = 'CD'

data['block_bins'] = pd.cut(data.trial_block, 8, labels=np.arange(1, 9))
```

[16]:
```
model_fit.get_grouped_posterior_predictives_summary(grouping_vars=['block_label', 'block_
↪bins', 'choice_pair'], n_posterior_predictives=500)
```

[16]:

				mean_accuracy
block_label	block_bins	choice_pair	sample	
1	1	AB	1	0.619048
			2	0.587302
			3	0.523810
			4	0.523810
			5	0.523810
...				...
3	8	CD	496	0.833333
			497	0.703704
			498	0.685185
			499	0.740741
			500	0.814815

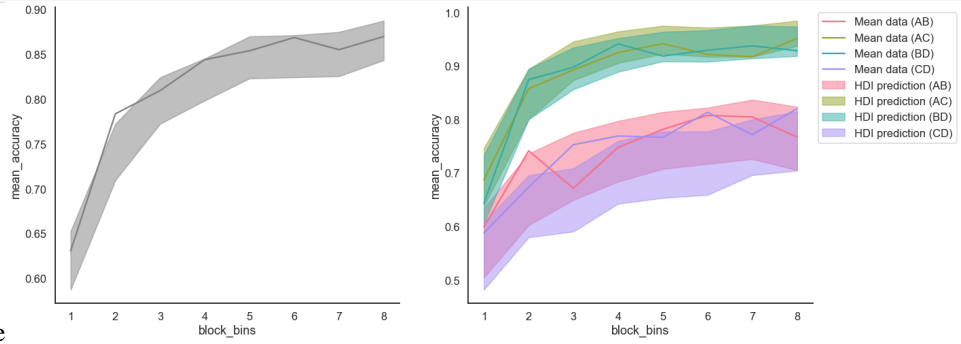
(continues on next page)

(continued from previous page)

```
[48000 rows x 1 columns]
```

[17]:
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(20,8))

model_fit.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins'], n_
↪posterior_predictives=500, ax=axes[0])

model_fit.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins', 'choice_
↪pair'], n_posterior_predictives=500, ax=axes[1])

sns.despine()
```

nbsphinx-code-borderwhite



CHAPTER

THIRTEEN

FIT THE RLDDM ON INDIVIDUAL DATA

[1]:
```
import rlssm
import pandas as pd
import os
```

13.1 Import the data

[2]:
```
# import some example data:
data = rlssm.load_example_dataset(hierarchical_levels = 1)

data.head()
```

[2]:

	participant	block_label	trial_block	f_cor	f_inc	cor_option	\
0	10	1	1	61	52	4	
1	10	1	2	54	37	4	
2	10	1	3	51	51	4	
3	10	1	4	50	35	3	
4	10	1	5	59	50	4	

	inc_option	times_seen	rt	accuracy
0	3	1	1.285418	0
1	2	1	1.577622	0
2	3	2	1.564731	0
3	1	2	1.217245	1
4	2	3	1.929781	0

13.2 Initialize the model

[3]:
```
# you can "turn on and off" different mechanisms:
model = rlssm.RLDDModel(hierarchical_levels=1,
                        separate_learning_rates=False,
                        threshold_modulation=False,
                        nonlinear_mapping=True)
```

```
Using cached StanModel
```

13.3 Fit

[4]:
```
# sampling parameters
n_iter = 3000
n_warmup = 1000
n_chains = 2

# learning parameters
K = 4 # n options in a learning block (participants see 2 at a time)
initial_value_learning = 27.5 # intitial learning value (Q0)
```

[5]:
```
model_fit = model.fit(
    data,
    K,
    initial_value_learning,
    warmup = n_warmup,
    iter = n_iter,
    chains = n_chains)
```

```
Fitting the model using the priors:
alpha_priors {'mu': 0, 'sd': 1}
drift_scaling_priors {'mu': 1, 'sd': 50}
drift_asymptote_priors {'mu': 1, 'sd': 50}
threshold_priors {'mu': 1, 'sd': 5}
ndt_priors {'mu': 1, 'sd': 1}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
WARNING:pystan:6 of 4000 iterations ended with a divergence (0.15 %).
WARNING:pystan:Try running with adapt_delta larger than 0.8 to remove the divergences.
WARNING:pystan:6 of 4000 iterations saturated the maximum tree depth of 10 (0.15 %)
WARNING:pystan:Run again with max_treedepth larger than 10 to avoid saturation
```

```
Checks MCMC diagnostics:
n_eff / iter looks reasonable for all parameters
6.0 of 4000 iterations ended with a divergence (0.15%)
  Try running with larger adapt_delta to remove the divergences
6 of 4000 iterations saturated the maximum tree depth of 10 (0.15%)
Run again with max_depth set to a larger value to avoid saturation
E-BFMI indicated no pathological behavior
```

13.3.1 get Rhat

[6]:
```
model_fit.rhat
```

[6]:

	rhat	variable
0	1.018417	alpha
1	1.037281	drift_scaling
2	1.027634	drift_asymptote
3	1.000449	threshold
4	1.001236	ndt

13.3.2 get wAIC

[7]:
```
model_fit.waic
```

[7]:
```
{'lppd': -222.10485280567747,
 'p_waic': 4.70099816274729,
 'waic': 453.6117019368495,
 'waic_se': 28.500117385818506}
```

13.4 Posteriors

[8]:
```
model_fit.samples.describe()
```

[8]:

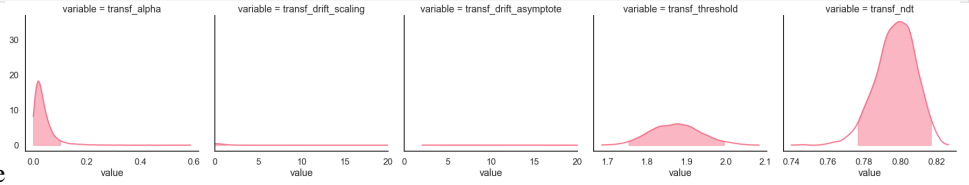
	chain	draw	transf_alpha	transf_drift_scaling	\
count	4000.000000	4000.000000	4000.000000	4000.000000	
mean	0.500000	999.500000	0.037983	0.794183	
std	0.500063	577.422379	0.039915	4.429973	
min	0.000000	0.000000	0.000057	0.002300	
25%	0.000000	499.750000	0.014890	0.013506	
50%	0.500000	999.500000	0.027752	0.029816	
75%	1.000000	1499.250000	0.048028	0.103515	
max	1.000000	1999.000000	0.587557	64.531895	

	transf_drift_asymptote	transf_threshold	transf_ndt
count	4000.000000	4000.000000	4000.000000
mean	36.160095	1.873717	0.797365
std	30.778307	0.063024	0.010966
min	2.043295	1.685766	0.739923
25%	9.554932	1.828763	0.790487
50%	28.618335	1.873455	0.798071
75%	54.034849	1.915392	0.805142
max	190.530448	2.084672	0.826664

[9]:
```
import seaborn as sns
sns.set(context = "talk",
        style = "white",
        palette = "husl",
        rc={'figure.figsize':(15, 8)})
```

[10]:
```
g = model_fit.plot_posteriors(height=5, show_intervals='HDI');

g.axes.flat[1].set_xlim(0, 20)
g.axes.flat[2].set_xlim(0, 20);
```

nbsphinx-code-borderwhite



13.5 Posterior predictives

13.5.1 Ungrouped

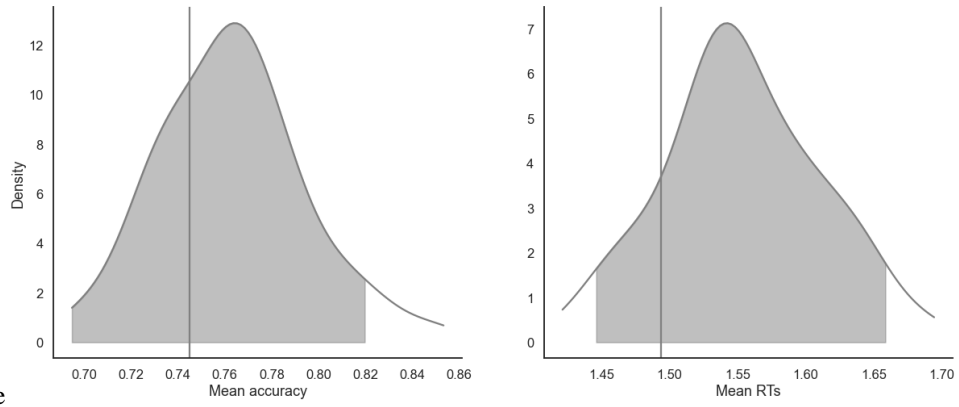
[11]:
```
pp = model_fit.get_posterior_predictives_df(n_posterior_predictives=100)
pp
```

[11]:
```
variable        rt                                                    \
trial          1         2         3         4         5         6
sample
1         1.262933  1.069933  1.784933  1.543933  1.083933  1.111933
2         1.025510  5.076510  1.705510  1.124510  1.230510  1.659510
3         1.563506  1.586506  1.463506  1.030506  0.960506  1.705506
4         1.377698  1.081698  2.731698  2.004698  1.227698  1.097698
5         1.211327  0.994327  1.420327  1.985327  0.982327  1.153327
...            ...       ...       ...       ...       ...       ...
96        1.638157  0.958157  1.110157  1.300157  1.791157  1.347157
97        1.278817  1.766817  1.079817  1.173817  1.616817  2.170817
98        1.075953  1.300953  1.090953  1.380953  1.067953  2.644953
99        1.154160  1.308160  1.598160  3.026160  2.950160  0.944160
100       2.028603  1.326603  1.547603  1.012603  1.077603  3.032603

variable                                          ... accuracy                \
trial          7         8         9         10   ...      230  231  232  233
sample                                            ...
1         2.414933  1.016933  1.438933  1.144933  ...      1.0  0.0  1.0  1.0
2         1.058510  2.201510  1.932510  1.383510  ...      1.0  1.0  1.0  1.0
3         1.394506  1.566506  1.668506  1.237506  ...      1.0  1.0  0.0  1.0
4         2.421698  2.383698  1.264698  2.216698  ...      1.0  1.0  1.0  1.0
5         1.323327  1.587327  1.402327  1.462327  ...      1.0  1.0  1.0  1.0
...            ...       ...       ...       ...  ...      ...  ...  ...  ...
96        1.563157  1.162157  1.243157  2.055157  ...      1.0  1.0  1.0  1.0
97        1.960817  1.078817  1.794817  1.288817  ...      1.0  1.0  1.0  1.0
98        1.694953  1.517953  1.402953  1.505953  ...      1.0  1.0  1.0  1.0
99        1.161160  1.392160  1.694160  1.407160  ...      1.0  1.0  1.0  1.0
100       1.243603  1.232603  1.043603  0.913603  ...      1.0  1.0  1.0  1.0

variable
trial     234  235  236  237  238  239
sample
1         1.0  1.0  1.0  1.0  1.0  0.0
2         1.0  0.0  1.0  1.0  1.0  0.0
3         1.0  1.0  1.0  1.0  0.0  1.0
4         1.0  1.0  1.0  1.0  1.0  1.0
5         1.0  1.0  1.0  1.0  1.0  1.0
...       ...  ...  ...  ...  ...  ...
96        1.0  0.0  1.0  1.0  1.0  1.0
97        1.0  1.0  1.0  1.0  1.0  1.0
98        1.0  1.0  1.0  1.0  1.0  1.0
99        1.0  0.0  1.0  1.0  1.0  1.0
100       1.0  0.0  1.0  1.0  1.0  0.0

[100 rows x 478 columns]
```

[12]:
```
pp_summary = model_fit.get_posterior_predictives_summary(n_posterior_predictives=100)
pp_summary
```

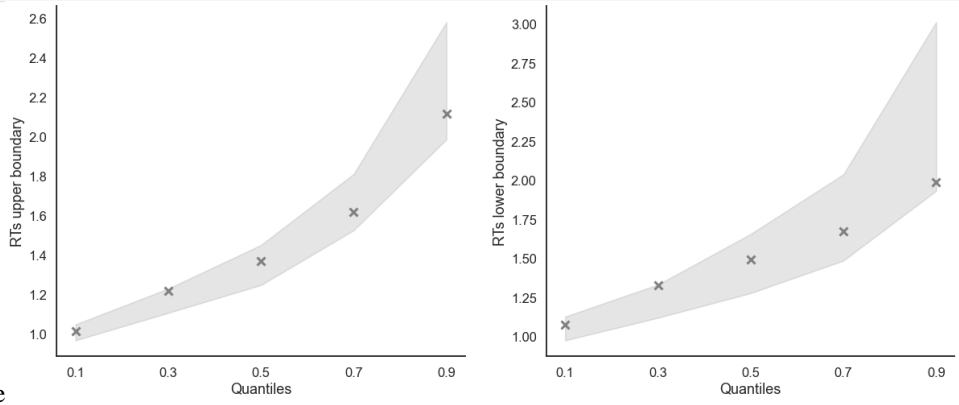
[12]:
```
        mean_accuracy   mean_rt  skewness  quant_10_rt_low  quant_30_rt_low  \
sample
1            0.736402  1.597975  2.020770         1.046733         1.180133
2            0.803347  1.568459  2.537627         1.036110         1.224110
3            0.744770  1.592808  2.887339         1.006506         1.161506
4            0.748954  1.596832  1.956484         1.062198         1.234398
5            0.765690  1.517963  2.230927         1.002327         1.154827
...               ...       ...       ...              ...              ...
96           0.686192  1.420931  1.572826         0.967357         1.108957
97           0.778243  1.513348  3.399520         1.023017         1.141617
98           0.757322  1.657665  2.967858         1.047453         1.262653
99           0.761506  1.491198  2.239910         1.002160         1.203360
100          0.769874  1.476908  2.132750         1.057803         1.188803

        quant_50_rt_low  quant_70_rt_low  quant_90_rt_low  quant_10_rt_up  \
sample
1              1.408933         1.834733         2.707733        1.018933
2              1.456510         1.908310         2.555710        1.007610
3              1.417506         1.887506         2.220506        1.027106
4              1.577698         1.925998         2.465298        1.015098
5              1.437327         1.729827         2.573327        1.000527
...                 ...              ...              ...             ...
96             1.323157         1.570757         2.142957        0.958757
97             1.316817         1.542817         1.993417        1.025817
98             1.514953         1.938653         2.723953        1.015953
99             1.428160         1.722160         2.474160        1.015160
100            1.390603         1.567003         2.074803        0.990203

        quant_30_rt_up  quant_50_rt_up  quant_70_rt_up  quant_90_rt_up
sample
1             1.226433        1.405433        1.629933        2.320933
2             1.186710        1.328510        1.644910        2.335410
3             1.195906        1.334506        1.637906        2.401806
4             1.157698        1.359698        1.651698        2.269698
5             1.139527        1.300327        1.564927        2.151527
...                ...             ...             ...             ...
96            1.081057        1.277157        1.438257        2.128157
97            1.198817        1.376317        1.621317        2.195317
98            1.177953        1.370953        1.686953        2.368953
99            1.135160        1.244660        1.503260        2.128160
100           1.108203        1.276603        1.540803        2.212203

[100 rows x 13 columns]
```

[13]:
```
model_fit.plot_mean_posterior_predictives(n_posterior_predictives=100, figsize=(20,8),␣
↪show_intervals='HDI');
```



nbsphinx-code-borderwhite

[14]:
```
model_fit.plot_quantiles_posterior_predictives(n_posterior_predictives=100, kind='shades
↪');
```



nbsphinx-code-borderwhite

13.5.2 Grouped

[15]:
```
import numpy as np
```

[16]:
```
# Define new grouping variables, in this case, for the different choice pairs, but any␣
↪grouping var can do
data['choice_pair'] = 'AB'
data.loc[(data.cor_option == 3) & (data.inc_option == 1), 'choice_pair'] = 'AC'
data.loc[(data.cor_option == 4) & (data.inc_option == 2), 'choice_pair'] = 'BD'
data.loc[(data.cor_option == 4) & (data.inc_option == 3), 'choice_pair'] = 'CD'

data['block_bins'] = pd.cut(data.trial_block, 8, labels=np.arange(1, 9))
```

[17]:
```
model_fit.get_grouped_posterior_predictives_summary(
                grouping_vars=['block_label', 'choice_pair'],
                quantiles=[.3, .5, .7],
                n_posterior_predictives=100)
```

[17]:
```
                                  mean_accuracy   mean_rt  skewness  \
block_label choice_pair sample
1           AB          1                  0.75  2.049033  1.861715
                        2                  0.65  1.672710  1.528835
```

(continues on next page)

(continued from previous page)

```
                        3                0.55  1.501656  0.745925
                        4                0.75  1.343298  1.370879
                        5                0.65  1.492577  2.332409
...                                       ...       ...       ...
3           CD          96               0.55  1.266957  1.404526
                        97               0.60  1.652267  0.660315
                        98               0.55  1.756253  0.798266
                        99               0.70  1.721210  3.783809
                        100              0.70  1.791353  2.759446

                                  quant_30_rt_low  quant_30_rt_up  \
block_label choice_pair sample
1           AB          1              1.215533        1.493133
                        2              1.153710        1.194110
                        3              1.239906        1.102506
                        4              1.110298        1.043498
                        5              1.176527        1.166127
...                                         ...             ...
3           CD          96             1.086357        1.191157
                        97             1.150917        1.331017
                        98             1.631753        1.265953
                        99             1.402160        1.234060
                        100            0.991603        1.474003

                                  quant_50_rt_low  quant_50_rt_up  \
block_label choice_pair sample
1           AB          1              1.261933        1.537933
                        2              1.548510        1.394510
                        3              1.471506        1.159506
                        4              1.464698        1.161698
                        5              1.297327        1.237327
...                                         ...             ...
3           CD          96             1.123157        1.262157
                        97             1.527317        1.603817
                        98             1.652953        1.288953
                        99             1.556660        1.314660
                        100            1.134103        1.665103

                                  quant_70_rt_low  quant_70_rt_up
block_label choice_pair sample
1           AB          1              1.385933        2.341733
                        2              1.751310        1.922710
                        3              1.657906        1.648506
                        4              1.526298        1.313698
                        5              1.693927        1.416327
...                                         ...             ...
3           CD          96             1.181957        1.365157
                        97             2.327617        1.847717
                        98             2.349353        1.884953
                        99             1.684660        1.701360
                        100            1.659103        1.834503
```

(continues on next page)

(continued from previous page)

```
[1200 rows x 9 columns]
```

[18]:
```
model_fit.get_grouped_posterior_predictives_summary(
                grouping_vars=['block_bins'],
                quantiles=[.3, .5, .7],
                n_posterior_predictives=100)
```

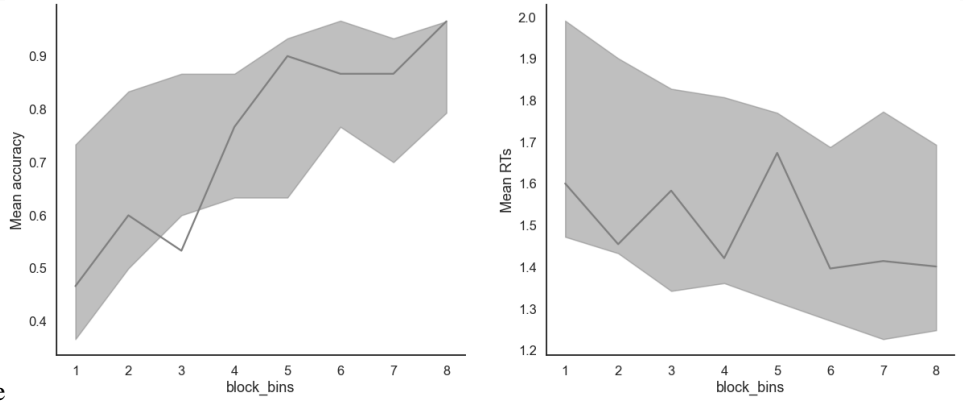
[18]:
```
                    mean_accuracy   mean_rt  skewness  quant_30_rt_low  \
block_bins sample
1          1             0.700000  1.602333  1.979790         1.158733
           2             0.566667  1.757410  1.151978         1.383910
           3             0.466667  1.813140  1.656559         1.101506
           4             0.566667  1.899765  0.471722         1.463898
           5             0.433333  1.572861  1.098442         1.180327
...                           ...       ...       ...              ...
8          96            0.862069  1.374777  2.498250         1.055157
           97            0.862069  1.655334  0.906741         1.427517
           98            0.827586  1.357953  2.034553         1.469153
           99            0.862069  1.354539  0.819861         1.521060
           100           0.965517  1.464017  2.051352         1.133603

                    quant_30_rt_up  quant_50_rt_low  quant_50_rt_up  \
block_bins sample
1          1              1.141933         1.284933        1.326933
           2              1.077710         1.912510        1.238510
           3              1.148406         1.655006        1.530006
           4              1.342298         1.783698        1.718698
           5              1.087927         1.458327        1.264327
...                            ...              ...             ...
8          96             1.124157         1.142157        1.198157
           97             1.215217         1.499317        1.459817
           98             1.014053         1.689953        1.115453
           99             1.143760         1.748660        1.229160
           100            1.094103         1.133603        1.317603

                    quant_70_rt_low  quant_70_rt_up
block_bins sample
1          1              1.535933        1.756933
           2              2.361110        1.508510
           3              1.874006        2.292106
           4              2.064898        2.397498
           5              1.775127        1.447327
...                            ...             ...
8          96             1.422057        1.367757
           97             1.592817        1.846817
           98             1.845153        1.261953
           99             1.941960        1.382560
           100            1.133603        1.514103

[800 rows x 9 columns]
```

[19]:
```
model_fit.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins'],
```

(continues on next page)

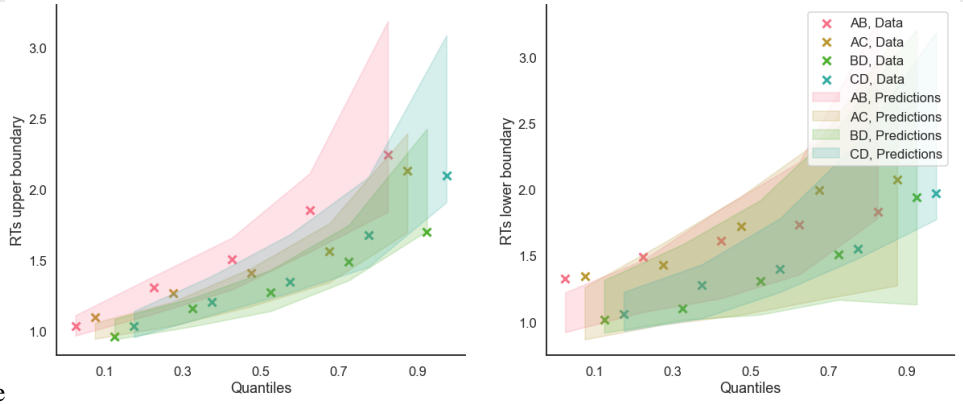
(continued from previous page)

```
                                                    n_posterior_predictives=100,
                                                    figsize=(20,8));
```



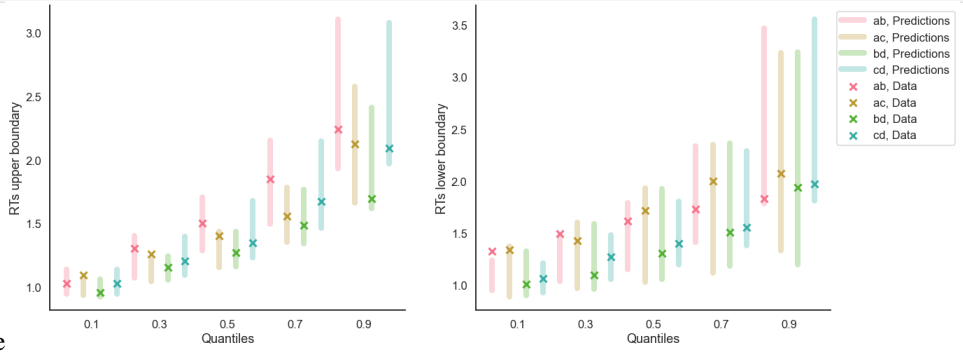
nbsphinx-code-borderwhite

[20]:
```
model_fit.plot_quantiles_grouped_posterior_predictives(n_posterior_predictives=100,
                                                       grouping_var='choice_pair',
                                                       kind='shades',
                                                       quantiles=[.1, .3, .5, .7, .9]);
```



nbsphinx-code-borderwhite

[21]:
```
model_fit.plot_quantiles_grouped_posterior_predictives(
    n_posterior_predictives=300,
    grouping_var='choice_pair',
    palette = sns.color_palette('husl'),
    intervals_kws={'alpha': .3, 'lw':8},
    hue_order=['AB', 'AC', 'BD', 'CD'],
    hue_labels=['ab', 'ac', 'bd', 'cd']);
```



nbsphinx-code-borderwhite

CHAPTER

FOURTEEN

FIT THE LBA ON INDIVIDUAL DATA

[1]:
```
import rlssm
import pandas as pd
import os
```

14.1 Import individual data

[2]:
```
# import some example data:
data = rlssm.load_example_dataset(hierarchical_levels = 1)

data.head()
```

[2]:

	participant	block_label	trial_block	f_cor	f_inc	cor_option	\
0	20	1	1	46	46	4	
1	20	1	2	60	33	4	
2	20	1	3	32	44	2	
3	20	1	4	56	40	4	
4	20	1	5	34	32	2	

	inc_option	times_seen	rt	accuracy
0	2	1	2.574407	1
1	2	2	1.952774	1
2	1	2	2.074999	0
3	2	3	2.320916	0
4	1	3	1.471107	1

14.2 Initialize the model

[3]:
```
model = rlssm.LBAModel_2A(hierarchical_levels = 1)
```

```
Using cached StanModel
```

14.3 Fit

[4]:
```
# sampling parameters
n_iter = 1000
n_chains = 2
n_thin = 5
```

[5]:
```
model_fit = model.fit(
    data,
    thin = n_thin,
    iter = n_iter,
    chains = n_chains)
```

```
Fitting the model using the priors:
drift_priors {'mu': 1, 'sd': 5}
k_priors {'mu': 1, 'sd': 1}
A_priors {'mu': 0.3, 'sd': 1}
tau_priors {'mu': 0, 'sd': 1}
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests␣
↪for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
```

```
Checks MCMC diagnostics:
n_eff / iter looks reasonable for all parameters
0.0 of 200 iterations ended with a divergence (0.0%)
0 of 200 iterations saturated the maximum tree depth of 10 (0.0%)
E-BFMI indicated no pathological behavior
```

14.3.1 Get rhat

[6]:
```
model_fit.rhat
```

[6]:
```
       rhat   variable
0  0.996595          k
1  0.996693          A
2  0.995655        tau
3  0.999796  drift_cor
4  1.002850  drift_inc
```

14.3.2 Get WAIC

[7]:
```
model_fit.waic
```

[7]:
```
{'lppd': -195.9117676732156,
 'p_waic': 3.4080101153548954,
 'waic': 398.639555577141,
 'waic_se': 35.24758887155065}
```

14.3.3 Save results

[8]:
```
model_fit.to_pickle()
```

```
Saving file as: /Users/laurafontanesi/git/rlssm/docs/notebooks/LBA_2A.pkl
```

14.4 Posteriors

[9]:
```
model_fit.samples.describe()
```

[9]:

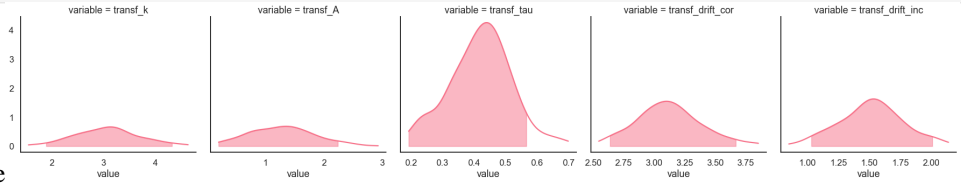
	chain	draw	transf_k	transf_A	transf_tau	\
count	200.000000	200.000000	200.000000	200.000000	200.000000	
mean	0.500000	49.500000	3.072969	1.282488	0.420315	
std	0.501255	28.938507	0.617627	0.545432	0.097345	
min	0.000000	0.000000	1.547620	0.199475	0.193221	
25%	0.000000	24.750000	2.617696	0.853162	0.356208	
50%	0.500000	49.500000	3.097387	1.279160	0.425263	
75%	1.000000	74.250000	3.441474	1.622801	0.477226	
max	1.000000	99.000000	4.636813	2.929345	0.698922	

	transf_drift_cor	transf_drift_inc
count	200.000000	200.000000
mean	3.123329	1.517988
std	0.264801	0.256313
min	2.546426	0.849490
25%	2.950180	1.360080
50%	3.109816	1.517640
75%	3.265753	1.679263
max	3.853958	2.140647

[10]:
```
import seaborn as sns
sns.set(context = "talk",
        style = "white",
        palette = "husl",
        rc={'figure.figsize':(15, 8)})
```

[11]:
```
model_fit.plot_posteriors(height=5, show_intervals='HDI');
```

nbsphinx-code-borderwhite



14.5 Posterior predictives

14.5.1 Ungrouped

[12]:
```
pp = model_fit.get_posterior_predictives_df(n_posterior_predictives=100)
pp
```

[12]:
```
variable        rt                                                    \
trial            1         2         3         4         5         6
sample
1         2.246846  1.537963  1.249976  2.667635  1.394088  1.185661
2         1.474279  1.317326  1.281292  1.776970  1.338612  1.595272
3         1.769881  1.248670  2.000013  2.477428  1.417273  1.537488
4         1.218720  1.806578  1.491374  1.556123  1.754127  2.173220
5         1.505410  1.545240  1.263004  1.981529  1.742220  1.447841
...            ...       ...       ...       ...       ...       ...
96        2.114076  1.682352  1.292053  1.290883  1.631570  2.536921
97        1.515118  1.732291  7.900185  1.992388  1.701881  1.195694
98        1.934512  1.704758  3.664648  2.025907  2.636455  1.388794
99        1.485397  1.458125  1.561393  1.374246  1.817840  1.899995
100       1.858069  1.869365  1.464518  1.568417  2.022131  1.332558

variable                                          ... accuracy                \
trial            7         8         9         10   ...      231  232  233  234
sample                                              ...
1         1.775463  1.299116  2.344744  1.585738  ...      1.0  1.0  1.0  1.0
2         1.423343  1.411392  1.368143  1.718686  ...      1.0  0.0  1.0  1.0
3         1.298886  2.047417  1.673906  1.490852  ...      1.0  1.0  1.0  1.0
4         1.786404  1.423031  1.509262  1.651425  ...      1.0  1.0  1.0  1.0
5         2.140920  1.298421  1.297520  1.534183  ...      1.0  1.0  1.0  0.0
...            ...       ...       ...       ...  ...      ...  ...  ...  ...
96        1.475092  1.839493  1.549875  1.748721  ...      1.0  1.0  1.0  1.0
97        1.765124  2.120031  1.681817  1.935664  ...      1.0  0.0  1.0  1.0
98        1.289439  1.364305  1.641123  1.961389  ...      1.0  1.0  1.0  1.0
99        2.221972  2.021029  1.958976  1.611125  ...      1.0  1.0  1.0  1.0
100       1.296797  2.512003  1.955222  3.026658  ...      0.0  1.0  1.0  1.0

variable
trial     235  236  237  238  239  240
sample
1         0.0  1.0  1.0  1.0  1.0  1.0
2         0.0  0.0  1.0  1.0  1.0  1.0
3         0.0  1.0  1.0  1.0  1.0  1.0
4         1.0  1.0  1.0  1.0  1.0  1.0
5         0.0  1.0  1.0  1.0  1.0  1.0
...       ...  ...  ...  ...  ...  ...
96        1.0  1.0  1.0  1.0  1.0  1.0
97        1.0  1.0  1.0  1.0  1.0  1.0
98        1.0  1.0  1.0  1.0  1.0  0.0
99        1.0  1.0  1.0  0.0  1.0  0.0
100       1.0  1.0  1.0  1.0  1.0  0.0

[100 rows x 480 columns]
```

[13]:
```
pp_summary = model_fit.get_posterior_predictives_summary(n_posterior_predictives=100)
pp_summary
```

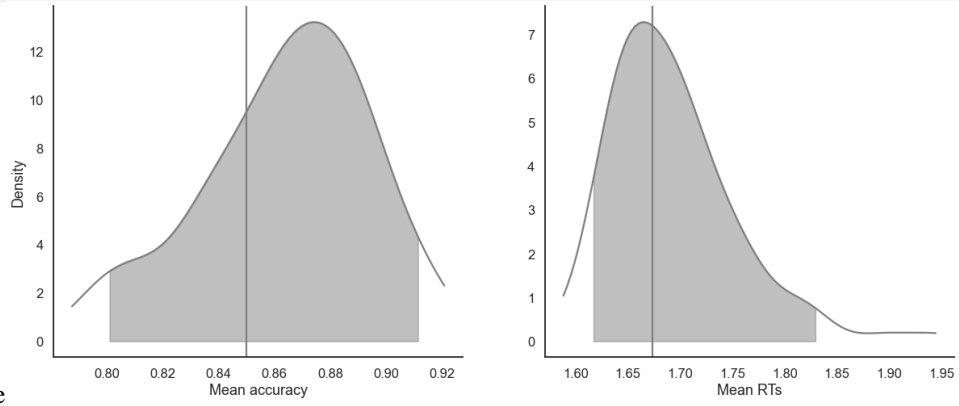
[13]:
```
        mean_accuracy   mean_rt  skewness  quant_10_rt_incorrect  \
sample
1            0.854167  1.664804  2.165810               1.506785
2            0.862500  1.682668  3.158625               1.368550
3            0.850000  1.712393  1.725298               1.613360
4            0.887500  1.619813  2.537257               1.331044
5            0.845833  1.768796  2.713919               1.382418
...               ...       ...       ...                    ...
96           0.858333  1.679296  2.592582               1.538260
97           0.887500  1.705270  8.036846               1.314066
98           0.891667  1.674279  2.983261               1.416547
99           0.812500  1.740113  3.226386               1.337939
100          0.904167  1.803922  4.509639               1.383537

        quant_30_rt_incorrect  quant_50_rt_incorrect  quant_70_rt_incorrect  \
sample
1                    1.669890               1.752361               2.038205
2                    1.540713               1.674629               2.156115
3                    1.732978               1.990309               2.124092
4                    1.556282               1.776372               2.082097
5                    1.583290               1.743878               2.181400
...                       ...                    ...                    ...
96                   1.704401               1.905312               2.145374
97                   1.598821               1.771878               2.127968
98                   1.601738               1.762865               1.999681
99                   1.570886               1.752819               2.034753
100                  1.553577               1.736791               1.890371

        quant_90_rt_incorrect  quant_10_rt_correct  quant_30_rt_correct  \
sample
1                    2.385568             1.221533             1.359931
2                    2.884746             1.240682             1.375500
3                    2.672556             1.243860             1.403816
4                    2.355689             1.149061             1.302143
5                    2.802916             1.258256             1.455523
...                       ...                  ...                  ...
96                   2.677621             1.191991             1.364760
97                   2.772499             1.191808             1.345362
98                   2.596806             1.222192             1.372858
99                   2.808015             1.211335             1.442884
100                  2.585798             1.243580             1.441850

        quant_50_rt_correct  quant_70_rt_correct  quant_90_rt_correct
sample
1                  1.511616             1.739581             2.151703
2                  1.534844             1.744025             2.127136
3                  1.528488             1.743321             2.166606
4                  1.471107             1.648440             2.088338
5                  1.567858             1.793322             2.336169
...                     ...                  ...                  ...
```

(continues on next page)

(continued from previous page)

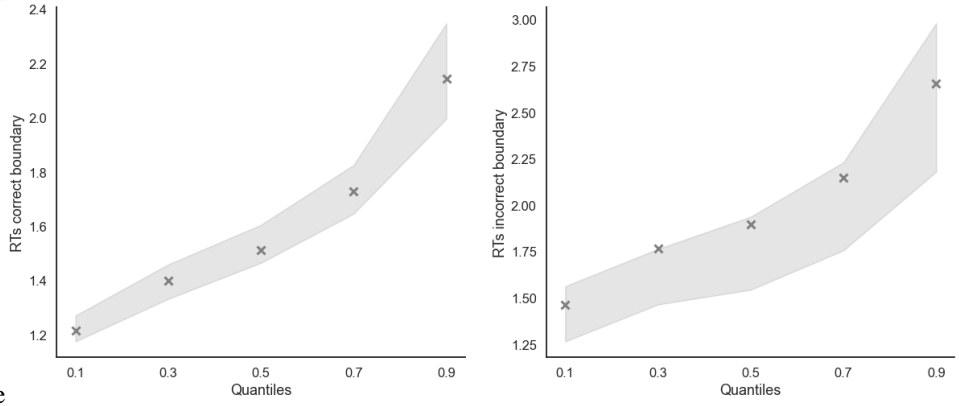
```
96                1.517171            1.705111            2.255572
97                1.502895            1.700021            2.239710
98                1.562998            1.767055            2.190829
99                1.596091            1.788202            2.252709
100               1.608690            1.858538            2.475641

[100 rows x 13 columns]
```

[14]:
```
model_fit.plot_mean_posterior_predictives(n_posterior_predictives=100, figsize=(20,8),␣
↪show_intervals='HDI');
```



nbsphinx-code-borderwhite

[15]:
```
model_fit.plot_quantiles_posterior_predictives(n_posterior_predictives=100, kind='shades
↪');
```



nbsphinx-code-borderwhite

14.5.2 Grouped

[16]:
```
import numpy as np
```

[17]:
```
# Define new grouping variables, in this case, for the different choice pairs, but any␣
↪grouping var can do
data['choice_pair'] = 'AB'
data.loc[(data.cor_option == 3) & (data.inc_option == 1), 'choice_pair'] = 'AC'
data.loc[(data.cor_option == 4) & (data.inc_option == 2), 'choice_pair'] = 'BD'
data.loc[(data.cor_option == 4) & (data.inc_option == 3), 'choice_pair'] = 'CD'
```

(continues on next page)

(continued from previous page)

```
data['block_bins'] = pd.cut(data.trial_block, 8, labels=np.arange(1, 9))
```

[18]:
```
model_fit.get_grouped_posterior_predictives_summary(
                grouping_vars=['block_label', 'choice_pair'],
                quantiles=[.3, .5, .7],
                n_posterior_predictives=100)
```

[18]:

block_label	choice_pair	sample	mean_accuracy	mean_rt	skewness	quant_30_rt_incorrect	quant_30_rt_correct	quant_50_rt_incorrect	quant_50_rt_correct	quant_70_rt_incorrect	quant_70_rt_correct
1	AB	1	0.85	1.760965	2.793220	2.273896	1.349401	2.481967	1.600919		
		2	0.75	1.791783	1.563911	1.768126	1.315160	2.557145	1.426042		
		3	0.90	1.743365	2.371058	2.337256	1.448736	2.416226	1.615792		
		4	0.85	1.801372	2.994665	1.981311	1.463829	2.114486	1.677760		
		5	0.90	1.617718	3.355427	2.051592	1.385195	2.619649	1.455759		
...			...	...	...	...	...	...	...		
3	CD	96	0.85	1.659586	-0.079339	1.661425	1.501355	1.726938	1.584971		
		97	0.85	1.524945	1.819670	1.528637	1.322557	1.713642	1.434720		
		98	0.90	1.668617	1.254209	1.800101	1.443070	1.805070	1.491801		
		99	0.75	1.462542	0.138408	1.512019	1.255587	1.536708	1.457608		
		100	0.70	1.609434	0.946274	1.566879	1.516307	1.694011	1.562540		

(continues on next page)

(continued from previous page)

```
1           AB          1                   3.183433            1.703819
                        2                   2.614723            1.619165
                        3                   2.495196            1.723226
                        4                   2.126189            1.782304
                        5                   3.187705            1.580824
...                                              ...                 ...
3           CD          96                  1.852454            1.809642
                        97                  1.860052            1.482929
                        98                  1.810039            1.667956
                        99                  1.646303            1.573202
                        100                 1.858372            1.618214

[1200 rows x 9 columns]
```

[19]:
```
model_fit.get_grouped_posterior_predictives_summary(
                grouping_vars=['block_bins'],
                quantiles=[.3, .5, .7],
                n_posterior_predictives=100)
```

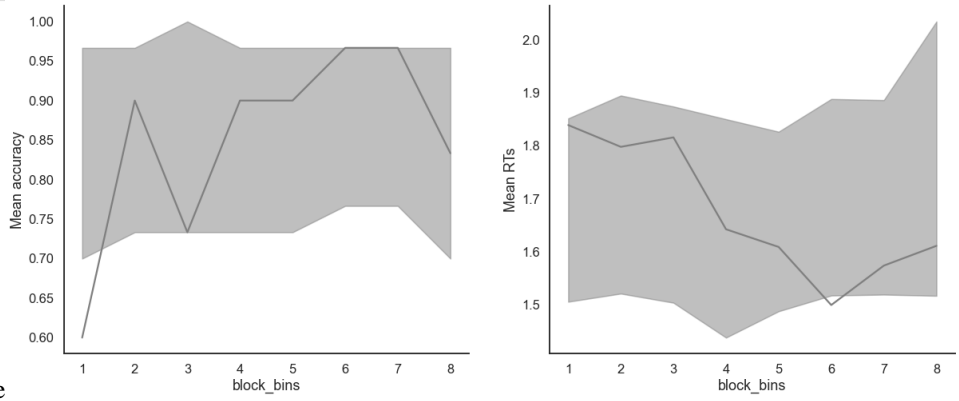
[19]:
```
                   mean_accuracy   mean_rt  skewness  quant_30_rt_incorrect  \
block_bins sample
1          1            0.866667  1.725833  1.289234               1.584221
           2            0.900000  1.752329  1.527601               2.231171
           3            0.900000  1.686033  0.924488               1.746004
           4            0.900000  1.732809  1.865226               2.113744
           5            0.800000  1.635091  2.085413               1.567397
...                          ...       ...       ...                    ...
8          96           0.966667  1.735385  4.445814               2.664596
           97           0.833333  1.693073  0.534953               1.727258
           98           0.866667  1.791996  0.811008               1.669274
           99           0.900000  1.738719  1.967172               1.483509
           100          0.866667  1.796386  1.052592               1.761800

                   quant_30_rt_correct  quant_50_rt_incorrect  \
block_bins sample
1          1                  1.460545               1.660907
           2                  1.498770               2.560241
           3                  1.332471               1.842445
           4                  1.246336               2.483196
           5                  1.324384               1.675482
...                                ...                    ...
8          96                 1.356872               2.664596
           97                 1.334572               1.769807
           98                 1.466517               1.782816
           99                 1.408075               1.496928
           100                1.419787               1.807006

                   quant_50_rt_correct  quant_70_rt_incorrect  \
block_bins sample
1          1                  1.644744               1.729153
           2                  1.612940               2.708332
           3                  1.544803               1.884541
```

(continues on next page)

(continued from previous page)

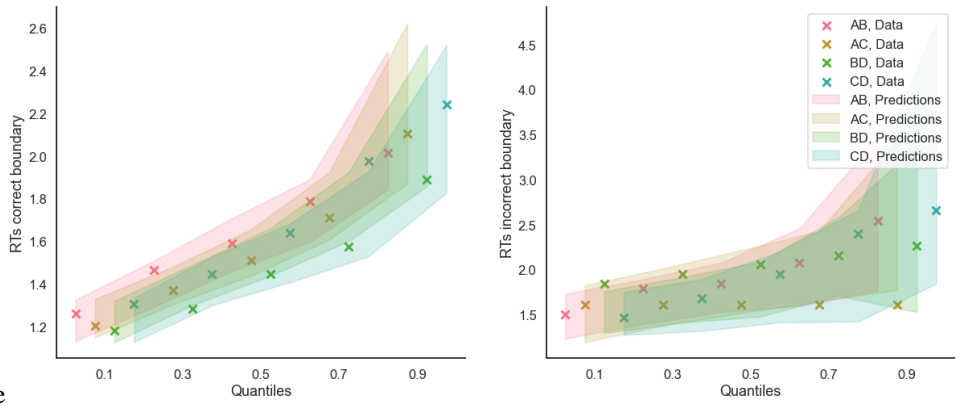
```
           4                   1.450803                2.575727
           5                   1.518778                1.730329
...                                 ...                     ...
8          96                  1.475787                2.664596
           97                  1.583482                1.853036
           98                  1.642452                1.919534
           99                  1.619821                2.271276
           100                 1.594746                1.927432

                    quant_70_rt_correct
block_bins sample
1          1                   1.791533
           2                   1.745022
           3                   1.863387
           4                   1.597772
           5                   1.790997
...                                 ...
8          96                  1.649329
           97                  1.902496
           98                  1.949012
           99                  1.783024
           100                 2.010042

[800 rows x 9 columns]
```

[20]:
```
model_fit.plot_mean_grouped_posterior_predictives(grouping_vars=['block_bins'],
                                                  n_posterior_predictives=100,
                                                  figsize=(20,8));
```



nbsphinx-code-borderwhite

[21]:
```
model_fit.plot_quantiles_grouped_posterior_predictives(
    n_posterior_predictives=100,
    grouping_var='choice_pair',
    kind='shades',
    quantiles=[.1, .3, .5, .7, .9]);
```



nbsphinx-code-borderwhite

CHAPTER

FIFTEEN

MODEL CLASSES

These classes can be used to define different available models. Curretly, 5 classes of models are implemented in *rlssm*:

1. simple reinforcement learning models: *RLModel_2A*
2. diffusion decision models: *DDModel*
3. reinforcement learning diffusion decision models: *RLDDModel*
4. race models: *RDModel_2A*, *LBAModel_2A*, *ARDModel_2A*, *ALBAModel_2A*
5. reinforcement learning race models: *RLRDModel_2A*, *RLLBAModel_2A*, *RLARDModel_2A*, *RLALBAModel_2A*

All classes have a hierarchical and non-hierarchical version, and come with additional cognitive mechanisms that can be added or excluded.

Note: At the moment, all model classes are meant for decisions between 2 alternatives.

15.1 Reinforcement learning models (for 2 alternatives)

class rlssm.**RLModel_2A**(*hierarchical_levels*, *increasing_sensitivity=False*, *separate_learning_rates=False*)

RLModel_2A allows to specify a reinforcement learning model.

When initializing the model, you should specify whether the model is hierarchical or not. Additionally, you can specify the mechanisms that you wish to include or exclude.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*, *increasing_sensitivity=False*, *separate_learning_rates=False*)

Initialize a RLModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect). However, more than two options can be presented in the same learning session.

Parameters

- **hierarchical_levels** (*int*) – Set to 1 for individual data and to 2 for grouped data.

- **increasing_sensitivity** (*bool, default False*) – By default, sensitivity is fixed throughout learning. If set to True, sensitivity increases throughout learning. In particular, it increases as a power function of the n times an option has been seen (as in Yechiam & Busemeyer, 2005).
- **separate_learning_rates** (*bool, default False*) – By default, there is only one learning rate. If set to True, separate learning rates are estimated for positive and negative prediction errors.

model_label

The label of the fully specified model.

Type
str

n_parameters_individual

The number of individual parameters of the fully specified model.

Type
int

n_parameters_trial

The number of parameters that are estimated at a trial level.

Type
int

stan_model_path

The location of the stan model code.

Type
str

compiled_model

The compiled stan model.

Type
pystan.StanModel

fit(*data*, *K*, *initial_value_learning*, *alpha_priors=None*, *sensitivity_priors=None*, *consistency_priors=None*, *scaling_priors=None*, *alpha_pos_priors=None*, *alpha_neg_priors=None*, *include_rhat=True*, *include_waic=True*, *include_last_values=True*, *pointwise_waic=False*, *print_diagnostics=True*, ***kwargs*)

Fits the specified reinforcement learning model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include (it's OK if some of them are column indexes too):

  – *trial_block*, the number of trial in a learning session. Should be integers starting from 1.

  – *f_cor*, the output from the correct option in the presented pair (the option with higher outcome on average).

  – *f_inc*, the output from the incorrect option in the presented pair (the option with lower outcome on average).

  – *cor_option*, the number identifying the correct option in the presented pair (the option with higher outcome on average).

  – *inc_option*, the number identifying the incorrect option in the presented pair (the option with lower outcome on average).

  - *block_label*, the number identifying the learning session. Should be integers starting from 1. Set to 1 in case there is only one learning session.
  - *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  - *participant*, the participant number. Should be integers starting from 1.

  If increasing_sensitivity is True, also include:

  - *times_seen*, average number of times the presented options have been seen in a learning session.

- **K** (*int*) – Number of options per learning session.
- **initial_value_learning** (*float*) – The assumed value expectation in the first learning session. The learning value in the following learning sessions is set to the average learned value in the previous learning session.
- **alpha_priors** (*dict, optional*) – Priors for the learning rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **sensitivity_priors** (*dict, optional*) – Priors for the sensitivity parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **consistency_priors** (*dict, optional*) – Priors for the consistency parameter (only meaningful if increasing_sensitivity is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **scaling_priors** (*dict, optional*) – Priors for the scaling parameter (only meaningful if increasing_sensitivity is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_pos_priors** (*dict, optional*) – Priors for the learning rate for the positive PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_neg_priors** (*dict, optional*) – Priors for the learning rate for the negative PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.

- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.RLModelResults

15.2 Diffusion decision models

class rlssm.**DDModel**(*hierarchical_levels*, *starting_point_bias=False*, *drift_variability=False*, *starting_point_variability=False*, *drift_starting_point_correlation=False*, *drift_starting_point_beta_correlation=False*, *drift_starting_point_regression=False*)

DDModel allows to specify a diffusion decision model.

When initializing the model, you should specify whether the model is hierarchical or not. Additionally, you can specify the mechanisms that you wish to include or exclude.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*, *starting_point_bias=False*, *drift_variability=False*, *starting_point_variability=False*, *drift_starting_point_correlation=False*, *drift_starting_point_beta_correlation=False*, *drift_starting_point_regression=False*)

Initialize a DDModel object.

Note: This model is restricted to two options per trial (coded as correct and incorrect).

Parameters

- **hierarchical_levels** (*int*) – Set to 1 for individual data and to 2 for grouped data.
- **starting_point_bias** (*bool, default False*) – By default, there is no starting point bias. If set to True, the starting point bias is estimated.
- **drift_variability** (*bool, default False*) – By default, there is no drift-rate variability across trials. If set to True, the standard deviation of the drift-rate across trials is estimated.
- **starting_point_variability** (*bool, default False*) – By default, there is no starting point bias variability across trials. If set to True, the standard deviation of the starting point bias across trials is estimated.
- **drift_starting_point_correlation** (*bool, default False*) – By default, the correlation between these 2 parameters is not estimated. If set to True, the 2 parameters are assumed to come from a multinormal distribution. Only relevant when drift_variability and starting_point_variability are True.
- **drift_starting_point_beta_correlation** (*bool, default False*) – If True, trial-by-trial drift-rate, rel_sp and an external variable beta are assumed to come from a multinormal distribution.

  Only relevant when drift_variability and starting_point_variability are True.
- **drift_starting_point_regression** (*bool, default False*) – If True, two regression coefficients are estimated, for trial drift and relative starting point, and an external variable beta. Only relevant when drift_variability and starting_point_variability are True.

`model_label`

The label of the fully specified model.

Type
str

`n_parameters_individual`

The number of individual parameters of the fully specified model.

Type
int

`n_parameters_trial`

The number of parameters that are estimated at a trial level.

Type
int

`stan_model_path`

The location of the stan model code.

Type
str

`compiled_model`

The compiled stan model.

Type
pystan.StanModel

`fit`(*data*, *drift_priors=None*, *threshold_priors=None*, *ndt_priors=None*, *rel_sp_priors=None*, *starting_point=0.5*, *drift_trialmu_priors=None*, *drift_trialsd_priors=None*, *rel_sp_trialmu_priors=None*, *rel_sp_trialsd_priors=None*, *corr_matrix_prior=None*, *beta_trialmu_priors=None*, *beta_trialsd_priors=None*, *regression_coefficients_priors=None*, *include_rhat=True*, *include_waic=True*, *include_last_values=True*, *pointwise_waic=False*, *print_diagnostics=True*, ***kwargs*)

Fits the specified diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include:

  – *rt*, response times in seconds.

  – *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  – *participant*, the participant number. Should be integers starting from 1.

  When either drift_starting_point_correlation, drift_starting_point_beta_correlation, or drift_starting_point_regression are True, also include:

  – *beta*, the external variable to correlate/regress to drift and rel_sp.

- **starting_point** (*float, default .5*) – The relative starting point of the diffusion process. By default there is no bias, so the starting point is .5. Should be between 0 and 1.
- **drift_priors** (*dict, optional*) – Priors for the drift-rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.

- **threshold_priors** (*dict, optional*) – Priors for the threshold parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ndt_priors** (*dict, optional*) – Priors for the non decision time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **rel_sp_priors** (*dict, optional*) – Priors for the relative starting point parameter (only meaningful if starting_point_bias is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **drift_trialmu_priors** (*dict, optional*) – Priors for the mean drift-rate across trials (only meaningful if drift_variability is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **drift_trialsd_priors** (*dict, optional*) – Priors for the standard deviation of the drift-rate across trials (only meaningful if drift_variability is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **rel_sp_trialmu_priors** (*dict, optional*) – Priors for the standard deviation of the relative starting point across trials (only meaningful if starting_point_variability is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **rel_sp_trialsd_priors** (*dict, optional*) – Priors for the standard deviation of the relative starting point across trials (only meaningful if starting_point_variability is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **corr_matrix_prior** (*float, default to 1*) – Prior for the eta parameter of the LKJ prior of the correlation matrix (only meaningful if drift_starting_point_correlation is True).
- **beta_trialmu_priors** (*dict, optional*) – Priors for the mean beta across trials (only meaningful if drift_starting_point_beta_correlation is True). Mean and standard deviation of the prior distr.
- **beta_trialsd_priors** (*dict, optional*) – Priors for the standard deviation of the beta across trials (only meaningful if drift_starting_point_beta_correlation is True). Mean and standard deviation of the prior distr.
- **regression_coefficients_priors** (*dict, optional*) – Priors for the regression coefficients (only relevant if drift_starting_point_regression is True). Mean and standard deviation of the prior distr.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.

- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

15.3 Reinforcement learning diffusion decision models

class rlssm.**RLDDModel**(*hierarchical_levels*, *nonlinear_mapping=False*, *separate_learning_rates=False*, *threshold_modulation=False*)

RLDDModel allows to specify a combination of reinforcement learning and diffusion decision models.

When initializing the model, you should specify whether the model is hierarchical or not. Additionally, you can specify the mechanisms that you wish to include or exclude.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*, *nonlinear_mapping=False*, *separate_learning_rates=False*, *threshold_modulation=False*)

Initialize a RLDDModel object.

Note: This model is restricted to two options per trial (coded as correct and incorrect). However, more than two options can be presented in the same learning session.

Parameters

- **hierarchical_levels** (*int*) – Set to 1 for individual data and to 2 for grouped data.
- **nonlinear_mapping** (*bool, default False*) – By default, the mapping between value differences and drift-rate is linear. If set to True, a non-linear mapping function is estimated.
- **separate_learning_rates** (*bool, default False*) – By default, there is only one learning rate. If set to True, separate learning rates are estimated for positive and negative prediction errors.
- **threshold_modulation** (*bool, default False*) – By default, the threshold is independent on the presented options. If set to True, the threshold can decrease or increase depending on the average value of the presented options.

model_label

The label of the fully specified model.
Type
str

n_parameters_individual

The number of individual parameters of the fully specified model.
Type
int

`n_parameters_trial`

The number of parameters that are estimated at a trial level.

Type
int

`stan_model_path`

The location of the stan model code.

Type
str

`compiled_model`

The compiled stan model.

Type
pystan.StanModel

fit(*data*, *K*, *initial_value_learning*, *alpha_priors=None*, *drift_scaling_priors=None*, *threshold_priors=None*, *ndt_priors=None*, *drift_asymptote_priors=None*, *threshold_modulation_priors=None*, *alpha_pos_priors=None*, *alpha_neg_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified reinforcement learning diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include (it's OK if some of them are column indexes too):

  – *trial_block*, the number of trial in a learning session. Should be integers starting from 1.

  – *f_cor*, the output from the correct option in the presented pair (the option with higher outcome on average).

  – *f_inc*, the output from the incorrect option in the presented pair (the option with lower outcome on average).

  – *cor_option*, the number identifying the correct option in the presented pair (the option with higher outcome on average).

  – *inc_option*, the number identifying the incorrect option in the presented pair (the option with lower outcome on average).

  – *block_label*, the number identifying the learning session. Should be integers starting from 1. Set to 1 in case there is only one learning session.

  – *rt*, response times in seconds.

  – *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  – *participant*, the participant number. Should be integers starting from 1.

- **K** (*int*) – Number of options per learning session.
- **initial_value_learning** (*float*) – The assumed value expectation in the first learning session. The learning value in the following learning sessions is set to the average learned value in the previous learning session.
- **alpha_priors** (*dict, optional*) – Priors for the learning rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.

- **drift_scaling_priors** (*dict, optional*) – Priors for the drift scaling parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **threshold_priors** (*dict, optional*) – Priors for the threshold parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ndt_priors** (*dict, optional*) – Priors for the non decision time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **drift_asymptote_priors** (*dict, optional*) – Priors for the drift-rate asymptote (only meaningful if nonlinear_mapping is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **threshold_modulation_priors** (*dict, optional*) – Priors for the threshold coefficient (only meaningful if threshold_modulation is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_pos_priors** (*dict, optional*) – Priors for the learning rate for the positive PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_neg_priors** (*dict, optional*) – Priors for the learning rate for the negative PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

15.4 Race models (for 2 alternatives)

class rlssm.**RDModel_2A**(*hierarchical_levels*)

RDModel_2A allows to specify a race diffusion model for 2 alternatives.

When initializing the model, you should specify whether the model is hierarchical or not.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*)

Initialize a RDModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect).

Parameters

hierarchical_levels (*int*) – Set to 1 for individual data and to 2 for grouped data.

model_label

The label of the fully specified model.

Type

str

n_parameters_individual

The number of individual parameters of the fully specified model.

Type

int

n_parameters_trial

The number of parameters that are estimated at a trial level.

Type

int

stan_model_path

The location of the stan model code.

Type

str

compiled_model

The compiled stan model.

Type

pystan.StanModel

fit(*data*, *threshold_priors=None*, *ndt_priors=None*, *drift_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include:

  – *rt*, response times in seconds.

– *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

If the model is hierarchical, also include:

– *participant*, the participant number. Should be integers starting from 1.

- **starting_point** (*float, default .5*) – The relative starting point of the diffusion process. By default there is no bias, so the starting point is .5. Should be between 0 and 1.
- **threshold_priors** (*dict, optional*) – Priors for the threshold parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ndt_priors** (*dict, optional*) – Priors for the non decision time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **drift_priors** (*dict, optional*) – Priors for the drift-rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

class rlssm.**LBAModel_2A**(*hierarchical_levels*)

LBAModel_2A allows to specify a linear ballistic accumulator model for 2 alternatives.

When initializing the model, you should specify whether the model is hierarchical or not.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*)

Initialize a LBAModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect).

Parameters
hierarchical_levels (*int*) – Set to 1 for individual data and to 2 for grouped data.

`model_label`

The label of the fully specified model.

Type
str

`n_parameters_individual`

The number of individual parameters of the fully specified model.

Type
int

`n_parameters_trial`

The number of parameters that are estimated at a trial level.

Type
int

`stan_model_path`

The location of the stan model code.

Type
str

`compiled_model`

The compiled stan model.

Type
pystan.StanModel

`fit`(*data*, *k_priors=None*, *A_priors=None*, *tau_priors=None*, *drift_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include:

  – *rt*, response times in seconds.

  – *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  – *participant*, the participant number. Should be integers starting from 1.

- **starting_point** (*float, default .5*) – The relative starting point of the diffusion process. By default there is no bias, so the starting point is .5. Should be between 0 and 1.
- **k_priors** (*dict, optional*) – Priors for the k parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **A_priors** (*dict, optional*) – Priors for the A parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **tau_priors** (*dict, optional*) – Priors for the non decision time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.

- **drift_priors** (*dict, optional*) – Priors for the drift-rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

class rlssm.**ARDModel_2A**(*hierarchical_levels*)

ARDModel_2A allows to specify a advantage race diffusion model for 2 alternatives.

When initializing the model, you should specify whether the model is hierarchical or not.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*)

Initialize a ARDModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect).

Parameters
hierarchical_levels (*int*) – Set to 1 for individual data and to 2 for grouped data.

model_label

The label of the fully specified model.
Type
str

n_parameters_individual

The number of individual parameters of the fully specified model.
Type
int

n_parameters_trial

The number of parameters that are estimated at a trial level.
Type
int

stan_model_path

The location of the stan model code.

Type
str

compiled_model

The compiled stan model.

Type
pystan.StanModel

fit(*data*, *threshold_priors=None*, *ndt_priors=None*, *v0_priors=None*, *ws_priors=None*, *wd_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include:

  – *rt*, response times in seconds.

  – *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  – *participant*, the participant number. Should be integers starting from 1.

- **starting_point** (*float, default .5*) – The relative starting point of the diffusion process. By default there is no bias, so the starting point is .5. Should be between 0 and 1.
- **threshold_priors** (*dict, optional*) – Priors for the threshold parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ndt_priors** (*dict, optional*) – Priors for the non decision time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **v0_priors** (*dict, optional*) – Priors for the v0 parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ws_priors** (*dict, optional*) – Priors for the ws parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **wd_priors** (*dict, optional*) – Priors for the wd parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.

- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

class rlssm.**ALBAModel_2A**(*hierarchical_levels*)

ALBAModel_2A allows to specify a advantage linear ballistic accumulator model for 2 alternatives.

When initializing the model, you should specify whether the model is hierarchical or not.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*)

Initialize a ALBAModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect).

Parameters
hierarchical_levels (*int*) – Set to 1 for individual data and to 2 for grouped data.

model_label

The label of the fully specified model.
Type
str

n_parameters_individual

The number of individual parameters of the fully specified model.
Type
int

n_parameters_trial

The number of parameters that are estimated at a trial level.
Type
int

stan_model_path

The location of the stan model code.
Type
str

compiled_model

The compiled stan model.
Type
pystan.StanModel

fit(*data*, *k_priors=None*, *A_priors=None*, *tau_priors=None*, *v0_priors=None*, *ws_priors=None*, *wd_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include:

  – *rt*, response times in seconds.

  – *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  – *participant*, the participant number. Should be integers starting from 1.

- **starting_point** (*float, default .5*) – The relative starting point of the diffusion process. By default there is no bias, so the starting point is .5. Should be between 0 and 1.
- **k_priors** (*dict, optional*) – Priors for the k parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **A_priors** (*dict, optional*) – Priors for the A parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **tau_priors** (*dict, optional*) – Priors for the non decision time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **v0_priors** (*dict, optional*) – Priors for the v0 parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ws_priors** (*dict, optional*) – Priors for the ws parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **wd_priors** (*dict, optional*) – Priors for the wd parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns

res

Return type

rlssm.fits.DDModelResults

15.5 Reinforcement learning race models (for 2 alternatives)

class rlssm.**RLRDModel_2A**(*hierarchical_levels*, *separate_learning_rates=False*, *nonlinear_mapping=False*)

RLRDModel_2A allows to specify a combination of reinforcement learning and race diffusion decision models.

When initializing the model, you should specify whether the model is hierarchical or not. Additionally, you can specify the mechanisms that you wish to include or exclude.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*, *separate_learning_rates=False*, *nonlinear_mapping=False*)

Initialize a RLRDModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect). However, more than two options can be presented in the same learning session.

Parameters

- **hierarchical_levels** (*int*) – Set to 1 for individual data and to 2 for grouped data.
- **separate_learning_rates** (*bool, default False*) – By default, there is only one learning rate. If set to True, separate learning rates are estimated for positive and negative prediction errors.
- **nonlinear_mapping** (*bool, default False*) – By default, the mapping between value differences and drift-rate is linear. If set to True, a non-linear mapping function is estimated.

model_label

The label of the fully specified model.

Type
str

n_parameters_individual

The number of individual parameters of the fully specified model.

Type
int

n_parameters_trial

The number of parameters that are estimated at a trial level.

Type
int

stan_model_path

The location of the stan model code.

Type
str

compiled_model

The compiled stan model.

Type
pystan.StanModel

fit(*data*, *K*, *initial_value_learning*, *alpha_priors=None*, *drift_scaling_priors=None*, *threshold_priors=None*, *ndt_priors=None*, *utility_priors=None*, *alpha_pos_priors=None*, *alpha_neg_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified reinforcement learning diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include (it's OK if some of them are column indexes too):

  – *trial_block*, the number of trial in a learning session. Should be integers starting from 1.

  – *f_cor*, the output from the correct option in the presented pair (the option with higher outcome on average).

  – *f_inc*, the output from the incorrect option in the presented pair (the option with lower outcome on average).

  – *cor_option*, the number identifying the correct option in the presented pair (the option with higher outcome on average).

  – *inc_option*, the number identifying the incorrect option in the presented pair (the option with lower outcome on average).

  – *block_label*, the number identifying the learning session. Should be integers starting from 1. Set to 1 in case there is only one learning session.

  – *rt*, response times in seconds.

  – *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  – *participant*, the participant number. Should be integers starting from 1.

- **K** (*int*) – Number of options per learning session.
- **initial_value_learning** (*float*) – The assumed value expectation in the first learning session. The learning value in the following learning sessions is set to the average learned value in the previous learning session.
- **alpha_priors** (*dict, optional*) – Priors for the learning rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **drift_scaling_priors** (*dict, optional*) – Priors for the drift scaling parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **threshold_priors** (*dict, optional*) – Priors for the threshold parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ndt_priors** (*dict, optional*) – Priors for the non decision time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **utility_priors** (*dict, optional*) – Priors for the utility time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.

- **alpha_pos_priors** (*dict, optional*) – Priors for the learning rate for the positive PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_neg_priors** (*dict, optional*) – Priors for the learning rate for the negative PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

class rlssm.**RLLBAModel_2A**(*hierarchical_levels*, *separate_learning_rates=False*, *nonlinear_mapping=False*)

RLLBAModel_2A allows to specify a combination of reinforcement learning and linear ballistic accumulator models.

When initializing the model, you should specify whether the model is hierarchical or not. Additionally, you can specify the mechanisms that you wish to include or exclude.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*, *separate_learning_rates=False*, *nonlinear_mapping=False*)

Initialize a RLLBAModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect). However, more than two options can be presented in the same learning session.

Parameters

- **hierarchical_levels** (*int*) – Set to 1 for individual data and to 2 for grouped data.
- **separate_learning_rates** (*bool, default False*) – By default, there is only one learning rate. If set to True, separate learning rates are estimated for positive and negative prediction errors.
- **nonlinear_mapping** (*bool, default False*) – By default, the mapping between value differences and drift-rate is linear. If set to True, a non-linear mapping function is estimated.

`model_label`

The label of the fully specified model.

Type
str

`n_parameters_individual`

The number of individual parameters of the fully specified model.

Type
int

`n_parameters_trial`

The number of parameters that are estimated at a trial level.

Type
int

`stan_model_path`

The location of the stan model code.

Type
str

`compiled_model`

The compiled stan model.

Type
pystan.StanModel

`fit`(*data*, *K*, *initial_value_learning*, *k_priors=None*, *A_priors=None*, *tau_priors=None*, *utility_priors=None*, *alpha_priors=None*, *drift_scaling_priors=None*, *alpha_pos_priors=None*, *alpha_neg_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified reinforcement learning diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include (it's OK if some of them are column indexes too):

  - *trial_block*, the number of trial in a learning session. Should be integers starting from 1.
  - *f_cor*, the output from the correct option in the presented pair (the option with higher outcome on average).
  - *f_inc*, the output from the incorrect option in the presented pair (the option with lower outcome on average).
  - *cor_option*, the number identifying the correct option in the presented pair (the option with higher outcome on average).
  - *inc_option*, the number identifying the incorrect option in the presented pair (the option with lower outcome on average).
  - *block_label*, the number identifying the learning session. Should be integers starting from 1. Set to 1 in case there is only one learning session.
  - *rt*, response times in seconds.
  - *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  - *participant*, the participant number. Should be integers starting from 1.

- **K** (*int*) – Number of options per learning session.
- **initial_value_learning** (*float*) – The assumed value expectation in the first learning session. The learning value in the following learning sessions is set to the average learned value in the previous learning session.
- **alpha_priors** (*dict, optional*) – Priors for the learning rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **drift_scaling_priors** (*dict, optional*) – Priors for the drift scaling parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **k_priors** (*dict, optional*) – Priors for the k parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **A_priors** (*dict, optional*) – Priors for the A parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **tau_priors** (*dict, optional*) – Priors for the tau parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **utility_priors** (*dict, optional*) – Priors for the utility time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_pos_priors** (*dict, optional*) – Priors for the learning rate for the positive PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_neg_priors** (*dict, optional*) – Priors for the learning rate for the negative PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

class rlssm.**RLARDModel_2A**(*hierarchical_levels*, *separate_learning_rates=False*)

RLARDModel_2A allows to specify a combination of reinforcement learning and advantage race diffusion decision models.

When initializing the model, you should specify whether the model is hierarchical or not. Additionally, you can specify the mechanisms that you wish to include or exclude.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*, *separate_learning_rates=False*)

Initialize a RLARDModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect). However, more than two options can be presented in the same learning session.

Parameters

- **hierarchical_levels** (*int*) – Set to 1 for individual data and to 2 for grouped data.
- **separate_learning_rates** (*bool, default False*) – By default, there is only one learning rate. If set to True, separate learning rates are estimated for positive and negative prediction errors.

model_label

The label of the fully specified model.

Type
str

n_parameters_individual

The number of individual parameters of the fully specified model.

Type
int

n_parameters_trial

The number of parameters that are estimated at a trial level.

Type
int

stan_model_path

The location of the stan model code.

Type
str

compiled_model

The compiled stan model.

Type
pystan.StanModel

fit(*data*, *K*, *initial_value_learning*, *threshold_priors=None*, *ndt_priors=None*, *v0_priors=None*, *ws_priors=None*, *wd_priors=None*, *alpha_priors=None*, *alpha_pos_priors=None*, *alpha_neg_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified reinforcement learning diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include (it's OK if some of them are column indexes too):

  - *trial_block*, the number of trial in a learning session. Should be integers starting from 1.
  - *f_cor*, the output from the correct option in the presented pair (the option with higher outcome on average).
  - *f_inc*, the output from the incorrect option in the presented pair (the option with lower outcome on average).
  - *cor_option*, the number identifying the correct option in the presented pair (the option with higher outcome on average).
  - *inc_option*, the number identifying the incorrect option in the presented pair (the option with lower outcome on average).
  - *block_label*, the number identifying the learning session. Should be integers starting from 1. Set to 1 in case there is only one learning session.
  - *rt*, response times in seconds.
  - *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  - *participant*, the participant number. Should be integers starting from 1.
- **K** (*int*) – Number of options per learning session.
- **initial_value_learning** (*float*) – The assumed value expectation in the first learning session. The learning value in the following learning sessions is set to the average learned value in the previous learning session.
- **threshold_priors** (*dict, optional*) – Priors for the threshold parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_priors** (*dict, optional*) – Priors for the learning rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **v0_priors** (*dict, optional*) – Priors for the v0 parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ws_priors** (*dict, optional*) – Priors for the ws parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **wd_priors** (*dict, optional*) – Priors for the wd parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ndt_priors** (*dict, optional*) – Priors for the non decision time parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_pos_priors** (*dict, optional*) – Priors for the learning rate for the positive PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.

- **alpha_neg_priors** (*dict, optional*) – Priors for the learning rate for the negative PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

class rlssm.**RLALBAModel_2A**(*hierarchical_levels*, *separate_learning_rates=False*)

RLALBAModel_2A allows to specify a combination of reinforcement learning and advantage linear ballistic accumulator models.

When initializing the model, you should specify whether the model is hierarchical or not. Additionally, you can specify the mechanisms that you wish to include or exclude.

The underlying stan model will be compiled if no previously compiled model is found. After initializing the model, it can be fitted to a particular dataset using pystan.

__init__(*hierarchical_levels*, *separate_learning_rates=False*)

Initialize a RLALBAModel_2A object.

Note: This model is restricted to two options per trial (coded as correct and incorrect). However, more than two options can be presented in the same learning session.

Parameters

- **hierarchical_levels** (*int*) – Set to 1 for individual data and to 2 for grouped data.
- **separate_learning_rates** (*bool, default False*) – By default, there is only one learning rate. If set to True, separate learning rates are estimated for positive and negative prediction errors.

model_label

The label of the fully specified model.

Type
str

`n_parameters_individual`

The number of individual parameters of the fully specified model.

Type
int

`n_parameters_trial`

The number of parameters that are estimated at a trial level.

Type
int

`stan_model_path`

The location of the stan model code.

Type
str

`compiled_model`

The compiled stan model.

Type
pystan.StanModel

fit(*data*, *K*, *initial_value_learning*, *k_priors=None*, *A_priors=None*, *tau_priors=None*, *v0_priors=None*, *ws_priors=None*, *wd_priors=None*, *alpha_priors=None*, *alpha_pos_priors=None*, *alpha_neg_priors=None*, *include_rhat=True*, *include_waic=True*, *pointwise_waic=False*, *include_last_values=True*, *print_diagnostics=True*, ***kwargs*)

Fits the specified reinforcement learning diffusion decision model to data.

Parameters

- **data** (*DataFrame*) – A pandas DataFrame containing data observations.

  Columns should include (it's OK if some of them are column indexes too):

  – *trial_block*, the number of trial in a learning session. Should be integers starting from 1.

  – *f_cor*, the output from the correct option in the presented pair (the option with higher outcome on average).

  – *f_inc*, the output from the incorrect option in the presented pair (the option with lower outcome on average).

  – *cor_option*, the number identifying the correct option in the presented pair (the option with higher outcome on average).

  – *inc_option*, the number identifying the incorrect option in the presented pair (the option with lower outcome on average).

  – *block_label*, the number identifying the learning session. Should be integers starting from 1. Set to 1 in case there is only one learning session.

  – *rt*, response times in seconds.

  – *accuracy*, 0 if the incorrect option was chosen, 1 if the correct option was chosen.

  If the model is hierarchical, also include:

  – *participant*, the participant number. Should be integers starting from 1.

- **K** (*int*) – Number of options per learning session.
- **initial_value_learning** (*float*) – The assumed value expectation in the first learning session. The learning value in the following learning sessions is set to the average learned value in the previous learning session.

- **k_priors** (*dict, optional*) – Priors for the k parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **A_priors** (*dict, optional*) – Priors for the A parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **tau_priors** (*dict, optional*) – Priors for the tau parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_priors** (*dict, optional*) – Priors for the learning rate parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **v0_priors** (*dict, optional*) – Priors for the v0 parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **ws_priors** (*dict, optional*) – Priors for the ws parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **wd_priors** (*dict, optional*) – Priors for the wd parameter. In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_pos_priors** (*dict, optional*) – Priors for the learning rate for the positive PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **alpha_neg_priors** (*dict, optional*) – Priors for the learning rate for the negative PE (only meaningful if separate_learning_rates is True). In case it is not a hierarchical model: Mean and standard deviation of the prior distr. In case it is a hierarchical model: Means and standard deviations of the hyper priors.
- **include_rhat** (*bool, default True*) – Whether to calculate the Gelman-Rubin convergence diagnostic r hat (Gelman & Rubin, 1992).
- **include_waic** (*bool, default True*) – Whether to calculate the widely applicable information criterion (WAIC; Watanabe, 2013).
- **pointwise_waic** (*bool, default False*) – Whether to also inclue the pointwise WAIC. Only relevant if include_waic is True.
- **include_last_values** (*bool, default True*) – Whether to extract the last values for each chain.
- **print_diagnostics** (*bool, default True*) – Whether to print mcmc diagnostics after fitting. It is advised to leave it to True and always check, on top of the r hat.
- ****kwargs** – Additional arguments to pystan.StanModel.sampling().

Returns
res

Return type
rlssm.fits.DDModelResults

CHAPTER

SIXTEEN

MODELRESULTS CLASS FOR RL MODELS

There is one class to inspect model fits of **RL models (fitted on choices alone)**: *RLModelResults_2A*.

The main functions of this class are:

- Assess the model's **convergence** and **mcmc diagnostics**, to make sure that the sampling was successful. This step is crucial and should be preferably done first.
- Provide a measure of the model's **quantitative fit** to the data (i.e., the Watanabe-Akaike information criterion). This is important when comparing the quantitative fit to the data of several, competing models.
- Visualize and make interval-based (either Bayesian Credible Intervals or Higher Density Intervals) inferences on the **posterior distributions** of the model's parameters. This is important when specific hypotheses were made about the parameters' values.
- Calculate and visualize **posterior predictive distributions** of the observed data. This step is important to assess the qualitative fit of the model to the data. Qualitative fit should be assessed not only when comparing different competing models, but also when a single candidate model is fitted. Different ways of calculating posterior predictive distributions are provided, together with different plotting options. In general, emphasis is given to calculating posterior predictive distributions across conditions. This allows us to assess whether a certain behavioral pattern observed in the data (e.g., due to experimental manipulations) can also be reproduced by the model.

16.1 All models

class rlssm.fits_RL.**ModelResults**(*model_label*, *data_info*, *parameters_info*, *priors*, *rhat*, *waic*, *last_values*, *samples*, *trial_samples*)

plot_posteriors(*gridsize=100*, *clip=None*, *show_intervals='HDI'*, *alpha_intervals=0.05*, *intervals_kws=None*, ***kwargs*)

Plots posterior predictives of the model's parameters.

If the model is hierarchical, then only the group parameters are plotted. In particular, group means are plotted in the first row and group standard deviations are plotted in the second row. By default, 95 percent HDI are shown. The kernel density estimation is calculated using scipy.stats.gaussian_kde.

Parameters

- **gridsize** (*int, default to 100*) – Resolution of the kernel density estimation function, default to 100.
- **clip** (*tuple of (float, float), optional*) – Range for the kernel density estimation function. Default is min and max values of the distribution.

- **show_intervals** (*str, default to "HDI"*) – Either "HDI", "BCI", or None. HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float, default to .05*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **intervals_kws** (*dict*) – Additional arguments for *matplotlib.axes.Axes.fill_between* that shows shaded intervals. By default, they are 50 percent transparent.
- ****kwargs** – Additional parameters for seaborn.FacetGrid.

Returns
g

Return type
seaborn.FacetGrid

to_pickle(*filename=None*)

Pickle the fitted model's results object to file.

This can be used to store the model's result and read them and inspect them at a later stage, without having to refit the model.

Parameters
filename (*str, optional*) – File path where the pickled object will be stored. If not specified, is set to

16.2 Reinforcement learning models

class rlssm.fits_RL.**RLModelResults_2A**(*model_label*, *data_info*, *parameters_info*, *priors*, *rhat*, *waic*, *last_values*, *samples*, *trial_samples*)

RLModelResults allows to perform various model checks on fitted RL models.

In particular, this can be used to to visualize the estimated posterior distributions and to calculate and visualize the estimated posterior predictives distributions.

show-inheritance
inherited-members

get_grouped_posterior_predictives_summary(*grouping_vars*, *n_posterior_predictives=500*)

Calculates summary of posterior predictives of choices, separately for a list of grouping variables.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials in all conditions combination.

For example, if grouping_vars=['reward', 'difficulty'], posterior predictives will be collapsed for all combinations of levels of the reward and difficulty variables.

Parameters

- **grouping_vars** (*list of strings*) – They should be existing grouping variables in the data.
- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.

Returns
out – Pandas DataFrame. The column contains the mean accuracy. The row indes is a pandas.MultIndex, with the grouping variables as higher level and number of samples as lower level.

Return type
DataFrame

`get_posterior_predictives`(*n_posterior_predictives=500*)

Calculates posterior predictives of choices.

Parameters
n_posterior_predictives (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.

Returns
pp_acc – Array of shape (n_samples, n_trials).

Return type
numpy.ndarray

`get_posterior_predictives_df`(*n_posterior_predictives=500*)

Calculates posterior predictives of choices.

Parameters
n_posterior_predictives (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.

Returns
out – Data frame of shape (n_samples, n_trials).

Return type
DataFrame

`get_posterior_predictives_summary`(*n_posterior_predictives=500*)

Calculates summary of posterior predictives of choices.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials.

Parameters
n_posterior_predictives (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.

Returns
out – Data frame, where every row corresponds to a posterior sample. The column contains the mean accuracy for each posterior sample over the whole dataset.

Return type
DataFrame

`plot_mean_grouped_posterior_predictives`(*grouping_vars*, *n_posterior_predictives=500*, ***kwargs*)

Plots the mean posterior predictives of choices, separately for either 1 or 2 grouping variables.

The first grouping variable will be plotted on the x-axis. The second grouping variable, if provided, will be showed with a different color per variable level.

Parameters

- **grouping_vars** (*list of strings*) – They should be existing grouping variables in the data. The list should be of lenght 1 or 2.
- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **x_order** (*list of strings*) – Order to plot the levels of the first grouping variable in, otherwise the levels are inferred from the data objects.
- **hue_order** (*lists of strings*) – Order to plot the levels of the second grouping variable (when provided) in, otherwise the levels are inferred from the data objects.
- **hue_labels** (*list of strings*) – Labels corresponding to hue_order in the legend. Advised to specify hue_order when using this to avoid confusion. Only makes sense when the second grouping variable is provided.
- **show_data** (*bool*) – Whether to show a vertical line for the mean data. Set to False to not show it.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **palette** (*palette name, list, or dict*) – Colors to use for the different levels of the second grouping variable (when provided). Should be something that can be interpreted by color_palette(), or a dictionary mapping hue levels to matplotlib colors.
- **color** (*matplotlib color*) – Color for both the mean data and intervals. Only used when there is 1 grouping variable.
- **ax** (*matplotlib axis, optional*) – If provided, plot on this axis. Default is set to current Axes.
- **intervals_kws** (*dictionary*) – Additional arguments for the matplotlib fill_between function that shows shaded intervals. By default, they are 50 percent transparent.

Returns

ax – Returns the *matplotlib.axes.Axes* object with the plot for further tweaking.

Return type

matplotlib.axes.Axes

plot_mean_posterior_predictives(*n_posterior_predictives*, ***kwargs*)

Plots the mean posterior predictives of choices.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials, and then it's plotted as a distribution. The mean accuracy in the data is plotted as vertical line. This allows to compare the real mean with the BCI or HDI of the predictions.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **show_data** (*bool*) – Whether to show a vertical line for the mean data. Set to False to not show it.
- **color** (*matplotlib color*) – Color for both the mean data and intervals.

- **ax** (*matplotlib axis, optional*) – If provided, plot on this axis. Default is set to current Axes.
- **gridsize** (*int*) – Resolution of the kernel density estimation function, default to 100.
- **clip** (*tuple*) – Range for the kernel density estimation function. Default is min and max values of the distribution.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **intervals_kws** (*dictionary*) – Additional arguments for the matplotlib fill_between function that shows shaded intervals. By default, they are 50 percent transparent.

Returns
ax – Returns the *matplotlib.axes.Axes* object with the plot for further tweaking.

Return type
matplotlib.axes.Axes

CHAPTER

SEVENTEEN

MODELRESULTS CLASS FOR DDMS (OR RLDDMS)

There is one class to inspect model fits of **DDM or combinations of RL and DDM (fitted on choices and response times)**: *DDModelResults*.

The main functions of this class are:

- To assess the model's **convergence** and **mcmc diagnostics**, to make sure that the sampling was successful. This step is crucial and should be preferably done first.
- To provide a measure of the model's **quantitative fit** to the data (i.e., the Watanabe-Akaike information criterion). This is important when comparing the quantitative fit to the data of several, competing models.
- To visualize and make interval-based (either Bayesian Credible Intervals or Higher Density Intervals) inferences on the **posterior distributions** of the model's parameters. This is important when specific hypotheses were made about the parameters' values.
- To calculate and visualize **posterior predictive distributions** of the observed data. This step is important to assess the qualitative fit of the model to the data. Qualitative fit should be assessed not only when comparing different competing models, but also when a single candidate model is fitted. Different ways of calculating posterior predictive distributions are provided, together with different plotting options. In general, emphasis is given to calculating posterior predictive distributions across conditions. This allows us to assess whether a certain behavioral pattern observed in the data (e.g., due to experimental manipulations) can also be reproduced by the model. Finally, posterior predictives are available not only for mean choices and response times, but also for other summary statistics of the response times distributions (i.e., skewness and quantiles).

17.1 All models

class rlssm.fits_DDM.**ModelResults**(*model_label*, *data_info*, *parameters_info*, *priors*, *rhat*, *waic*, *last_values*, *samples*, *trial_samples*)

plot_posteriors(*gridsize=100*, *clip=None*, *show_intervals='HDI'*, *alpha_intervals=0.05*, *intervals_kws=None*, ***kwargs*)

Plots posterior predictives of the model's parameters.

If the model is hierarchical, then only the group parameters are plotted. In particular, group means are plotted in the first row and group standard deviations are plotted in the second row. By default, 95 percent HDI are shown. The kernel density estimation is calculated using scipy.stats.gaussian_kde.

Parameters

- **gridsize** (*int, default to 100*) – Resolution of the kernel density estimation function, default to 100.
- **clip** (*tuple of (float, float), optional*) – Range for the kernel density estimation function. Default is min and max values of the distribution.

- **show_intervals** (*str, default to "HDI"*) – Either "HDI", "BCI", or None. HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float, default to .05*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **intervals_kws** (*dict*) – Additional arguments for *matplotlib.axes.Axes.fill_between* that shows shaded intervals. By default, they are 50 percent transparent.
- ****kwargs** – Additional parameters for seaborn.FacetGrid.

Returns
g

Return type
seaborn.FacetGrid

to_pickle(*filename=None*)

Pickle the fitted model's results object to file.

This can be used to store the model's result and read them and inspect them at a later stage, without having to refit the model.

Parameters
filename (*str, optional*) – File path where the pickled object will be stored. If not specified, is set to

17.2 Diffusion decision models

class rlssm.fits_DDM.**DDModelResults**(*model_label*, *data_info*, *parameters_info*, *priors*, *rhat*, *waic*, *last_values*, *samples*, *trial_samples*, *starting_point_bias*, *drift_variability*, *starting_point_variability*)

DDModelResults allows to perform various model checks on fitted DDM and RLDDM models.

In particular, this can be used to to visualize the estimated posterior distributions and to calculate and visualize the estimated posterior predictives distributions.

show-inheritance
inherited-members

get_grouped_posterior_predictives_summary(*grouping_vars*, *n_posterior_predictives=500*, *quantiles=None*, ***kwargs*)

Calculates summary of posterior predictives of choices and response times, separately for a list of grouping variables.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials in all conditions combination. Response times are summarized using mean, skewness, and quantiles.

For example, if grouping_vars=['reward', 'difficulty'], posterior predictives will be collapsed for all combinations of levels of the reward and difficulty variables.

Parameters

- **grouping_vars** (*list of strings*) – They should be existing grouping variables in the data.

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **quantiles** (*list of floats*) – Quantiles to summarize response times distributions (separately for correct/incorrect) with.
- **noise_constant** (*float*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.
- **rt_max** (*float*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.
- **dt** (*float*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns

out – Pandas DataFrame. The columns contains the mean accuracy for each posterior sample, as well as mean response times, response times skewness and response times quantiles. The row index is a pandas.MultIndex, with the grouping variables as higher level and number of samples as lower level.

Return type

DataFrame

get_posterior_predictives(*n_posterior_predictives=500*, ***kwargs*)

Calculates posterior predictives of choices and response times.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **noise_constant** (*float*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.
- **rt_max** (*float*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.
- **dt** (*float*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns

- **pp_rt** (*np.ndarray*) – Array of shape (n_samples, n_trials) containing predicted response times.
- **pp_acc** (*np.ndarray*) – Array of shape (n_samples, n_trials) containing predicted choices.

get_posterior_predictives_df(*n_posterior_predictives=500*, ***kwargs*)

Calculates posterior predictives of choices and response times.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **noise_constant** (*float*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.

- **rt_max** (*float*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.
- **dt** (*float*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns
out – Data frame of shape (n_samples, n_trials*2). Response times and accuracy are provided as hierarchical column indeces.

Return type
DataFrame

get_posterior_predictives_summary(*n_posterior_predictives=500*, *quantiles=None*, ***kwargs*)

Calculates summary of posterior predictives of choices and response times.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials. Response times are summarized using mean, skewness, and quantiles.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **quantiles** (*list of floats*) – Quantiles to summarize response times distributions (separately for correct/incorrect) with. Default to [.1, .3, .5, .7, .9].
- **noise_constant** (*float*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.
- **rt_max** (*float*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.
- **dt** (*float*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns
out – Pandas DataFrame, where every row corresponds to a posterior sample. The columns contains the mean accuracy for each posterior sample, as well as mean response times, response times skewness and response times quantiles.

Return type
DataFrame

plot_mean_grouped_posterior_predictives(*grouping_vars*, *n_posterior_predictives*, *figsize=(20, 8)*, *post_pred_kws=None*, ***kwargs*)

Plots the mean posterior predictives of choices and response times, separately for either 1 or 2 grouping variables.

The first grouping variable will be plotted on the x-axis. The second grouping variable, if provided, will be showed with a different color per variable level.

Parameters

- **grouping_vars** (*list of strings*) – They should be existing grouping variables in the data. The list should be of lenght 1 or 2.
- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.

- **x_order** (*list of strings*) – Order to plot the levels of the first grouping variable in, otherwise the levels are inferred from the data objects.
- **hue_order** (*lists of strings*) – Order to plot the levels of the second grouping variable (when provided) in, otherwise the levels are inferred from the data objects.
- **hue_labels** (*list of strings*) – Labels corresponding to hue_order in the legend. Advised to specify hue_order when using this to avoid confusion. Only makes sense when the second grouping variable is provided.
- **show_data** (*bool*) – Whether to show a vertical line for the mean data. Set to False to not show it.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **palette** (*palette name, list, or dict*) – Colors to use for the different levels of the second grouping variable (when provided). Should be something that can be interpreted by color_palette(), or a dictionary mapping hue levels to matplotlib colors.
- **color** (*matplotlib color*) – Color for both the mean data and intervals. Only used when there is 1 grouping variable.
- **ax** (*matplotlib axis, optional*) – If provided, plot on this axis. Default is set to current Axes.
- **intervals_kws** (*dictionary*) –

  Additional arguments for the matplotlib fill_between function
  that shows shaded intervals. By default, they are 50 percent transparent.

  post_pred_kws
  [dictionary] Additional parameters to get_grouped_posterior_predictives_summary.

Returns
fig

Return type
matplotlib.figure.Figure

plot_mean_posterior_predictives(*n_posterior_predictives*, *figsize=(20, 8)*, *post_pred_kws=None*, ***kwargs*)

Plots the mean posterior predictives of choices and response times.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials, and then it's plotted as a distribution. The mean accuracy in the data is plotted as vertical line. This allows to compare the real mean with the BCI or HDI of the predictions. The same is done for response times, and are plotted one next to each other.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **figsize** (*tuple*) – figure size of the matplotlib figure
- **show_data** (*bool*) – Whether to show a vertical line for the mean data. Set to False to not show it.
- **color** (*matplotlib color*) – Color for both the mean data and intervals.

- **ax** (*matplotlib axis, optional*) – If provided, plot on this axis. Default is set to current Axes.
- **gridsize** (*int*) – Resolution of the kernel density estimation function, default to 100.
- **clip** (*tuple*) – Range for the kernel density estimation function. Default is min and max values of the distribution.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **intervals_kws** (*dictionary*) – Additional arguments for the matplotlib fill_between function that shows shaded intervals. By default, they are 50 percent transparent.
- **post_pred_kws** (*dictionary*) – Additional parameters to get_posterior_predictives_summary.

Returns
fig

Return type
matplotlib.figure.Figure

plot_quantiles_grouped_posterior_predictives(*n_posterior_predictives*, *grouping_var*, *quantiles=None*, *figsize=(20, 8)*, *post_pred_kws=None*, ***kwargs*)

Plots the quantiles of the posterior predictives of response times, separately for correct/incorrect responses, and 1 grouping variable.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **grouping_vars** (*string*) – Should be an existing grouping variable in the data.
- **quantiles** (*list of floats*) – Quantiles to summarize response times distributions (separately for correct/incorrect) with.
- **figsize** (*tuple*) – figure size of the matplotlib figure
- **show_data** (*bool*) – Whether to show the quantiles of the data. Set to False to not show it.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **kind** (*either 'lines' or 'shades'*) – Two different styles to plot quantile distributions.
- **palette** (*palette name, list, or dict*) – Colors to use for the different levels of the second grouping variable (when provided). Should be something that can be interpreted by color_palette(), or a dictionary mapping hue levels to matplotlib colors.
- **hue_order** (*lists of strings*) – Order to plot the levels of the grouping variable in, otherwise the levels are inferred from the data objects.

- **hue_labels** (*list of strings*) – Labels corresponding to hue_order in the legend. Advised to specify hue_order when using this to avoid confusion.
- **jitter** (*float*) – Amount to jitter the grouping variable's levels for better visualization.
- **scatter_kws** (*dictionary*) – Additional plotting parameters to change how the data points are shown.
- **intervals_kws** (*dictionary*) – Additional plotting parameters to change how the quantile distributions are shown.
- **post_pred_kws** (*dictionary*) – Additional parameters to get_grouped_posterior_predictives_summary.

Returns
fig

Return type
matplotlib.figure.Figure

plot_quantiles_posterior_predictives(*n_posterior_predictives*, *quantiles=None*, *figsize=(20, 8)*, *post_pred_kws=None*, ***kwargs*)

Plots the quantiles of the posterior predictives of response times, separately for correct/incorrect responses.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **quantiles** (*list of floats*) – Quantiles to summarize response times distributions (separately for correct/incorrect) with.
- **figsize** (*tuple*) – figure size of the matplotlib figure
- **show_data** (*bool*) – Whether to show the quantiles of the data. Set to False to not show it.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **kind** (*either 'lines' or 'shades'*) – Two different styles to plot quantile distributions.
- **color** (*matplotlib color*) – Color for both the data and intervals.
- **scatter_kws** (*dictionary*) – Additional plotting parameters to change how the data points are shown.
- **intervals_kws** (*dictionary*) – Additional plotting parameters to change how the quantile distributions are shown.
- **post_pred_kws** (*dictionary*) – Additional parameters to get_posterior_predictives_summary.

Returns
fig

Return type
matplotlib.figure.Figure

17.3 Reinforcement learning diffusion decision models

See *DDModelResults*.

CHAPTER

EIGHTEEN

MODELRESULTS CLASS FOR RACE (OR RL+RACE) MODELS

There is one class to inspect model fits of **race models (RDM, LBA, ARDM, and ALBA) or combinations of RL and race models (fitted on choices and response times)**: *raceModelResults_2A*.

The main functions of this class are:

- To assess the model's **convergence** and **mcmc diagnostics**, to make sure that the sampling was successful. This step is crucial and should be preferably done first.
- To provide a measure of the model's **quantitative fit** to the data (i.e., the Watanabe-Akaike information criterion). This is important when comparing the quantitative fit to the data of several, competing models.
- To visualize and make interval-based (either Bayesian Credible Intervals or Higher Density Intervals) inferences on the **posterior distributions** of the model's parameters. This is important when specific hypotheses were made about the parameters' values.
- To calculate and visualize **posterior predictive distributions** of the observed data. This step is important to assess the qualitative fit of the model to the data. Qualitative fit should be assessed not only when comparing different competing models, but also when a single candidate model is fitted. Different ways of calculating posterior predictive distributions are provided, together with different plotting options. In general, emphasis is given to calculating posterior predictive distributions across conditions. This allows us to assess whether a certain behavioral pattern observed in the data (e.g., due to experimental manipulations) can also be reproduced by the model. Finally, posterior predictives are available not only for mean choices and response times, but also for other summary statistics of the response times distributions (i.e., skewness and quantiles).

18.1 All models

class rlssm.fits_race.**ModelResults**(*model_label*, *data_info*, *parameters_info*, *priors*, *rhat*, *waic*, *last_values*, *samples*, *trial_samples*)

plot_posteriors(*gridsize=100*, *clip=None*, *show_intervals='HDI'*, *alpha_intervals=0.05*, *intervals_kws=None*, ***kwargs*)

Plots posterior predictives of the model's parameters.

If the model is hierarchical, then only the group parameters are plotted. In particular, group means are plotted in the first row and group standard deviations are plotted in the second row. By default, 95 percent HDI are shown. The kernel density estimation is calculated using scipy.stats.gaussian_kde.

Parameters

- **gridsize** (*int, default to 100*) – Resolution of the kernel density estimation function, default to 100.
- **clip** (*tuple of (float, float), optional*) – Range for the kernel density estimation function. Default is min and max values of the distribution.

- **show_intervals** (*str, default to "HDI"*) – Either "HDI", "BCI", or None. HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float, default to .05*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **intervals_kws** (*dict*) – Additional arguments for *matplotlib.axes.Axes.fill_between* that shows shaded intervals. By default, they are 50 percent transparent.
- ****kwargs** – Additional parameters for seaborn.FacetGrid.

Returns
g

Return type
seaborn.FacetGrid

to_pickle(*filename=None*)

Pickle the fitted model's results object to file.

This can be used to store the model's result and read them and inspect them at a later stage, without having to refit the model.

Parameters
filename (*str, optional*) – File path where the pickled object will be stored. If not specified, is set to

18.2 Race diffusion models (RDM, LBA, ARDM, and ALBA)

class rlssm.fits_race.**raceModelResults_2A**(*model_label*, *data_info*, *parameters_info*, *priors*, *rhat*, *waic*, *last_values*, *samples*, *trial_samples*, *family*)

show-inheritance
inherited-members

get_grouped_posterior_predictives_summary(*grouping_vars*, *n_posterior_predictives=500*, *quantiles=None*, ***kwargs*)

Calculates summary of posterior predictives of choices and response times, separately for a list of grouping variables.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials in all conditions combination. Response times are summarized using mean, skewness, and quantiles.

For example, if grouping_vars=['reward', 'difficulty'], posterior predictives will be collapsed for all combinations of levels of the reward and difficulty variables.

Parameters

- **grouping_vars** (*list of strings*) – They should be existing grouping variables in the data.
- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **quantiles** (*list of floats*) – Quantiles to summarize response times distributions (separately for correct/incorrect) with.

- **noise_constant** (*float*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.
- **rt_max** (*float*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.
- **dt** (*float*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns

out – Pandas DataFrame. The columns contains the mean accuracy for each posterior sample, as well as mean response times, response times skewness and response times quantiles. The row index is a pandas.MultIndex, with the grouping variables as higher level and number of samples as lower level.

Return type

DataFrame

get_posterior_predictives_df(*n_posterior_predictives=500*, ***kwargs*)

Calculates posterior predictives of choices and response times.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **noise_constant** (*float*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.
- **rt_max** (*float*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.
- **dt** (*float*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns

out – Data frame of shape (n_samples, n_trials*2). Response times and accuracy are provided as hierarchical column indeces.

Return type

DataFrame

get_posterior_predictives_summary(*n_posterior_predictives=500*, *quantiles=None*, ***kwargs*)

Calculates summary of posterior predictives of choices and response times.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials. Response times are summarized using mean, skewness, and quantiles.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **quantiles** (*list of floats*) – Quantiles to summarize response times distributions (separately for correct/incorrect) with. Default to [.1, .3, .5, .7, .9].
- **noise_constant** (*float*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.
- **rt_max** (*float*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.

- **dt** (*float*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns

out – Pandas DataFrame, where every row corresponds to a posterior sample. The columns contains the mean accuracy for each posterior sample, as well as mean response times, response times skewness and response times quantiles.

Return type

DataFrame

plot_mean_grouped_posterior_predictives(*grouping_vars*, *n_posterior_predictives*, *figsize=(20, 8)*, *post_pred_kws=None*, ***kwargs*)

Plots the mean posterior predictives of choices and response times, separately for either 1 or 2 grouping variables.

The first grouping variable will be plotted on the x-axis. The second grouping variable, if provided, will be showed with a different color per variable level.

Parameters

- **grouping_vars** (*list of strings*) – They should be existing grouping variables in the data. The list should be of lenght 1 or 2.
- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **x_order** (*list of strings*) – Order to plot the levels of the first grouping variable in, otherwise the levels are inferred from the data objects.
- **hue_order** (*lists of strings*) – Order to plot the levels of the second grouping variable (when provided) in, otherwise the levels are inferred from the data objects.
- **hue_labels** (*list of strings*) – Labels corresponding to hue_order in the legend. Advised to specify hue_order when using this to avoid confusion. Only makes sense when the second grouping variable is provided.
- **show_data** (*bool*) – Whether to show a vertical line for the mean data. Set to False to not show it.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **palette** (*palette name, list, or dict*) – Colors to use for the different levels of the second grouping variable (when provided). Should be something that can be interpreted by color_palette(), or a dictionary mapping hue levels to matplotlib colors.
- **color** (*matplotlib color*) – Color for both the mean data and intervals. Only used when there is 1 grouping variable.
- **ax** (*matplotlib axis, optional*) – If provided, plot on this axis. Default is set to current Axes.
- **intervals_kws** (*dictionary*) –

  Additional arguments for the matplotlib fill_between function

  that shows shaded intervals. By default, they are 50 percent transparent.

post_pred_kws
[dictionary] Additional parameters to get_grouped_posterior_predictives_summary.

Returns
fig

Return type
matplotlib.figure.Figure

plot_mean_posterior_predictives(*n_posterior_predictives*, *figsize=(20, 8)*, *post_pred_kws=None*, ***kwargs*)

Plots the mean posterior predictives of choices and response times.

The mean proportion of choices (in this case coded as accuracy) is calculated for each posterior sample across all trials, and then it's plotted as a distribution. The mean accuracy in the data is plotted as vertical line. This allows to compare the real mean with the BCI or HDI of the predictions. The same is done for response times, and are plotted one next to each other.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **figsize** (*tuple*) – figure size of the matplotlib figure
- **show_data** (*bool*) – Whether to show a vertical line for the mean data. Set to False to not show it.
- **color** (*matplotlib color*) – Color for both the mean data and intervals.
- **ax** (*matplotlib axis, optional*) – If provided, plot on this axis. Default is set to current Axes.
- **gridsize** (*int*) – Resolution of the kernel density estimation function, default to 100.
- **clip** (*tuple*) – Range for the kernel density estimation function. Default is min and max values of the distribution.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **intervals_kws** (*dictionary*) – Additional arguments for the matplotlib fill_between function that shows shaded intervals. By default, they are 50 percent transparent.
- **post_pred_kws** (*dictionary*) – Additional parameters to get_posterior_predictives_summary.

Returns
fig

Return type
matplotlib.figure.Figure

plot_quantiles_grouped_posterior_predictives(*n_posterior_predictives*, *grouping_var*, *quantiles=None*, *figsize=(20, 8)*, *post_pred_kws=None*, ***kwargs*)

Plots the quantiles of the posterior predictives of response times, separately for correct/incorrect responses, and 1 grouping variable.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **grouping_vars** (*string*) – Should be an existing grouping variable in the data.
- **quantiles** (*list of floats*) – Quantiles to summarize response times distributions (separately for correct/incorrect) with.
- **figsize** (*tuple*) – figure size of the matplotlib figure
- **show_data** (*bool*) – Whether to show the quantiles of the data. Set to False to not show it.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **kind** (*either 'lines' or 'shades'*) – Two different styles to plot quantile distributions.
- **palette** (*palette name, list, or dict*) – Colors to use for the different levels of the second grouping variable (when provided). Should be something that can be interpreted by color_palette(), or a dictionary mapping hue levels to matplotlib colors.
- **hue_order** (*lists of strings*) – Order to plot the levels of the grouping variable in, otherwise the levels are inferred from the data objects.
- **hue_labels** (*list of strings*) – Labels corresponding to hue_order in the legend. Advised to specify hue_order when using this to avoid confusion.
- **jitter** (*float*) – Amount to jitter the grouping variable's levels for better visualization.
- **scatter_kws** (*dictionary*) – Additional plotting parameters to change how the data points are shown.
- **intervals_kws** (*dictionary*) – Additional plotting parameters to change how the quantile distributions are shown.
- **post_pred_kws** (*dictionary*) – Additional parameters to get_grouped_posterior_predictives_summary.

Returns
fig

Return type
matplotlib.figure.Figure

plot_quantiles_posterior_predictives(*n_posterior_predictives*, *quantiles=None*, *figsize=(20, 8)*, *post_pred_kws=None*, ***kwargs*)

Plots the quantiles of the posterior predictives of response times, separately for correct/incorrect responses.

Parameters

- **n_posterior_predictives** (*int*) – Number of posterior samples to use for posterior predictives calculation. If n_posterior_predictives is bigger than the posterior samples, then calculation will continue with the total number of posterior samples.
- **quantiles** (*list of floats*) – Quantiles to summarize response times distributions (separately for correct/incorrect) with.
- **figsize** (*tuple*) – figure size of the matplotlib figure

- **show_data** (*bool*) – Whether to show the quantiles of the data. Set to False to not show it.
- **show_intervals** (*either "HDI", "BCI", or None*) – HDI is better when the distribution is not simmetrical. If None, then no intervals are shown.
- **alpha_intervals** (*float*) – Alpha level for the intervals. Default is 5 percent which gives 95 percent BCIs and HDIs.
- **kind** (*either 'lines' or 'shades'*) – Two different styles to plot quantile distributions.
- **color** (*matplotlib color*) – Color for both the data and intervals.
- **scatter_kws** (*dictionary*) – Additional plotting parameters to change how the data points are shown.
- **intervals_kws** (*dictionary*) – Additional plotting parameters to change how the quantile distributions are shown.
- **post_pred_kws** (*dictionary*) – Additional parameters to get_posterior_predictives_summary.

Returns
fig

Return type
matplotlib.figure.Figure

18.3 Reinforcement learning race diffusion models (RLRDM, RLLBA, RLARDM, and RLALBA)

See *raceModelResults_2A*.

CHAPTER

NINETEEN

SIMULATE DATA WITH THE DDM

These functions can be used to simulate data of a single participant or of a group of participants, given a set of parameter values.

These functions can be thus used for parameter recovery: A model can be fit on the simulated data in order to compare the generating parameters with their estimated posterior distributions. For such purpose, *simulate_ddm* (for a single participant) and *simulate_hier_ddm* (for a group of participants) should be used.

For faster calculations, parameters can be given as numpy.ndarrays to *random_ddm* and *random_ddm_vector*.

19.1 In pandas

rlssm.random.**simulate_ddm**(*n_trials*, *gen_drift*, *gen_threshold*, *gen_ndt*, *gen_rel_sp=0.5*, *participant_label=1*, *gen_drift_trialsd=None*, *gen_rel_sp_trialsd=None*, ***kwargs*)

Simulates behavior (rt and accuracy) according to the diffusion decision model.

This function is to simulate data for, for example, parameter recovery.

Simulates data for one participant.

In this parametrization, it is assumed that 0 is the lower threshold, and, when *rel_sp* = .5, the diffusion process starts halfway through the threshold value.

Note: When *gen_drift_trialsd* is not specified, there is no across-trial variability in the drift-rate.

Instead, when *gen_drift_trialsd* is specified, the trial-by-trial drift-rate has the following distribution:

- drift ~ normal(gen_drift, gen_drift_trialsd).

Similarly, when *gen_rel_sp_trialsd* is not specified, there is no across-trial variability starting point.

Instead, when *gen_rel_sp_trialsd* is specified, the trial-by-trial relative starting point has the following distribution:

- rel_sp ~ Phi(normal(rel_sp, gen_rel_sp_trialsd)).

In this case, *gen_rel_sp* is first trasformed to the -Inf/+Inf scale, so the input value is the same (no bias still corresponds to .5).

Parameters

- **n_trials** (*int*) – Number of trials to be simulated.
- **gen_drift** (*float*) – Drift-rate of the diffusion decision model.

- **gen_threshold** (*float*) – Threshold of the diffusion decision model. Should be positive.
- **gen_ndt** (*float*) – Non decision time of the diffusion decision model, in seconds. Should be positive.
- **gen_rel_sp** (*float, default .5*) – Relative starting point of the diffusion decision model. Should be higher than 0 and smaller than 1. When is 0.5 (default), there is no bias.
- **gen_drift_trialsd** (*float, optional*) – Across trial variability in the drift-rate. Should be positive.
- **gen_rel_sp_trialsd** (*float, optional*) – Across trial variability in the realtive starting point. Should be positive.
- **participant_label** (*string or float, default 1*) – What will appear in the participant column of the output data.
- ****kwargs** – Additional arguments to rlssm.random.random_ddm().

Returns

data – *pandas.DataFrame*, with n_trials rows. Columns contain simulated response times and accuracy ["rt", "accuracy"], as well as the generating parameters (both for each trial and across-trials when there is across-trial variability).

Return type

DataFrame

Examples

Simulate 1000 trials from 1 participant.

Relative starting point is set towards the upper bound (higher than .5), so in this case there will be more accurate and fast decisions:

```
from rlssm.random import simulate_ddm
>>> data = simulate_ddm(n_trials=1000,
                        gen_drift=.8,
                        gen_threshold=1.3,
                        gen_ndt=.23,
                        gen_rel_sp=.6)

>>> print(data.head())
        participant  drift  rel_sp  threshold   ndt     rt  accuracy
trial
1                 1    0.8     0.6        1.3  0.23  0.344       1.0
2                 1    0.8     0.6        1.3  0.23  0.376       0.0
3                 1    0.8     0.6        1.3  0.23  0.390       1.0
4                 1    0.8     0.6        1.3  0.23  0.434       0.0
5                 1    0.8     0.6        1.3  0.23  0.520       1.0
```

To have trial number as a column:

```
>>> print(data.reset_index())
    trial  participant  drift  rel_sp  threshold   ndt     rt  accuracy
0       1            1    0.8     0.6        1.3  0.23  0.344       1.0
1       2            1    0.8     0.6        1.3  0.23  0.376       0.0
2       3            1    0.8     0.6        1.3  0.23  0.390       1.0
```

(continues on next page)

(continued from previous page)

```
3        4            1    0.8     0.6        1.3  0.23  0.434       0.0
4        5            1    0.8     0.6        1.3  0.23  0.520       1.0
..     ...          ...    ...     ...        ...   ...    ...       ...
995    996            1    0.8     0.6        1.3  0.23  0.423       1.0
996    997            1    0.8     0.6        1.3  0.23  0.956       1.0
997    998            1    0.8     0.6        1.3  0.23  0.347       1.0
998    999            1    0.8     0.6        1.3  0.23  0.414       1.0
999   1000            1    0.8     0.6        1.3  0.23  0.401       1.0

[1000 rows x 8 columns]
```

rlssm.random.**simulate_hier_ddm**(*n_trials*, *n_participants*, *gen_mu_drift*, *gen_sd_drift*, *gen_mu_threshold*, *gen_sd_threshold*, *gen_mu_ndt*, *gen_sd_ndt*, *gen_mu_rel_sp=0.5*, *gen_sd_rel_sp=None*, ***kwargs*)

Simulates behavior (rt and accuracy) according to the diffusion decision model.

This function is to simulate data for, for example, parameter recovery.

Simulates hierarchical data for a group of participants.

In this parametrization, it is assumed that 0 is the lower threshold, and, when *rel_sp* = .5, the diffusion process starts halfway through the threshold value.

The individual parameters have the following distributions:

- drift ~ normal(gen_mu_drift, gen_sd_drift)
- threshold ~ log(1 + exp(normal(gen_mu_threshold, gen_sd_threshold)))
- ndt ~ log(1 + exp(normal(gen_mu_ndt, gen_sd_ndt)))

Note: When *gen_sd_rel_sp* is not specified, the relative starting point is assumed to be fixed across participants at *gen_mu_rel_sp*.

Instead, when *gen_sd_rel_sp* is specified, the starting point has the following distribution:

- rel_sp ~ Phi(normal(gen_mu_rel_sp, gen_sd_rel_sp))

Parameters

- **n_trials** (*int*) – Number of trials to be simulated.
- **n_participants** (*int*) – Number of participants to be simulated.
- **gen_mu_drift** (*float*) – Group-mean of the drift-rate of the diffusion decision model.
- **gen_sd_drift** (*float*) – Group-standard deviation of the drift-rate of the diffusion decision model.
- **gen_mu_threshold** (*float*) – Group-mean of the threshold of the diffusion decision model.
- **gen_sd_threshold** (*float*) – Group-standard deviation of the threshold of the diffusion decision model.
- **gen_mu_ndt** (*float*) – Group-mean of the non decision time of the diffusion decision model.

- **gen_sd_ndt** (*float*) – Group-standard deviation of the non decision time of the diffusion decision model.
- **gen_mu_rel_sp** (*float, default .5*) – Relative starting point of the diffusion decision model. When *gen_sd_rel_sp* is not specified, *gen_mu_rel_sp* is fixed across participants. When *gen_sd_rel_sp* is specified, *gen_mu_rel_sp* is the group-mean of the starting point.
- **gen_sd_rel_sp** (*float, optional*) – Group-standard deviation of the relative starting point of the diffusion decision model.
- ****kwargs** – Additional arguments to *rlssm.random.random_ddm()*.

Returns

data – *pandas.DataFrame*, with n_trials*n_participants rows. Columns contain simulated response times and accuracy ["rt", "accuracy"], as well as the generating parameters (at the participant level).

Return type

DataFrame

Examples

Simulate data from 15 participants, with 200 trials each.

Relative starting point is on average across participants towards the upper bound. So, in this case, there will be more accurate and fast decisions:

```
from rlssm.random import simulate_hier_ddm
>>> data = simulate_hier_ddm(n_trials=200,
                             n_participants=15,
                             gen_mu_drift=.6, gen_sd_drift=.3,
                             gen_mu_threshold=.5, gen_sd_threshold=.1,
                             gen_mu_ndt=-1.2, gen_sd_ndt=.05,
                             gen_mu_rel_sp=.1, gen_sd_rel_sp=.05)

>>> print(data.head())
                      drift  threshold       ndt    rel_sp        rt  accuracy
participant trial
1           1      0.773536   1.753562  0.300878  0.553373  0.368878       1.0
            1      0.773536   1.753562  0.300878  0.553373  0.688878       1.0
            1      0.773536   1.753562  0.300878  0.553373  0.401878       1.0
            1      0.773536   1.753562  0.300878  0.553373  1.717878       1.0
            1      0.773536   1.753562  0.300878  0.553373  0.417878       1.0
```

Get mean response time and accuracy per participant:

```
>>> print(data.groupby('participant').mean()[['rt', 'accuracy']])
                   rt  accuracy
participant
1            0.990313     0.840
2            0.903228     0.740
3            1.024509     0.815
4            0.680104     0.760
5            0.994501     0.770
6            0.910615     0.865
7            0.782978     0.700
```

(continues on next page)

(continued from previous page)

```
8            1.189268     0.740
9            0.997170     0.760
10           0.966897     0.750
11           0.730522     0.855
12           1.011454     0.590
13           0.972070     0.675
14           0.849755     0.625
15           0.940542     0.785
```

To have participant and trial numbers as a columns:

```
>>> print(data.reset_index())
      participant  trial     drift  threshold       ndt    rel_sp        rt ␣
↪accuracy
0               1      1  0.773536   1.753562  0.300878  0.553373  0.368878       1.
↪0
1               1      1  0.773536   1.753562  0.300878  0.553373  0.688878       1.
↪0
2               1      1  0.773536   1.753562  0.300878  0.553373  0.401878       1.
↪0
3               1      1  0.773536   1.753562  0.300878  0.553373  1.717878       1.
↪0
4               1      1  0.773536   1.753562  0.300878  0.553373  0.417878       1.
↪0
...           ...    ...       ...        ...       ...       ...       ...       ..
↪.
2995           15    200  0.586573   1.703662  0.302842  0.556116  0.826842       1.
↪0
2996           15    200  0.586573   1.703662  0.302842  0.556116  0.925842       1.
↪0
2997           15    200  0.586573   1.703662  0.302842  0.556116  0.832842       1.
↪0
2998           15    200  0.586573   1.703662  0.302842  0.556116  0.628842       1.
↪0
2999           15    200  0.586573   1.703662  0.302842  0.556116  0.856842       1.
↪0

[3000 rows x 8 columns]
```

19.2 In numpy

rlssm.random.**random_ddm**(*drift*, *threshold*, *ndt*, *rel_sp=0.5*, *noise_constant=1*, *dt=0.001*, *max_rt=10*)

Simulates behavior (rt and accuracy) according to the diffusion decision model.

In this parametrization, it is assumed that 0 is the lower threshold, and, when rel_sp=1/2, the diffusion process starts halfway through the threshold value.

Note: This function is mainly for the posterior predictive calculations. It assumes that drift, threshold and ndt are provided as numpy.ndarray of shape (n_samples, n_trials).

However, it also works when the rel_sp is given as a float. Drift, threshold and ndt should have the same shape.

Parameters

- **drift** (*numpy.ndarray*) – Shape is usually (n_samples, n_trials). Drift-rate of the diffusion decision model.
- **threshold** (*numpy.ndarray*) – Shape is usually (n_samples, n_trials). Threshold of the diffusion decision model.
- **ndt** (*numpy.ndarray*) – Shape is usually (n_samples, n_trials). Non decision time of the diffusion decision model, in seconds.
- **rel_sp** (*numpy.ndarray or float, default .5*) – When is an array , shape is usually (n_samples, n_trials). Relative starting point of the diffusion decision model.
- **noise_constant** (*float, default 1*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.
- **max_rt** (*float, default 10*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.
- **dt** (*float, default 0.001*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns

- **rt** (*numpy.ndarray*) – Shape is the same as the input parameters. Contains simulated response times according to the diffusion decision model. Every element corresponds to the set of parameters given as input with the same shape.
- **acc** (*numpy.ndarray*) – Shape is the same as the input parameters. Contains simulated accuracy according to the diffusion decision model. Every element corresponds to the set of parameters given as input with the same shape.

rlssm.random.**random_ddm_vector**(*drift*, *threshold*, *ndt*, *rel_sp=0.5*, *noise_constant=1*, *dt=0.001*, *rt_max=10*)

Simulates behavior (rt and accuracy) according to the diffusion decision model.

In this parametrization, it is assumed that 0 is the lower threshold, and, when rel_sp=1/2, the diffusion process starts halfway through the threshold value.

Note: This is a vectorized version of rlssm.random.random_ddm(). It seems to be generally slower, but might work for higher dt values and shorter rt_max (with less precision). There is more trade-off between dt and rt_max here compared to the random_ddm function.

This function is mainly for the posterior predictive calculations. It assumes that drift, threshold and ndt are provided as numpy.ndarray of shape (n_samples, n_trials).

However, it also works when the rel_sp and/or the ndt are given as floats. Drift and threshold should have the same shape.

Parameters

- **drift** (*numpy.ndarray*) – Shape is usually (n_samples, n_trials). Drift-rate of the diffusion decision model.

- **threshold** (*numpy.ndarray*) – Shape is usually (n_samples, n_trials). Threshold of the diffusion decision model.
- **ndt** (*numpy.ndarray or float*) – Shape is usually (n_samples, n_trials). Non decision time of the diffusion decision model, in seconds.
- **rel_sp** (*numpy.ndarray or float, default .5*) – When is an array , shape is usually (n_samples, n_trials). Relative starting point of the diffusion decision model.
- **noise_constant** (*float, default 1*) – Scaling factor of the diffusion decision model. If changed, drift and threshold would be scaled accordingly. Not to be changed in most applications.
- **max_rt** (*float, default 10*) – Controls the maximum rts that can be predicted. Making this higher might make the function a bit slower.
- **dt** (*float, default 0.001*) – Controls the time resolution of the diffusion decision model. Default is 1 msec. Lower values of dt make the function more precise but much slower.

Returns

- **rt** (*numpy.ndarray*) – Shape is the same as the input parameters. Contains simulated response times according to the diffusion decision model. Every element corresponds to the set of parameters given as input with the same shape.
- **acc** (*numpy.ndarray*) – Shape is the same as the input parameters. Contains simulated accuracy according to the diffusion decision model. Every element corresponds to the set of parameters given as input with the same shape.

CHAPTER

TWENTY

SIMULATE DATA WITH RACE MODELS

Note: At the moment, these functions are not fully optimized and only possible to simulate data from one subject.

20.1 In numpy

rlssm.random.**random_rdm_2A**(*cor_drift*, *inc_drift*, *threshold*, *ndt*, *noise_constant=1*, *dt=0.001*, *max_rt=10*)

rlssm.random.**random_lba_2A**(*k*, *A*, *tau*, *cor_drift*, *inc_drift*)

CHAPTER

TWENTYONE

SIMULATE DATA WITH RL MODELS, RLDDMS, AND RL+RACE MODELS

These functions can be used to simulate data of a single participant or of a group of participants, given a set of parameter values.

These functions can be thus used for parameter recovery: A model can be fit on the simulated data in order to compare the generating parameters with their estimated posterior distributions.

Note: At the moment, only non-hierarchical RLRDM data can be simulated.

21.1 Simulate RL stimuli

rlssm.random.**generate_task_design_fontanesi**(*n_trials_block*, *n_blocks*, *n_participants*, *trial_types*, *mean_options*, *sd_options*)

Generates the RL stimuli as in the 2019 Fontanesi et al.'s paper.

Note: In the original paper we corrected for repetition and order presentation of values too. This is not implemented here.

Parameters

- **n_trials_block** (*int*) – Number of trials per learning session.
- **n_blocks** (*int*) – Number of learning session per participant.
- **n_participants** (*int*) – Number of participants.
- **trial_types** (*list of strings*) – List containing possible pairs of options. E.g., in the original experiment: ['1-2', '1-3', '2-4', '3-4']. It is important that they are separated by a '-', and that they are numbered from 1 to n_options (4 in the example). Also, the "incorrect" option of the couple should go first in each pair.
- **mean_options** (*list or array of floats*) – Mean reward for each option. The length should correspond to n_options.
- **sd_options** (*list or array of floats*) – SD reward for each option. The length should correspond to n_options.

Returns

task_design – *pandas.DataFrame*, with n_trials_block*n_blocks rows. Columns contain:

"f_cor", "f_inc", "trial_type", "cor_option", "inc_option", "trial_block", "block_label", "participant".

Return type
DataFrame

21.2 Simulate only-choices RL data

rlssm.random.**simulate_rl_2A**(*task_design*, *gen_alpha*, *gen_sensitivity*, *initial_value_learning=0*)

Simulates behavior (accuracy) according to a RL model,

where the learning component is the Q learning (delta learning rule) and the choice rule is the softmax.

This function is to simulate data for, for example, parameter recovery. Simulates data for one participant.

Note: The number of options can be actaully higher than 2, but only 2 options (one correct, one incorrect) are presented in each trial. It is important that "trial_block" is set to 1 at the beginning of each learning session (when the Q values at resetted) and that the "block_label" is set to 1 at the beginning of the experiment for each participants. There is no special requirement for the participant number.

Parameters

- **task_design** (*DataFrame*) – *pandas.DataFrame*, with n_trials_block*n_blocks rows. Columns contain: "f_cor", "f_inc", "trial_type", "cor_option", "inc_option", "trial_block", "block_label", "participant".
- **gen_alpha** (*float or list of floats*) – The generating learning rate. It should be a value between 0 (no updating) and 1 (full updating). If a list of 2 values is provided then 2 separate learning rates for positive and negative prediction error are used.
- **gen_sensitivity** (*float*) – The generating sensitivity parameter for the soft_max choice rule. It should be a value higher than 0 (the higher, the more sensitivity to value differences).
- **initial_value_learning** (*float*) – The initial value for Q learning.

Returns
data – *pandas.DataFrame*, that is the task_design, plus: 'Q_cor', 'Q_inc', 'alpha', 'sensitivity', 'p_cor', and 'accuracy'.

Return type
DataFrame

rlssm.random.**simulate_hier_rl_2A**(*task_design*, *gen_mu_alpha*, *gen_sd_alpha*, *gen_mu_sensitivity*, *gen_sd_sensitivity*, *initial_value_learning=0*)

Simulates behavior (accuracy) according to a RL model, where the learning component is the Q learning (delta learning rule) and the choice rule is the softmax.

Simulates hierarchical data for a group of participants. The individual parameters have the following distributions:

- alpha ~ Phi(normal(gen_mu_alpha, gen_sd_alpha))
- sensitivity ~ log(1 + exp(normal(gen_mu_sensitivity, gen_sd_sensitivity)))

When 2 learning rates are estimated:

- alpha_pos ~ Phi(normal(gen_mu_alpha[0], gen_sd_alpha[1]))

- alpha_neg ~ Phi(normal(gen_mu_alpha[1], gen_sd_alpha[1]))

Note: The number of options can be actaully higher than 2, but only 2 options (one correct, one incorrect) are presented in each trial. It is important that "trial_block" is set to 1 at the beginning of each learning session (when the Q values at resetted) and that the "block_label" is set to 1 at the beginning of the experiment for each participants. There is no special requirement for the participant number.

Parameters

- **task_design** (*DataFrame*) – *pandas.DataFrame*, with n_trials_block*n_blocks rows. Columns contain: "f_cor", "f_inc", "trial_type", "cor_option", "inc_option", "trial_block", "block_label", "participant".
- **gen_mu_alpha** (*float or list of floats*) – The generating group mean of the learning rate. If a list of 2 values is provided then 2 separate learning rates for positive and negative prediction error are used.
- **gen_sd_alpha** (*float or list of floats*) – The generating group SD of the learning rate. If a list of 2 values is provided then 2 separate learning rates for positive and negative prediction error are used.
- **gen_mu_sensitivity** (*float*) – The generating group mean of the sensitivity parameter for the soft_max choice rule.
- **gen_sd_sensitivity** (*float*) – The generating group SD of the sensitivity parameter for the soft_max choice rule.
- **initial_value_learning** (*float*) – The initial value for Q learning.

Returns

data – *pandas.DataFrame*, that is the task_design, plus: 'Q_cor', 'Q_inc', 'alpha', 'sensitivity', 'p_cor', and 'accuracy'.

Return type

DataFrame

21.3 Simulate RLDDM data (choices and RTs)

rlssm.random.**simulate_rlddm_2A**(*task_design*, *gen_alpha*, *gen_drift_scaling*, *gen_threshold*, *gen_ndt*, *initial_value_learning=0*, ***kwargs*)

Simulates behavior (rt and accuracy) according to a RLDDM model,

where the learning component is the Q learning (delta learning rule) and the choice rule is the DDM.

Simulates data for one participant.

In this parametrization, it is assumed that 0 is the lower threshold, and the diffusion process starts halfway through the threshold value.

Note: The number of options can be actaully higher than 2, but only 2 options (one correct, one incorrect) are presented in each trial. It is important that "trial_block" is set to 1 at the beginning of each learning session (when the Q values at resetted) and that the "block_label" is set to 1 at the beginning of the experiment for each participants. There is no special requirement for the participant number.

Parameters

- **task_design** (*DataFrame*) – *pandas.DataFrame*, with n_trials_block*n_blocks rows. Columns contain: "f_cor", "f_inc", "trial_type", "cor_option", "inc_option", "trial_block", "block_label", "participant".
- **gen_alpha** (*float or list of floats*) – The generating learning rate. It should be a value between 0 (no updating) and 1 (full updating). If a list of 2 values is provided then 2 separate learning rates for positive and negative prediction error are used.
- **gen_drift_scaling** (*float*) – Drift-rate scaling of the RLDDM.
- **gen_threshold** (*float*) – Threshold of the diffusion decision model. Should be positive.
- **gen_ndt** (*float*) – Non decision time of the diffusion decision model, in seconds. Should be positive.
- **initial_value_learning** (*float*) – The initial value for Q learning.
- ****kwargs** – Additional arguments to rlssm.random.random_ddm().

Returns

data – *pandas.DataFrame*, that is the task_design, plus: 'Q_cor', 'Q_inc', 'drift', 'alpha', 'drift_scaling', 'threshold', 'ndt', 'rt', and 'accuracy'.

Return type

DataFrame

rlssm.random.**simulate_hier_rlddm_2A**(*task_design*, *gen_mu_alpha*, *gen_sd_alpha*, *gen_mu_drift_scaling*, *gen_sd_drift_scaling*, *gen_mu_threshold*, *gen_sd_threshold*, *gen_mu_ndt*, *gen_sd_ndt*, *initial_value_learning=0*, ***kwargs*)

Simulates behavior (rt and accuracy) according to a RLDDM model,

where the learning component is the Q learning (delta learning rule) and the choice rule is the DDM.

Simulates hierarchical data for a group of participants.

In this parametrization, it is assumed that 0 is the lower threshold, and the diffusion process starts halfway through the threshold value.

The individual parameters have the following distributions:

- alpha ~ Phi(normal(gen_mu_alpha, gen_sd_alpha))
- drift_scaling ~ log(1 + exp(normal(gen_mu_drift, gen_sd_drift)))
- threshold ~ log(1 + exp(normal(gen_mu_threshold, gen_sd_threshold)))
- ndt ~ log(1 + exp(normal(gen_mu_ndt, gen_sd_ndt)))

When 2 learning rates are estimated:

- alpha_pos ~ Phi(normal(gen_mu_alpha[0], gen_sd_alpha[1]))
- alpha_neg ~ Phi(normal(gen_mu_alpha[1], gen_sd_alpha[1]))

Note: The number of options can be actaully higher than 2, but only 2 options (one correct, one incorrect) are presented in each trial. It is important that "trial_block" is set to 1 at the beginning of each learning session (when the Q values at resetted) and that the "block_label" is set to 1 at the beginning of the experiment for each participants. There is no special requirement for the participant number.

Parameters

- **task_design** (*DataFrame*) – *pandas.DataFrame*, with n_trials_block*n_blocks rows. Columns contain: "f_cor", "f_inc", "trial_type", "cor_option", "inc_option", "trial_block", "block_label", "participant".
- **gen_mu_alpha** (*float or list of floats*) – The generating group mean of the learning rate. If a list of 2 values is provided then 2 separate learning rates for positive and negative prediction error are used.
- **gen_sd_alpha** (*float or list of floats*) – The generating group SD of the learning rate. If a list of 2 values is provided then 2 separate learning rates for positive and negative prediction error are used.
- **gen_mu_drift_scaling** (*float*) – Group-mean of the drift-rate scaling of the RLDDM.
- **gen_sd_drift_scaling** (*float*) – Group-standard deviation of the drift-rate scaling of the RLDDM.
- **gen_mu_threshold** (*float*) – Group-mean of the threshold of the RLDDM.
- **gen_sd_threshold** (*float*) – Group-standard deviation of the threshold of the RLDDM.
- **gen_mu_ndt** (*float*) – Group-mean of the non decision time of the RLDDM.
- **gen_sd_ndt** (*float*) – Group-standard deviation of the non decision time of the RLDDM.
- **initial_value_learning** (*float*) – The initial value for Q learning.
- ****kwargs** – Additional arguments to rlssm.random.random_ddm().

Returns

data – *pandas.DataFrame*, that is the task_design, plus: 'Q_cor', 'Q_inc', 'drift', 'alpha', 'drift_scaling', 'threshold', 'ndt', 'rt', and 'accuracy'.

Return type

DataFrame

21.4 Simulate RLRDM data (choices and RTs)

rlssm.random.**simulate_rlrdm_2A**(*task_design*, *gen_alpha*, *gen_drift_scaling*, *gen_threshold*, *gen_ndt*, *initial_value_learning=0*, ***kwargs*)

CHAPTER

TWENTYTWO

INDEX

- genindex

INDEX

Symbols

A

C

D

F

G

R

S

T