Conditioned fear generalization

Marcello Calza

November 26, 2025

0.1 Neurobiology of Fear Generalization

0.1.1 Introduction to Fear Generalization

Fear generalization is a psychological phenomenon where a response to a specific stimuli is extended to similar stimulus. This can be categorized into positive and negative generalizations:

- **Positive Generalization:** This occurs when the fear response to a threat conditioned stimuli (`CS+`) extends to other stimulus that closely resemble the `CS+`. It involves increasing activation of a subsets of brain regions to cues with increasing similarity to the `CS+`. These regions are sensitive to threat cues, facilitating a rapid response to potential dangers.

- **Negative Generalization:** In contrast, negative generalization involves a decreased fear response as the similarity to the `CS+` decreases. This is characterized by a subset of brain regions with activation peaking to a safety conditioned stimuli (`CS-`), and diminishing to cues with increasing similarity to the `CS+`. These regions are responsible of evaluating and inhibiting fear responses, providing a more measured approach to less threatening or neutral stimuli.

0.1.1.1 Methodology for Identifying Relevant Neural Substrates

Functional magnetic resonance imaging (fMRI) studies have been pivotal in mapping the neural substrates of fear generalization. This method allows researchers to observe brain activity and identify regions involved in the generalization process by tracking changes in blood flow associated with neuronal activity during exposure to various stimuli.

0.1.1.2 A Meta-analytic Study to Investigate Fear Generalization

The study conducted by Webler et al. (2021) provides the first coherent meta-analysis of fMRI investigations of neural substrates involved in conditioned fear generalization in humans. In the study, by leveraging datasets's statistical parametric maps (SPMs), the Seed-based d Mapping with Permutation of Subject Images (SDM-PSI) neuroimaging method is used to produce voxel brain maps of activations forming, both linear and quadratic positive and linear negative generalization gradients for a total of six different studies. All the studies used in the meta-analysis were conducted in healthy humans, an adversive stimuli (a shock) was used as an unconditioned stimuli (`US`) for the `CS+`, but a safety conditioned stimulus (`CS-`) was not allways present and the number of `CS+` and general stimulus presentations varied across studies. This meta-analysis work effectively summarizes the neural substrates of positive and negative conditioned generalization and provides important insights regarding the subject, as well as a neural model of fear generalization informed by the study results.

0.1.2 Neural Substrates of Positive Fear Generalization

Thanks to Webler et al. (2021) and other recent studies' findings it is possible to pin down the brain regions involved in conditioned positive fear generalization, along with their specific roles and interactions.

0.1.2.1 Cingulo-opercular Network

This network is crucial for maintaining alertness, directing attention towards relevant events (like potentially conditioned stimuli), and optimizing responses to such events. Furthermore it is probable that two nodes of this network may be linked to unique positive generalization processes:

- the **Anterior Insula (AI)** has been linked to interoceptive awareness when it comes to fear, so the activation of AI related to positive generalization may indicate an increasing conscious awareness that the body is in an axious state, given the presented possible threat reflected by stimuli that are similar to the `CS+`.

- Instead, the **dorsal Anterior Cingulate Cortex (dACC)** and the adjacent **dmPFC**, have been found to have a role in risk analysis when it comes to fear response, so their observed activation linked to positive generalization could be caused by rising levels of perceived risk due to presentations of stimulus with increasing similarity to the `CS+`.

0.1.2.2 Frontoparietal Network

This network is involved in multiple cognitive functions such as attention, cognitive control and emotional regulation that could be linked to positive fear generalization. Two frontoparietal brain regions in specific have been found to have greater activation as a presented stimuli increases in `CS+` resemblance:

- the **Lateral Prefrontal Cortex (lPFC)** engagement could be caused by an increase in cognitive load due to anxiety-driven growing attention to stimulus similar to the `CS+` combined with the necessity of completing experiment related tasks (like attending stimulus and giving fear ratings). A second possible cause of lPFC increased activation associated with positive generalization could be its attempt to down-regulate negative emotions such as fear through inhibition of subcortical brain regions like the amygdala fear network.
- The **Inferior Parietal Lobule (IPL)** has been found to be implicated in the retrieval of semantic and episodic memory, with its function involving stimulus driven attentional shift towards salient external events, therefore its increased activation related to positive generalization may be the result of attentional shifting towards external stimuli with increasing similarity to the `CS+`, or towards internal presentations of the `CS+` triggered by the displaying of the stimuli.

0.1.2.3 Brainstem Nuclei

The brainstem loci has an important role in the production of autonomic and behavioral responses to salient stimuli, and three brainstem nuclei have shown to be possibly involed in positive fear generalization:

- the **Locus Coeruleus (LC)** regulates autonomic arousal, attentional orienting and learning via noradrenaline transmission to various brain regions, that include a projection that extends to the hippocampus, where the noradrenergic signaling affects plasticity causing the retrieval of threat related memories. Therefore LC activation related to positive fear generalization may be caused by presentations of stimuli similar to the `CS+`, which increase attention arousal and trigger the retrieval of the `CS+` memory trace.

- The **Periaqueductal Gray (PAG)** is involved in defensive behaviors in response to threat, thus its growth in activation as presented stimulus increase in similarity to the `CS+` may be caused by the PAG preparing the body for potential danger.
- The **Ventral Tegmental Area (VTA)** is related to the reward system and motivational aspects of fear learning, modulating behavioral responses to fear-relevant stimulus. Its increased activation associated with positive fear generalization may be caused by unexpected omission of the `US` during unreinforced `CS+` and similar stimulus presentations, which could generate VTA-mediated positive prediction error, which can effectively increase safety learning by updating stimuli-US assocations.

0.1.2.4 Striatal-thalamic Areas

These areas are associated with habit formation and procedural learning, they are involved in a circuit that enables the selection and execution of motivated behaviors:

- the **Striatal nuclei**, including the caudate, are responsible of forming the input for the circuit and signaling if a specific action should be executed or inhibited.
- The **Thalamic nuclei**, mainly motoric ventral lateral (VLN) and ventral anterior (VAN) nuclei, are then disinhibited to execute the actions selected by the striatal nuclei.

The engagement of these areas linked to positive fear generalization may be caused by increased threat response actions selection and/or execution, which reflect defensive readiness to potentially dangerous stimuli.
Additionally, a thalamic nuclei, the pulvinar, has been implicated in the processing of salient visual information, and has been found to possibly form a rapid pulvinar-amygdala visual pathway. Therefore positive fear generalization activation of this nuclei may also reflect increased visual processing of the `CS+` and similarly close stimulus and the formation of a thalamic-amygdala threath processing circuit.

0.1.3 Neural Substrates of Negative Fear Generalization

Thanks to the studies it is also possible to identify the brain regions involved in conditioned negative fear generalization, which have an activation pattern that decreases as presentations of stimuli increase in `CS+` similarity, and grows as the stimuli become more distinguishable from the `CS+`, peaking in activation to the safety conditioned stimuli `CS-` (if present).

0.1.3.1 Default Mode Network

This network is generally involved in self-referential mentation, retrospective, prospective memory, and safety responding in threatening contexts. Some default mode nodes in specific showed negative fear generalization effects:

- the **Ventromedial Prefrontal Cortex (vmPFC)** anterior parts are responsible of tracking value of anticipated outcomes, meanwhile the posterior parts of the vmPFC manage inhibition of fear. Therefore the negative generalization effects in vmPFC may result from stimulus with decreasing `CS+` similarity causing increased positive valuation and fear inhibitory responses, that degrade as the stimuli increase in `CS+` resemblance.

- The **Middle Temporal Gyrus (MTG) and Angular Gyrus (AG)** are involved in the retrieval of episodic memories, in dynamic self-referencing and in safety related processes, thus their activation associated with negative fear generalization may result from presentations of stimulus with decreasing `CS+` resemblance causing less disruption of internal mentation and increasing thoughts of security and relief.

- The **Anterior Hippocampus** is thought to play a central role in pattern completion, but is also responsible for pattern separation, thus its engagement related to negative fear generalization may be caused by presentations of stimulus with weak `CS+` resemblance triggering a strong hippocampally mediated pattern separation of themselves and `CS+` internal representations.

0.1.3.2 Amygdala

While typically associated with fear processing, in the Webler et al. (2021) meta analysis (and so in all the analysed studies) the amygdala showed increased activation as presented stimuli decreased in `CS+` similarity, therefore showing negative fear generalization effects, that may reflect activity of basolateral amygdala neurons implicated in reward and inhibitory safety learning, or GABAergic cells, which inhibit threat related amygdala outputs. It is also worth mentioning that the absence of amygdala positive fear generalization effects may be caused by the multiple exposures to the `CS+` during learning trials, that may have caused fMRI repetition suppression, reducing the proportion of threat-sensitive amygdala neurons during the generalization phase of the experiments.

0.1.4 Interaction Between Brain Regions

Thanks to the updated neural account of fear generalization, in the meta-analytic study, Webler et al. (2021) are also able to provide a better explanation on how the interactions between the regions determine the balance of fear response and inhibition.
Based on the Webler et al. (2021) neurobiological model, the sensory information generated by exposures to stimuli resembling the `CS+` activates the **thalamus** which initiates two "roads":

- through the "low road" the **thalamus** signals the threat-related **amygdala** circuits, triggering activity across subcortical (**LC**, **PAG**) and cortical (**AI**, **dmPFC**) regions of the network. Furthermore, "low road" mechanisms, likely through projections from the **basolateral amygdala (BLA)** and **LC**, and **LC** activation by the **central nucleus of the amygdala (CeA)**, engage **CA1 and CA3 hyppocampal subfields**, effectively inducing the **hyppocampus** towards pattern completion.

- Afterwards, the "high road" generates fine-grained visual rappresentations of the stimuli which reach the **hyppocampus**, where their overlap with the previously encoded internal image of the `CS+` is assested.

With enough stimuli/`CS+` overlap and stimuli-evoked **LC-CA3**, **BLA-CA1** signaling the hyppocampus initiates pattern completion resulting in the activation of all the brain regions that showed positive fear generalization effects:

structures associated with fear excitation (**AI**, **dmPFC**, **PAG**, **LC**), and subsequently regions involved in attentional and excitatory control (**IPL**, **lPFC**). However, one component of the positive fear generalization group, the **VTA**, that is responsible for safety learning and reduction of generalized fear, is likely to activate only in response of a stimuli presentation followed by `US` omission. In the case of an insufficient synergy between stimuli/`CS+` overlap and **LC-CA3/BLA-CA1** signaling, dentate gyrus neurons initiate pattern separation in the **hyppocampus**, resulting in the activation of structures involved in negative fear generalization (**vmPFC**, **MTG**, **AG**), that effectively attenuate activity in the amygdala fear-networks.

0.2 Background and Experiments of the Statistical Model

0.2.1 Model Overview

The statistical model developed by Yu et al. (2023) represents a significant advance in understanding the individual differences and the various processes involved in fear generalization. By leveraging a *Bayesian multilevel mixture model*, the research aims to disentangle the latent mechanisms that contribute to these differences, moving beyond traditional generalization research which often describes fear generalization as a static-like system, and focuses solely on overall-level behaviors. The model is peculiar for four main reasons:

1. it employs *Bayesian statistics* to characterize the uncertainty about the parameters as probability distributions, while making the most of the available evidence.

2. It has a *multilevel structure* to take into account the individual differences by inferring parameters values both from the individual and group levels.

3. It considers the generalization behaviour as a *dynamic system* which could emerge from several different processes, like learning, perception and generalization.

4. It incorporates a *mixture framework* that allows to allocate participants into potential clinical relevant groups: *Non-Learners*, *Overgeneralizers*, *Physical Generalizers* and *Perceptual Generalizers.*

In syntesis, while traditional approaches to studying fear generalization have predominantly relied on descriptive statistics and assumed a single generalization mechanism across all individuals, this model challenges that assumption by considering the variability and individual differences that influence generalization behaviors, which can be crucial for understanding related psychopathologies such as anxiety disorders, autism, and obsessive-compulsive disorder.

0.2.2 Experimental Design

0.2.2.1 General Setup

The model is based on data from two fear generalization experiments. These experiments involved various trials during which different levels of stimuli (circles of various diameters) were shown to a group of 40 participants. Conditioned stimulus (`CS+` or `CS-`) were associated with the presence (for `CS+`) or the absence (for `CS-`) of an unconditioned stimuli (US, an electric shock), while test stimulus (`TS`) were circles that varied slightly from the `CS(s)` in diameter size. None of the `TS` or `CS-` trials were paired with the US, and the experiments had an *acquisition phase* (learning trials) during which only `CS(s)` were shown and a *generalization phase* (generalization trials) during which also `TS` were shown mixed with the `CS(s)`. Such designs help to capture how subjects generalize their conditioned fear responses to new stimulus that are not exactly the same as the original stimuli.

0.2.2.2 Specifics of the Experiments

- **Experiment 1 Simple Conditioning:** Involved *simple fear conditioning* with one cue associated with the `US`. The *acquisition phase* comprised 14 `CS+` trials. The *generalization phase* comprised 4 blocks for a total of 174 trials, where each block, except the first one, started with 10 consecutive `CS+` trials, and comprised a total of 22 `CS+` and 24 `TS` trials. The *reinforcement rate* was 50%.

- **Experiment 2 Differential Conditioning:** Used a *differential fear conditioning* setup with two cues; one predicting the `US` and another indicating no US. The *acquisition phase* comprised 24 trials(12 `CS+`, 12 `CS-`). The *generalization phase* comprised 3 blocks for a total of 156 trials, all the blocks, except for the first one, started with 6 `CS+`

trials, and comprised a total of 14 `CS+`, 8 `CS-` and 32 `TS` trials. The *reinforcement rate* for the `CS+` was 83%.

Both experiments were structured to capture perceptual data and `US` expectancy ratings from participants during the *learning* and *generalization phases*, providing a rich dataset for modeling individual differences in generalization behavior.

0.2.3 Linking Neural Mechanisms to the Bayesian Mixture Model

The multilevel mixture model of Yu et al. (2023) separates *learning-rate* and *generalization-rate* parameters and clusters participants into four phenotypes (Non-Learners, Over-generalizers, Physical Generalizers, Perceptual Generalizers). Here we speculate how the positive and negative fear-generalization substrates described earlier may serve as the biological engines that drive those parameters and phenotypes.

0.2.3.1 Brain circuits most likely to modulate the *learning rate* during the *acquisition phase*

- **Basolateral Amygdala (BLA) & Locus Coeruleus (LC):** BLA activity, boosted by LC noradrenaline, governs the magnitude of cue-US updates; BLA-LC coupling therefore could have a part in encoding the learning curve, although these same regions are later involed in generalization decisions.

- **Ventral Tegmental Area (VTA):** dopaminergic prediction-error signals update both threat and safety values; VTA could accelerate or decelerate learning depending on the surprise of US omission.

- **Dorsal Striatum / Thalamo-striatal loop:** consolidates stimulus-response habits; rapid dorsal-striatal recruitment could appear as ”fast learning” in expectancy ratings.

- **Ventromedial Prefrontal Cortex (vmPFC):** exerts inhibitory control; excessive early vmPFC activation could blunt the effective learning rate, even when subcortical plasticity is intact.

0.2.3.2 Brain circuits most likely to modulate the *generalization rate* during the *generalization phase*

- **Anterior Hippocampus (pattern completion) vs. Dentate Gyrus (pattern separation):** during generalization trials, completion-biased dynamics widen the gradient, whereas separation-biased dynamics sharpen it.
- **Anterior Insula (AI) & dACC/dmPFC:** heighten interoceptive threat awareness and risk appraisal; stronger AI/dACC activity across similar pairings could translates into a steeper generalization rate.
- **Pulvinar $\rightarrow$ Amygdala "low-road":** coarse, fast visual routing favors generalization on the basis of gross physical similarity.
- **Default-Mode Network nodes (vmPFC, MTG, Angular Gyrus):** supply safety/value context; safety-valuation signals arrving during the generalization phase sharpen discrimination and thus lower the generalization rate.

0.2.3.3 Prossible neural profiles of the model's four participant groups

Non-Learners

- Hypo-activation of BLA, LC, and VTA $\Rightarrow$ weak associative strengths ancoding.
- Dominant hippocampal pattern separation and early vmPFC inhibition yield a flat, narrow gradient around both `CS+` and `CS-`.

Overgeneralizers

- Hyper-reactive LC-BLA bursts heightening the generalization.
- Sustained AI and dACC/dmPFC activity across the stimulus continuum drives a broad, steep generalization gradient.
- Completion-biased hippocampus plus weak DMN safety gating spread fear toward `CS-`-like stimulus *and* relief toward `CS+` omissions, giving a broad, bidirectional gradient.

Physical Generalizers

- Strong pulvinar-amygdala ”low-road” gating plus heightened visual-association cortex activity bias decisions toward external, metric similarity.
- Hippocampal completion is engaged only when physical overlap is large, producing a narrow but high-magnitude generalization curve around the `CS+`.

Perceptual Generalizers

- Greater reliance on internally generated stimulus templates (anterior hippocampus, vmPFC, MTG); completion can occur despite weak physical overlap.
- LC-hippocampal bursts triggered by imagery or expectancy amplify associative spread, while vmPFC safety gating lags, producing diffuse but moderate generalization.

0.2.4 Mathematical Formulations by Group

0.2.4.1 Introduction

To explain the model a coordinate system will be used, where **i** indicates participant i and **j** indicates trial j in the dataset. Furthermore a $\pm$ notation will be used to indicate those parameters which need to be computed for both `CS+` and `CS-` in case of a *differential conditioning* setup (like in experiment 2).

0.2.4.2 Latent Group Indicator ($\boldsymbol{m_i}$)

Most of the equations and priors of the model are conditioned on the discrete **Latent Group Indicator** $\boldsymbol{m_i}(m_i = 1, 2, 3, 4)$, which indicates the group membership for participant i:

$$m_i \sim \text{Multinomial}(1, \pi_1, \pi_2, \pi_3, \pi_4)$$

where $\pi_1, \pi_2, \pi_3, \pi_4 \sim \text{Dirichlet}(c(1, 1, 1, 1))$ with $\sum_{i=1}^{4} \pi_i = 1$ are the group probabilities.

The four latent groups, which correspond to the labels 1, 2, 3 and 4 respectively, are: *Non-Learners*, *Overgeneralizers*, *Physical Generalizers* and *Perceptual Generalizers*.

0.2.4.3 Learning Rate ($\boldsymbol{\alpha_i}$) and Generalization Rate ($\boldsymbol{\lambda_i}$)

The **Learning Rate $\boldsymbol{\alpha_i}$** describes how quickly an individual updates their understanding based on new information:

$$\alpha_i = \begin{cases} 0 & \text{for Non-Learners, who do not adapt their responses,} \\ \text{Beta}(a_\alpha, k_\alpha) & \text{for Overgeneralizers, Physical and Perceptual Generalizers,} \end{cases}$$

where $a_\alpha = \mu_\alpha \cdot k_\alpha$ and $b_\alpha = (1 - \mu_\alpha) \cdot k_\alpha$ with $\mu_\alpha \sim \text{Beta}(1, 1)$ and $k_\alpha \sim \text{Uniform}(1, 10)$.

The larger the *learning rate* is, the greater the amount of learning will be for a given prediction error.
The **Generalization Rate $\boldsymbol{\lambda_i}$** reflects the generalization propensity of an individual:

$$\lambda_i = \begin{cases} 0 & \text{for Non-Learners, indicating no generalization,} \\ \text{Normal}(\mu_\lambda, \sigma_\lambda^2) \text{ truncated at } [0, 0.0052] & \text{for Overgeneralizers, who tend to generalize broadly,} \\ \text{Normal}(\mu_\lambda, \sigma_\lambda^2) \text{ truncated at } [0.0052, \infty] & \text{for Physical and Perceptual Generalizers,} \end{cases}$$
$$\text{with } \mu_\lambda \sim \text{Normal}(0.1, 1) \text{ truncated at } [0, \infty] \text{ and } \sigma_\lambda \sim \text{Uniform}(10^{-9}, 1).$$

The *generalization rate* will be used as a decay parameter, so smaller values of λ_i indicate a stronger generalization propensity.

0.2.4.4 Stimulus Similarity ($\boldsymbol{s_{ij}^{\pm}}$) and Distance ($\boldsymbol{d_{ij}^{\pm}}$)

Stimulus Similarity $\boldsymbol{s_{ij}^{\pm}}$ and how it is perceived varies across groups:

$$s_{ij}^{\pm} = \begin{cases} 1 & \begin{array}{l}\text{for Non-Learners, who perceive all stimuli as equally similar,} \\ \text{or for Associative Strength } v_{ij}^{\pm}{=}0,\end{array} \\ e^{-\lambda_i \cdot d_{ij}^{\pm}} & \text{for other groups, where similarity decays exponentially with distance.} \end{cases}$$

where the **Stimulus Distance $\boldsymbol{d_{ij}^{\pm}}$** itself depends on the group:

$$d_{ij}^{\pm} = \begin{cases} |x^{\text{CS}\pm} - x_j^{\text{TS}}| & \begin{array}{l}\text{for Physical Generalizers and Overgeneralizers,} \\ \text{based on absolute physical distances,}\end{array} \\ |\tilde{x}_{i,1,\ldots,j}^{\text{CS}\pm} - \tilde{x}_{ij}^{\text{TS}}| & \text{for Perceptual Generalizers, based on absolute perceived differences,} \end{cases}$$
$$\text{where } x^{\text{CS}\pm} \text{ is the coordinate of the CS}\pm\text{, } x_j^{\text{TS}} \text{ is the coordinate of the TS on trial j,}$$
$$\tilde{x}_{i,1,\ldots,j}^{\text{CS}\pm} \text{is the perceived CS}\pm \text{ until trial j and } \tilde{x}_{i,j}^{\text{TS}} \text{ is the perceived TS at trial j.}$$

0.2.4.5 Associative Strength ($\boldsymbol{v_{ij}^{\pm}}$) and Generalized Associative Strength ($\boldsymbol{g_{ij}}$)

The **Associative Strength** $\boldsymbol{v_{ij}^{\pm}}$ is updated based on the Rescorla-Wagner rule, for which the *associative strength* of the CS± is determined by the individual's *learning rate* $\boldsymbol{\alpha_i}$ and the prediction error observed by the individual from the previous CS± presentation:

$$v_{i,j+1}^{\pm} = \begin{cases} 0 & \text{for Non-Learners, which do not learn,} \\ v_{i,j}^{\pm} + \alpha_i (r_{ij}^{\pm} - v_{ij}^{\pm}) \cdot k_{ij}^{\pm} & \text{for all other groups,} \end{cases}$$

where $r_{ij}^{+} \in \{0, 1\}$ for CS+ and $r_{ij}^{-} \in \{-1, 0\}$ for CS- shows trials outcomes (US or no US), and $k_{ij}^{\pm} \in \{0, 1\}$ indicates when updating is happening (learning only happens during CS± trials).

The **Generalized Associative Strength** $\boldsymbol{g_{ij}}$ represents the interaction between *associative strength* $\boldsymbol{v_{ij}^{\pm}}$ and *stimulus similarity* $\boldsymbol{s_{ij}^{\pm}}$, crucial for calculating `US` expectancy:

$$g_{ij} = \begin{cases} v_{ij}^{+} \cdot s_{ij}^{+} & \text{if only excitatory learning is involved (CS+ is the only CS),} \\ v_{ij}^{+} \cdot s_{ij}^{+} + v_{ij}^{-} \cdot s_{ij}^{-} & \begin{array}{l}\text{if both excitatory and inhibitory learning are involved} \\ \text{(both CS+ and CS- are present in the stimuli).}\end{array} \end{cases}$$

0.2.4.6 Baseline Response ($\boldsymbol{w_{0_i}}$) and Scaling Factor ($\boldsymbol{w_{1_i}}$)

The **Baseline Response** $\boldsymbol{w_{0_i}}$ will govern the response in absence of *associative strengths* $\boldsymbol{g_{ij}}$, this parameter is normally distributed as follows:

$$w_{0_i} \sim \text{Normal}(\mu_{w_0}, \sigma_{w_0}^2)$$

where $\mu_{w_0} \sim \text{Normal}(0, 10^2)$, and $\sigma_{w_0} \sim \text{Half-Cauchy}(0, 2)$.

The **Scaling Factor** $\boldsymbol{w_{1_i}}$ is a parameter that will govern the mapping between the *latent and observed responses*, and it is distributed as follows:

$$w_{1_i} \sim \text{Gamma}(a_{w_1}, b_{w_1})$$

where $a_{w_1} \sim \text{Half-Cauchy}(0, 2)$, and $b_{w_1} \sim \text{Half-Cauchy}(0, 2)$.

0.2.4.7 US Expectancy ($\boldsymbol{\theta_{ij}}$) and Observable Response ($\boldsymbol{y_{ij}}$)

The **US Expectancy** $\boldsymbol{\theta_{ij}}$ is modeled as:

$$\theta_{ij} = A + \frac{K - A}{1 + e^{-(w_{0_i} + w_{1_i} \cdot g_{ij})}}$$

with $A = 1$ and $K = 10$ parameters that scale and adjust the impact of generalized associative strength on US expectancy.

This logistic function models the probability of expecting the `US` based on the strength of the *generalized association*.
The **Observable Response ($\boldsymbol{y_{ij}}$)** is normally distributed around the expected `US` probability:

$$y_{ij} \sim \text{Normal}(\theta_{ij}, \sigma^2_{m_i})$$

where the parameter $\boldsymbol{\sigma^2_{m_i}}$ regulates the amount of response noise which depends on the group:

$$\sigma^2_{m_i} \sim \begin{cases} \text{Uniform}(1.5, 3) & \text{for Non-Learners, for which the final response is completely random,} \\ \text{Uniform}(10^{-9}, 1.5) & \text{for all other groups.} \end{cases}$$

0.3 Python Project

The primary objective of this project was to replicate the original `R` and JAGS workflow from the repository hosted at osf.io/sxjak/ entirely in Python. This involved:

1. **Translating `R` data-processing scripts to `Python`:** Original `R` code, which pivoted and preprocessed the experimental data, was partially reimplemented using `numpy` and `pandas` for data manipulation.

2. **Recreating `JAGS` models in `PyMC`:** The Bayesian models for fear generalization originally built in JAGS were adapted to PyMC to enable fully Python-based inference with modern MCMC techniques.

3. **Translating the `R` exploratory and diagnostic visualizations to `Python`:** Functions that produced data visualizations and results analysis plots in `R` were rewritten in Python, leveraging `matplotlib` and `seaborn`.

4. **Translating the `R` simulation of latent mechanisms to `Python`:** The illustrative `R` scripts, used to demonstrate the impact of models' latent mechanisms of fear generalization on observed behavior, were also translated to `Python`.

5. **Integrating additional tools:** Libraries such as `arviz` were used for model diagnostics, trace plots, and posterior predictive checks, ensuring a Python-driven analysis pipeline.

Overall, this translation preserves the fundamental statistical formulations and while taking advantage of Python's extensive data and modeling ecosystem. The following subsections elaborate on each step, beginning with the data preprocessing necessary to produce PyMC-ready input structures.

0.3.1 Data Preprocessing (`Data_Preprocessing/data_preprocessing.py`)

The already fully processed `.rds` dictionaries that were fed to JAGS in the original project implementation, couldnt be correctly loaded in Python and used as input for the PyMC models, therefore some preprocessing on the partially preprocessed `.rds` data files was needed in order to construct input dictionaries for PyMC. The data preprocessing code is largely a direct translation of the original data preprocessing `R` scripts into Python. To replicate the `R` pipeline's functionality, the following Python libraries were used:

- **pyreadr** to load the partially preprocessed `.rds` files.
- **pandas** to organize the data in `DataFrame` structures, pivot long data to wide format (and vice versa), and merge different tables.
- **numpy** to facilitate array-based computations, including vectorized operations for distance calculations.
- **pickle** to serialize and save the final `DataFrame` or `NumPy` array structures in `.pkl` files for later use in Bayesian modeling.

The two separate experiments, *simple conditioning* and *differential conditioning*, require slightly different preprocessing steps. However, both follow a common pattern:

1. Load `.rds` files containing experiments' long format datasets and minimally clean the data (e.g. convert data types, ensure correct columns names and indexing).
2. Select and pivot into wide format (with shape `(participants, trials)`) specific columns of the datasets (e.g. participant's trial's `US` expectancy, trial's `US` presence, trial's stimulus size, participant's perceived trial's stimulus size, experiment's `CS` physical size)
3. Compute *perceptual distance* and *physical distance* measures for each trial of each participant.

4. Extract or exclude "learning" trials as needed for modeling variants.

5. Package the resulting arrays (including `US` expectancy, number of participants, and so on) into dictionaries for PyMC analysis.

0.3.1.1 Experiment 1: Simple Conditioning

In this experiment, we track the participant's perceived size of the conditioned stimulus (`CS+`) across trials. A *moving average* of these perceptions is computed, and we then calculate the absolute difference between each trial's perception and that average (perceptual distance, `d_per`), as well as the difference between physical stimulus sizes and a reference size (physical distance, `d_phy`).

These distance arrays (`d_per` and `d_phy`) alongside `US` expectancy (`y`) are then organized, optionally including reinforcement (`r`) and indicator arrays (`k`). The function `data_input_s1` combines all arrays into a dictionary, exporting multiple versions that either exclude the initial "learning" trials or retain them.

0.3.1.2 Experiment 2: Differential Conditioning

Here, since the experiment has a differential conditioning setup, two separate `CS` types (`CS+` and `CS-`) are managed, producing four distance measures: `d_per_p`, `d_phy_p` (for `CS+`), and `d_per_m`, `d_phy_m` (for `CS-`). The arrays are then managed in `data_input_s2`, similarly producing a dictionary containing `US` expectancy (`y`), distances for each `CS` type, and optionally reinforcement (`r`), indicator arrays (`k`). Both with and without learning-trials variants of the dictionaries are pickled to disk.

0.3.1.3 Resulting Data Structures

In both experiments, the core outputs are Python dictionaries with the following keys:

- Number of participants (`Nparticipants`), number of trials (`Ntrials`), number of learning trials (`Nactrials`),
- Perceptual (`d_per`) and physical distance (`d_phy`) arrays for each `CS`,
- `US` expectancy values (`y`),

- (Optionally) reinforcement (`r`) and `CS` indicator arrays (`k`).

The resulting Python dictionaries are basically organized datafiles which can be of three types:

- **Generalization-only (G)**: excludes all learning trials, retaining only pure generalization data.
- **Learning + Generalization, non-continuous (LG)**: includes learning trials but treats them as independent events (no carry-over learning).
- **Learning + Generalization, continuous (CLG)**: includes learning trials with a continuous (cumulative) learning assumption.

All dictionaries are serialized to `.pkl` files for direct ingestion by PyMC, ensuring that multiple model hypotheses can be tested without rewriting the preprocessing pipeline.

0.3.1.4 Data Visualization of the Partially Preprocessed Data (`Data_Preprocessing/data_visualization.py`)

In addition to the core preprocessing, the study's original `R` data visualization code was translated into Python, creating `data_visualization.py`. This module relies on `seaborn` and `matplotlib.pyplot` to produce diagnostic and exploratory plots from the partially preprocessed data files of the *simple* and *differential* conditioning experiments (`Data_s1.pkl`, `Data_s2.pkl` in `Preprocessed_data/`). These plots provide insights into how participants learned (or did not learn) the associations, how they generalized beyond the conditioned stimuli, and how accurately they perceived stimulus sizes.

The key functions and their roles are described below:

`dv_lr_gr_plot`
: Generates an *overall* learning curve plot considering only the "learning trials" (14 for Experiment 1, 24 for Experiment 2) and focusing on relevant stimuli (`CS+` and `CS-`). The function then computes and plots the *average* `US` expectancy across participants (`mean_ac`), as well as each participant's individual mean trace in a lighter shade.

`dv_lr_indi_plot`
: Focuses on the *individual participant* level, produces a grid of subplots, where each subplot shows a specific participant's mean `US` expectancy over the learning trials, for `CS+` and `CS-` (if present) stimuli, enabling a direct comparison.

`dv_ge_gr_plot`
: Focuses on *generalization* beyond the learning trials. Once the code has filtered out the learning trials, it examines multiple stimulus levels (sizes) (`S4`, `S5`, `S6`, `CS+`, `S8`, `S9`, `S10` in Experiment 1; `CS+`, `S2`, `S3`, `S4`, `S5`, `S6`, `S7`, `S8`, `S9`, `CS-`, etc. in Experiment 2) to show how `US` expectancy changes as the stimulus differs from the conditioned one. It plots both an overall mean and the individual-participant means in lighter lines.

`dv_ge_indi_plot`
: Again, this is a more *individual-level* version of the generalization plots, drawing a multi-subplot layout (one per participant). Each participants' mean `US` expectancy (with error bars) is plotted across the various stimulus levels that appear after the acquisition phase. This clarifies which participants show broad generalization (high `US` expectancy for all similar stimuli) versus narrow generalization (steep drop-off in expectancy for stimuli increasingly different from the `CS+`).

`dv_pe_gr_plot`
: Visualizes the *perceived* sizes of the stimuli across all participants in aggregate, using *violin* and *scatter* plots. By sorting stimuli in ascending order of physical size and overlaying the distribution of perceived sizes, one can see overall accuracy or biases in perception. For instance, if the `CS+` is physically in the middle range but gets perceived as consistently larger, it might indicate a bias tied to threat expectancy.

`dv_pe_indi_plot`
: Explores perception *per-participant*, plotting for each individual the distribution of perceived stimulus sizes with box/violin plots and comparing them to the actual physical size (shown as a red line). This helps highlight whether certain participants have systematic over- or underestimation of sizes across the stimuli set.

`dv_cor_plot`
: Examines the *correlation* between perceived and physical sizes for each participant. In each subplot, a scatterplot shows each stimulus's physical size vs. the participant's perceived size; a regression line (via `seaborn.regplot`) indicates the overall trend. The function then calculates Pearson's `R` for each participant, annotating it on the plot. This is a direct measure of the degree to which that participant is accurate in discriminating bigger vs. smaller stimuli.

All these plotting functions store their output in `Plots/Data_Visualization/`. They do not affect the final PyMC modeling dictionaries. Instead, they provide a thorough descriptive understanding of:

- How learning progressed over the "acquisition" trials (`dv_lr_gr_plot, dv_lr_indi_plot`).
- How participants generalized responses to novel or intermediate stimuli (`dv_ge_gr_plot, dv_ge_indi_plot`).
- How accurately participants perceived stimuli (`dv_pe_gr_plot, dv_pe_indi_plot`).
- Whether perceived sizes correlate well with actual sizes (`dv_cor_plot`).

0.3.2 Translating the JAGS Models to PyMC (`models_definitions.py`)

In the original study, the authors proposed a unified Bayesian formulation for fear generalization that applies to both *simple conditioning* (one CS) and *differential conditioning* (two CSs). Although the data from each experiment differ (one vs. two conditioned stimuli), the paper's model description relies on a single set of priors and the same conceptual structure (latent groups, learning rates, generalization parameters, etc.). The JAGS code thus implemented a *common* prior configuration for both experiments. Our goal was to replicate these models in Python/PyMC while preserving that core statistical formulation.

0.3.2.1 Overview of Model Variants

All models share a core concept of multiple latent groups (e.g. overgeneralizers, physical or perceptual generalizers, non-learners). However, they differ along two main axes:

1. Whether there is a *learning* component (`L`) that updates associative strength over trials, or only a *generalization* component (`G`).
2. Whether we consider *physical* distance alone (`PHY`), or both *physical* and *perceptual* distances (denoted `2D` for two distance dimensions).

Hence, our naming convention (e.g. `LG2D`, `G2D`, `LGPHY`) succinctly indicates:

- **G2D**: A *generalization-only* model that uses *two* distances (physical and perceptual) and *three* latent groups (overgeneralizers, physical generalizers, perceptual generalizers).
- **LG2D**: A *learning + generalization* model with *two* distances and *four* latent groups (adding non-learners to the three in G2D).
- **LGPHY**: A *learning + generalization* model that focuses on *physical* distance only and retains *three* latent groups (non-learners, overgeneralizers, physical generalizers).

In addition, each model may be prefixed with:

- `model_1v_`: single-CS (danger CS+) variants.
- `model_2v_`: two-CS (danger CS+ and safety CS-) variants.

Thus, for instance, `model_1v_LG2D` is a single-CS, learning-generalization model that uses both physical and perceptual distances and has four latent groups, while `model_2v_LGPHY` is a two-CS, learning-generalization model with only physical distances and three latent groups.

0.3.2.2 Key Differences from JAGS to PyMC

1. Vectorization. JAGS typically processes each participant and trial via nested loops (as seen in the original code snippet below). In PyMC, we often apply vectorized operations over participants (`Nparticipants`) and trials (`Ntrials`). For instance, the expression:

```
for (i in 1:Nparticipants) {
  for (j in 1:Ntrials) {
    ...
    theta [i,j] <- A + (K-A) / (1 + exp (- (w0 [i] + w1 [i] * g [i,j
        ])))
    ...
  }
}
```

in JAGS becomes array-based arithmetic in PyMC, with `tensor` structures representing all participants/trials in one or two arrays:

```
...
theta = 1 + (10 - 1) / (1 + pm.math.exp(-(w0[:, None] + w1[:, None]
     * g)))
...
```

2. Using `scan` to Replace Nested Loops. Some aspects of the model particularly the *trial-by-trial* update of an associative strength `v` cannot be fully vectorized over the trial dimension. In JAGS, this logic appears as nested loops for participants and trials:

```
v[i, j+1] <- ifelse( ... ,
                    v[i,j] + alpha[i] * (r[i,j] - v[i,j]),
                    ...)
```

In PyMC, we replace these loops with `pytensor.scan`, which steps iteratively over a specified dimension while preserving the computational graph for gradient-based sampling. Since `scan` processes its `sequences` parameter along the `axis=0` dimension, we transpose the `k` and `r` arrays (originally `(Nparticipants, Ntrials)`) into shape `(Ntrials, Nparticipants)`. This ensures each scan iteration corresponds directly to a *single trial across all participants*, rather than scanning one participant at a time. The code snippet below illustrates:

```
def trial_update(k_t, r_t, v_t, alpha, gp):
    v_next = v_t + alpha * (r_t - v_t) * k_t
    return v_next

v_seq, _ = pm.scan(
    fn=trial_update,
    sequences=[k.T, r.T], # Transposed so each iteration is one
        trial
    outputs_info=v_init,
    non_sequences=alpha[None,:]
)
v = v_seq.squeeze(1).T
```

This approach avoids writing an explicit Python loop over trials (and/or participants), while retaining the stepwise, sequential nature of the learning update found in the original JAGS model.

3. Removing Non-Theoretical Truncations The JAGS code introduced numerous bounds like `T(1e-9, )` or `T(1, )` on parameters (e.g. α_1, α_μ, `w1_1`, `d_sigma`). However, the original *paper's* model formulation did *not* require these truncations researchers had added them in JAGS to prevent zero or negative draws in certain scale parameters. In practice, these bounds complicated the posterior geometry in PyMC, causing sampling pathologies. We thus removed such constraints unless they were explicitly stated in the theoretical description. Concretely:

- **Retained truncations**: λ_1, λ_2 remain `TruncatedNormal`, because the original theory required positive or bounded values for the generalization-rate parameters.
- **Removed or relaxed**: All other `T(...)` constraints on α_1, α_μ, `w1_a`, `w1_b`, `w1_1`, and `d_sigma`. None were mandated by the published statistical formulation, and their removal eased numeric issues in PyMC's HMC sampling.

4. Prior & parameterisation refinements. While replicating the JAGS priors verbatim is *possible*, several of them created hard $0 / \infty$ gradients that slowed down No-U-Turn Sampling, produced large tree depths and/or divergences. Below is listed every change that was eventually adopted together with the principled reason *why* it probably improved HMC geometry.

- **Gamma hyperparameters for the slope w_1: $w1_a, w1_b$.** The original JAGS used HalfCauchy(2) for both the *shape-rate* hyperparameters of $\Gamma(\alpha{=}w1_a, \beta{=}w1_b)$. In PyMC/NUTS these Cauchy tails frequently generate very large hyper-draws, which in turn yield extreme α/β combinations and sharp curvature in the Γ density. We replaced them with Half-Student-t($\nu{=}2, \sigma{=}2$). This keeps heavy tails (so large slopes remain a priori possible) but is less extreme than Cauchy, reducing occasional "blow-ups" in w_1 and improving adaptation. Empirically this removed divergences while leaving posterior flexibility intact.

- **Hierarchical baseline response w_0 (non-centred) and its scale.** We retain the non-centred parameterisation $w_{0,i} = \mu_{w_0} + \sigma_{w_0}\,\mathrm{raw}_i$ with $\mathrm{raw}_i \sim \mathcal{N}(0,1)$ to break the usual μ-σ funnel. In addition, we changed the group-level scale prior from HalfCauchy(2) to Half-Student-t(ν=2, σ=2). The relatively lighter tail trimmed the frequency of extremely large σ_{w_0}, improving NUTS geometry, removing divergences and increasing effective sample sizes.

- **Participant-specific response noise σ_i.** The JAGS model used two disjoint uniforms $[10^{-9}, 1.5]$ and $[1.5, 3]$ with a hard boundary at 1.5. Hard walls and piecewise densities tend to produce jagged gradients and the discontinuity at 1.5 also introduces awkward, non-smooth behaviour. In PyMC a smooth, participant-level hierarchy was used:

$$\sigma_{\text{mean,learners}} \sim \mathcal{N}(0.75,\ 0.2), \quad \sigma_{\text{mean,nonlearners}} \sim \mathcal{N}(2.25,\ 0.2),$$

$$\sigma_{\mu,i} = \begin{cases} \sigma_{\text{mean,nonlearners}}, & gp_i = 0 \\ \sigma_{\text{mean,learners}}, & gp_i \in \{1,2,3\} \end{cases}, \qquad \sigma_{\text{scatter}} \sim \text{HalfNormal}(1.5),$$

$$\sigma_i \sim \text{TruncatedNormal}\big(\mu = \sigma_{\mu,i},\ \sigma = \sigma_{\text{scatter}};\ \text{lower} = 10^{-9},\ \text{upper} = 3\big).$$

  This provided smooth, positive support on $(0, 3)$, partial pooling across participants (i), and no hard boundary at 1.5, while still differentiating between learners (Overgeneralizers, Physical Generalizers, Perceptual Generalizers) and Non-learners. Probably, because σ_i is the likelihood scale, in PyMC, removing hard boundaries and sharing information stabilised adaptation (fewer divergences), improved effective sample sizes, and yielded more coherent PPCs than the slab prior.

Practical note. The final choices above were arrived at via iterative prior-predictive, posterior-predictive, and convergence checks with small, isolated edits starting from the original JAGS formulation, with the goal of minimising divergences, keeping $\hat{R} \approx 1$, maximising ESS, and preserving reasonable PPCs. Along the way multiple alternatives were tested, such as: for λ (group and participant levels), non-centred parameterisations, soft-truncated hierarchies; for σ (likelihood scale), two-scatter variant, ordered centre/width parameterisations, two group-level priors with no per-participant pooling; for w_0, centred vs. non-centred hierarchies, HalfCauchy vs. Half-Student-t scales; for w_1, Gamma with $(w1_a, w1_b)$ hyperpriors vs. hierarchical log-normal and mean-sigma reparameterisations. It was also tried a marginalised mixture (integrating out *gp* via `pm.Mixture`) that just slowed sampling with no benefits. While some options improved pair-plot aesthetics (e.g., reduced boundary piling) they typically slowed sampling and/or hurt PPCs. The configuration reported here yielded the most favourable overall trade-off (ESS, $\hat{R}$, zero-divergences, and predictive fit) across both the single-CS and two-CS models. Consistent with the original paper and the original JAGS models, another guiding design goal was to keep *the same* prior families, parameterisations, and graphical model structure across the single-CS (simple conditioning) and two-CS (differential conditioning) variants, so any differences in fit stem from the data/likelihood rather than shifting priors.

5. Expressing conditional logic: `switch` vs. boolean masks. In the original JAGS code many deterministic relations (including the trial-wise learning update) were written with `ifelse`. Our first PyMC translation mirrored that via `pm.math.switch`/`pt.where`. We then refactored several of those into simple boolean-mask algebra (e.g.

$$\alpha_i \;=\; \mathbf{1}\{gp_i = 0\} \cdot 0 \;+\; \mathbf{1}\{gp_i \neq 0\} \cdot \alpha_{1,i}$$

). Functionally these forms are equivalent for the models, and modern PyTensor evaluates both branches of a `switch`/`where` during gradient steps anyway, so masks do not "avoid computing" inactive branches nor prevent domain errors in the masked path. In the runs this refactor had no consistent impact on posterior results and, at most, a modest reduction in graph size/compile time. The mask style was kept primarily for shape clarity (broadcasting) and readability, not because it was required for convergence.

0.3.2.3 Fitting the Models (`run_pymc.py`)

All models defined in `models_definitions.py` (e.g., `model_1v_LG2D`, `model_2v_LGPHY`, etc.) are sampled via the `run_pymc.py` script. This script is itself a Python translation of the *original R sampling code*, preserving the same overall workflow but adapted for PyMC's inference engine. It provides a function `sampling_fun`, which:

1. Dynamically imports the requested model from `models_definitions.py` (using `importlib`).
2. Instantiates and initializes the model with a given `datafile`.

3. Creates directories for storing the model's graph (`Model_graphs`) and sampler outputs (`Fitting_results`).

4. Executes `pm.sample(...)` with an appropriate combination of step methods:

   - **Discrete variables (latent group indicators)** sampled via `CategoricalGibbsMetropolis`.
   - **Continuous variables** sampled via `NUTS` (No-U-Turn Sampler).

   Alternative approaches such as Metropolis, Slice sampling, and even `pm.sample_smc` for Sequential Monte Carlo were tried. However, these methods often encountered stability issues, threw warnings, or produced prohibitively slow sampling for these high-dimensional models. Despite being gradient-based (and thus differing from the original JAGS implementation's mix of Gibbs, random-walk Metropolis, and slice sampling), `NUTS` ultimately delivered the most robust sampling performance in the PyMC setup.

5. Saves the resulting `arviz.InferenceData` (including posterior and posterior predictive samples) to disk for subsequent analysis.

By running this function for each desired model name (e.g., `model_1v_LG2D`, `model_2v_G2D`) and data set (simple or differential conditioning), the full sampling pipeline was replicated, much like in the original R scripts only now in Python, with PyMC's sampling mechanics.

Sampler settings and run lengths. All experimental data fits ($N_{\text{participants}} \approx 40$) were finally run with `chains` = 4, `tune` = 10,000, `draws` = 10,000, and `target_accept` = 0.97. For the simulated dataset described in 0.3.4.2 ($N_{\text{participants}} = 200$), in the last runs was used `chains` = 4, `tune` = 3,000, `draws` = 3,000 (`target_accept` = 0.97 as well). The larger simulated dataset increases the cost for NUTS, so a shorter runs were used due to memory constraints and practical reasons.

For reference, the original JAGS runs used `draws` = 25,000 after `burn-in` = 75,000. Because JAGS combines Gibbs, random-walk Metropolis, and slice updates, it typically requires more iterations to reach an ESS comparable to NUTS. By contrast, gradient-based NUTS often attains higher ESS per iteration, so fewer draws should suffice.

Longer PyMC runs on *earlier prototypes* of the models were also tested (`draws` = 20,000, `tune` = 30,000) for the experimental datasets. Posterior summaries, $\hat{R}$, ESS, and PPCs were essentially unchanged compared with `draws` = 10,000, `tune` = 10,000 (and, in many cases, `draws` = 5,000, `tune` = 5,000), while wall-clock time increased substantially. Hence, for the *final* models reported here, we adopted the settings above as a pragmatic balance of precision and compute.

That said, it is not impossible that *longer* PyMC runs (e.g. modestly increasing `draws` and `tune`, or `target_accept`) could yield small improvements, especially for parameters with weaker identifiability and in the differential-conditioning model.

0.3.3 Analysis

After fitting each PyMC model to the experimental data, we conducted a series of analyses to interpret and validate the model outcomes. The code for these analyses is largely a direct Python translation of the original `R` scripts from the study's repository (osf.io/sxjak/), ensuring comparable functionality and consistency with the researchers' prior work.

The analysis focuses on the results of the *full* models for both **simple conditioning** (`model_1v_LG2D`) and **differential conditioning** (`model_2v_LG2D`), each using the assumption of *continuous learning plus generalization "CLG"* dataset. These fitting results (e.g., `Results_Study1_CLG2D_nc` and `Results_Study2_CLG2D_nc`) serve as our main source for parameter estimates, posterior predictive checks, and latent group allocations.

The scripts leverage `arviz`, `matplotlib`, and `seaborn` to generate plots that are then saved to the `Plots` directory. The Python-based analysis includes the following steps:

- **Extracting latent group allocations** for each participant, based on posterior samples of the discrete group indicator parameter.
- **Evaluating convergence** using standard MCMC diagnostics (rank-normalized $\hat{R}$, effective sample sizes, trace plots).
- **Running posterior predictive checks** to determine how well the models reproduce observed data and highlight any systematic discrepancies.

- **Examining key estimated parameters** (learning rates, generalization rates, group probabilities) and visualizing them across participants.

- **Examining response patterns** (generalization, learning, perception, similarity) across trials, participants, and latent groups.

In the following, it is detailed each step of the analysis for the single-CS (*simple conditioning*) and two-CS (*differential conditioning*) datasets under the continuous-learning-plus-generalization (CLG) assumption.

0.3.3.1 Convergence Diagnostics (`Analysis/convergence.py`)

The `convergence.py` module performs posterior convergence checks for the Bayesian CLG2D fits of both studies (simple conditioning, differential conditioning). It loads precomputed `ArviZ InferenceData` files, prints numerical diagnostics ($\hat{R}$, bulk/tail ESS), counts divergences, and produces compact visual diagnostics (trace, density and pair plots) for priors and hyperpriors.

- **Inputs and artifacts:** netCDF files for Study 1 and Study 2 fits (`az.from_netcdf`), plus the experiments' metadata for labelling figures.

- **Numerical diagnostics:** per-experiment $\hat{R}$ and effective sample sizes (ESS) for the main priors and hyperpriors; reporting also includes mixture weights π, and participant level quantities $(\sigma, \alpha, \lambda, w_0, w_1)$.

- **Divergences:** the total number of NUTS divergences is printed.

- **Visual diagnostics:** for each parameter set, compact *trace+density* plots (alignment across chains, mixing) and *pair plots* (shape of joint posteriors, ridge structure) are saved.

- **Output location:** all figures are written to `Plots/Convergence/`.

Key Functionalities and Corresponding Functions.

`convergence_plots`
Given a dictionary of `InferenceData` objects and named parameter sets, this function iterates over experiments and draws:

- *Trace+density* panels for the selected priors and hyperpriors (compact layout, divergences marked).
- *Pair plots* for higher-level hyperpriors (including `sigma_mean_learners`, `sigma_mean_nonlearners`) to visualise posterior correlation structure and potential ridges.

Plots are saved with experiment-specific filenames.

Empirical Convergence Summary (Posterior).

- **Experiment 1 (simple conditioning):** Overall looks converged. The vast majority of monitored quantities have $\hat{R} \leq 1.01$ with bulk/tail ESS typically in the thousands. Trace plots for priors and hyperpriors show good chain overlap and stationarity; posteriors occupy reasonable ranges.
- **Experiment 2 (differential conditioning):** Convergence is weaker. Several monitored elements show $\hat{R} > 1.01$ and noticeably lower ESS. Trace plots for priors/hyperpriors are less well aligned across chains, indicating slower mixing and stronger posterior correlations, consistent with the harder geometry of the two-CS model.

Hyperprior Behaviour in Pair Plots. In both experiments, the pair plots of the hyperpriors reveal that the *uniform* hyperpriors tend to push mass against their bounds: α_κ accumulates near its *upper* bound and λ_μ near its *lower* bound. We tested replacing these with smooth, unbounded positives (e.g., log-normals or half-normals). This change *visibly improved* the pair plots (less boundary pile-up) but did *not* improve R-hat/ESS or posterior predictive checks; in some runs convergence was slightly worse. For this reason, and to preserve comparability with the original specification, the final analyses reported here retain the original uniform hyperpriors.

0.3.3.2 Latent Group Allocation (`Analysis/group_allocation.py`)

One of the primary analysis steps is to identify the most likely latent group for each participant based on posterior samples of the discrete group indicator (`gp`). The `group_allocation.py` module accomplishes this by:

1. Loading the final fitting results from `arviz.InferenceData` objects (stored as `.nc` files).
2. Extracting the MCMC samples of the group indicator variable.
3. Computing the proportion of samples in which each participant falls into each group.
4. Assigning a *dominant* group if its proportion exceeds a specified threshold (75%).
5. Generating bar plots to visualize the resulting allocations (including an "Unknown" label for participants who do not surpass the 75% criterion).

Implementation Highlights. The core functions in `group_allocation.py` replicate the original `R` scripts' logic in Python:

- `process_sample`: Collapses posterior samples for the latent group indicator across all chains/iterations, calculates per-participant group proportions, flags a dominant group if Proportion > 0.75, and outputs a `DataFrame`.
- `plot_gp_allocation`: Produces bar plots of group proportions versus participant number, including a dashed line at the 75% threshold.

Comparison with the original `R` and `JAGS` implementation. Although this module follows the same conceptual logic as the JAGS version, there are notable differences in the final allocations:

- **Experiment 1 (single CS)**: The PyMC-based results show a broad spread of *non-learners*, in part because the *simple conditioning* dataset should yields weaker learning compared to the *differential conditioning* one because of the presence of only a single conditioned stimulus CS^+. This does differ from the JAGS implementation's group allocations, which also identified many non-learners but to a somewhat larger degree and with slightly different allocations.

- **Experiment 2 (two CSs: CS+ and CS-)**: In contrast, these data promote stronger learning effects overall due to the presence of two conditioned stimuli, so we observe many *physical* and *perceptual generalizers* in the PyMC group allocation results. The original JAGS results for the second experiment tended to highlight an abundance of *perceptual generalizers* over physical generalizers, and very few non-learners, overgeneralizers and "unknown" particiapnts which do not reach the 75% threshold. The PyMC-based result still finds a high concentration in the generalizer groups, albeit with a somewhat different balance between physical and perceptual generalizers, doesn't find any non-learner and allocates more individuals in the "unknown" group (7 vs 4 of JAGS).

Such deviations are not unexpected given:

1. The PyMC-based implementation uses `NUTS` for continuous variables rather than JAGS's mix of Gibbs, random-walk Metropolis, and slice sampling.
2. Certain artificial truncations present in the JAGS code were removed, which can shift posterior modes.
3. The changes in priors and parameterisation that were applied to adapt the original model to the PyMC framework.

Despite these differences, the qualitative patterns e.g. more non-learners in a single-CS environment, more generalization and learning among participants in a two-CS environment largely replicate the core findings of the original study.

0.3.3.3 Posterior Predictive Checks (`Analysis/posterior_predictive_checks.py`)

An essential part of any Bayesian modeling workflow is assessing how well the model's predictions match the observed data. The `posterior_predictive_checks.py` module provides a range of plotting functions for these *posterior predictive checks*, mirroring the original `R` scripts in both logic and structure. The functions first randomly subsample from the posterior predictive distribution (`y_pre`) and then, for each relevant stimulus, visualize how closely the model-based predictions align with the actual data, under various grouping and conditioning scenarios.

Key Functionalities and Corresponding Functions.

`ppc_plot`
: Helper function that creates quantile- and mean-based comparisons of the *observed* vs. *predicted* data for a **single study**. It computes quantiles (e.g. 10%, 30%, 70%, 90%) and mean in both the predicted and observed data, allowing a check of whether the model captures the major distributional features per stimulus.

`ppc_plot_both`
: Extends `ppc_plot` to visualize **two studies** at once (e.g., single-CS vs. two-CS experiments). It arranges the subplots created by `ppc_plot` for each experiment side by side, creating an unified figure that is then saved to visually compare the two experiments.

`ppc_plot_gr`
: Helper function that focuses on **group-level** PPC by participant *dominant group*. It divides participants according to their assigned group and creates plots of quantiles for each group, stimulus by stimulus.

`ppc_plot_gr_both`
: Combines the logic of `ppc_plot_gr` with the dual-experiment approach of `ppc_plot_both`, enabling a **group-level** PPC comparison across both simple and differential conditioning experiments. It arranges the subplots for each group in each experiment created by `ppc_plot_gr`, then generates and saves a unified figure.

`ppc_plot_indi`
: Produces and saves a figure containing **participant-by-participant** series of mean-based plots. For each individual, it plots the average predicted value vs. the average observed US expectancy per stimulus, and shows the allocated group. This fine-grained check is useful for diagnosing any specific participant whose data deviate sharply from typical patterns or from the model's predictions.

Comparison with the original `R` and `JAGS` implementation.

- *General PPC:* Overall, our global predictive checks (`ppc_plot_both`) yield results very similar to those in the JAGS-based analysis, differing only slightly in some stimulus-specific quantiles.

- *Group PPC:* By contrast, the `ppc_plot_gr` graphs show more notable divergences, reflecting the fact that the PyMC approach sometimes allocates participants to different latent groups than the original JAGS approach, and in the case of **Experiment 2** (differential conditioning) doesn't allocate any participant in the non-leaners group. Consequently, the group-by-group traces do not align with JAGS's ones.

- *Individual PPC:* The participant-level checks (`ppc_plot_indi`) resemble the JAGS outputs for most individuals. In **Experiment 1** (simple conditioning), the PyMC version arguably yields slightly better fits for some participants' US expectancy (e.g. participant 10 and 24). Meanwhile, **Experiment 2** (differential conditioning) remains close overall, except that JAGS appears to better capture: responses to smaller stimuli (i.e. CS$^+$ and close stimuli for participants 10, 17, 20, 23, 32, 39), and overall responses of some specific individuals (21, 26, 29, 30, 37).

Overall, these posterior predictive checks confirm that the PyMC-based models perform slightly worse than JAGS on a global scale, while individual discrepancies arise probably from differences in sampling methods, prior differences, and subsequent group allocations. The final plots are all stored in the `Plots/Posterior Predictive Checks/` directory.

0.3.3.4 Parameter Estimation (`Analysis/parameter_estimation.py`)

The Bayesian models revolve around key parameters like the learning rate (α), generalization rate (λ), and group probabilities (π). To facilitate a deeper understanding of these parameters' distributions, once again by closely mirroring the original `R` analysis scripts, a `parameter_estimation.py` module was developed in Pyhton.

Key Functionalities and Corresponding Functions.

`pe_data_indi`

Processes single-study data at the **individual-participant level**. It extracts α and λ from the posterior, computes quantiles $(2.5\%, 25\%, 50\%, 75\%, 97.5\%)$, and merges these with the participant's group allocation, forming the basis for participant-level parameter plots.

`pe_plot_indi`
: Consumes the quantile-based DataFrame from `pe_data_indi`, generating scatter plots with error bars and a median point for each participant's parameter. Participants are colored by their dominant group, making it straightforward to see how each group's parameter estimates compare.

`pe_plot_indi_both`
: Extends `pe_plot_indi` to handle data from **both experiments** simultaneously. It arranges subplots for the simple-conditioning vs. differential-conditioning results, facilitating direct visual comparisons of how α and λ differ between the single-CS and two-CS setups by generating and saving an unified figure.

`pe_data_gr`
: Gathers "hyperparameters" data from the posterior of both studies, for the mean of the learning rate α_μ, mean of the generalization rate λ_μ, and group probabilities (π), then organizes these values in a format suitable for multi-study plotting.

`pe_plot_gr`
: Generates violin plots for the data organized by `pe_data_gr`. It creates and saves an unified figure, that reveals how each hyperparameter's distribution spreads across participants in the simple-conditioning (study 1) and differential-conditioning (study 2) scenarios.

Comparison with the Original `R` Implementation.

- *Computations and Plotting*: The Python code replicates the same calculations and processing used in the original `R` scripts, ensuring consistent analysis. The violin and strip plots closely mimic the `R` version, albeit with the `matplotlib/seaborn` aesthetics.
- *Differences*: As with other modules, participants' group assignments differ from JAGS, meaning that the parameter values in Python might not identically match the ones in `R`. Nonetheless, the overarching patterns (like the approximate location of α and λ) remain consistent with the original study's ones.

All parameter-estimation plots produced here are saved to `Plots/Parameter_Estimation/`.

0.3.3.5 Response Patterns (`Analysis/response_patterns.py`)

The `response_patterns.py` module primarily addresses how participants' behavior evolves over trials and across stimuli, focusing on a variety of *observed* and *simulated* measures:

- **Generalization data**: US expectancy across stimuli in the datasets.
- **Learning data**: observed US expectancy and associative strengths `v` adjustments over trials for conditioned stimuli `CS+` (and `CS-` in the differential conditioning).
- **Perception data**: perceptual stimuli distances from `CS+`.
- **Similarity data**: "similarity to CS+" measure produced by the interaction between participants' posterior λ generalization rate and perceptual or physical distances.

This module translates the original `R` code, producing multiple plots that help identify whether participants *learned*, how they *generalized* and *perceived* the stimuli.

Key Functionalities and Corresponding Functions.

`rp_process_data`
: Merges the raw experimental data with simulated participants' group allocations. It computes stimulus-wise and trial-wise statistics such as mean or standard deviation of US expectancy, perceived stimulus size and perceived stimulus distance from `CS+` for both individual participants and groups.

`ge_plot / ge_gr_plot`
: Generate and save figures visualizing generalization patterns in the *observed data* for both studies:

  - `ge_plot` focuses on overall US expectancy across stimuli.
  - `ge_gr_plot` further breaks out `ge_plot` by each participant's dominant group, allowing alook at how generalization differs between groups.

`lr_gr_plot`

Generates and saves a unified figure for both studies, simple conditioning (single-CS) and differential conditioning (two-CS), showing how US expectancy evolves over `trials` for `CS+` (and also `CS-` for the differential conditioning scenario), both at the group average level and individually.

`v_gr_plot`

Plots the **associative strengths v** (v^+ relative to `CS+` and v^- relative to `CS-`) adjustments over trials, based on participants' posterior $\boldsymbol{\alpha}$ **learning rate** values. It visualizes the strengths adjustments at the group average level and at the individual level, and saves a unified figure for both studies.

`per_gr_mean` & `per_gr_sd`

Two *helper functions* that individually plot a study's mean or standard deviation of perceived stimulus distance from `CS+` across stimuli, broken down by participants' dominant group. They allow for group-wise comparisons of either average distance or variability in perception.

`per_gr_plot`

Combines the outputs of `per_gr_mean` and `per_gr_sd` into a single figure, presenting both the *mean* and *standard deviation* of perceived distance from `CS+` for a single study.

`sim_gr_plot`

Computes and plots **stimulus similarity to CS+** $\boldsymbol{s^+} = e^{-\lambda \cdot \textbf{distance}}$, where $\boldsymbol{\lambda}$ are the participants' posterior **generalization rate** values, and the *distance* metric depends on which group an individual was allocated to:

- **Non-learners**: Hard-coded to a similarity of 1 (since they never update).
- **Overgeneralizers**: stimulus similarity to `CS+` depends on *physical distance* $s^+ = e^{-\lambda \cdot d^+_{phy}}$.
- **Physical generalizers**: stimulus similarity to `CS+` depends on *physical distance* $s^+ = e^{-\lambda \cdot d^+_{phy}}$.
- **Perceptual generalizers**: stimulus similarity to `CS+` depends on *perceptual distance* $s^+ = e^{-\lambda \cdot d^+_{per}}$.

The function generates and saves a unified figure for both studies. By plotting these similarities on the x-axis of the distances, we can see, in each conditioning scenario, how each group's theoretical "similarity" falls off with `CS+` distances.

Comparison with the original `R` and `JAGS` implementation.

- *Overall Patterns*: Despite some group-allocation differences (see Section 0.3.3.2), the primary *learning*, *generalization* and *similarity* patterns looks broadly similar to the original JAGS outputs (except for the Non-learners group of the second study to which no participants were allocated to by the PYMC model). For example in both the Python/PyMC and `R/JAGS` implementations:
  - **Non-Learners** remain zero in the associative strength plot for the first study.
  - **Overgeneralizers** show overlap between `CS+` and `CS-` response over trials in the learning plot of the differential conditioning study.
  - **Physical/Perceptual Generalizers** show great divergence between `CS+` and `CS-` response in the learning plot of the second study, and also a great difference between `v_plus` and `v_minus` associative strengths adjustments over trials.
  - **All four groups** show nearly identical "distance vs. similarity" patterns in `sim_gr_plot` as they do in the original `R/JAGS` scripts.
- *Minor Differences*: Because PyMC's sampling sometimes assigned participants to different groups than JAGS, the group-labeled lines in `ge_gr_plot` might shift. Nonetheless, the same broad trends emerge, e.g. a strong contrast in `CS+` vs. `CS-` for physical/perceptual generalizers, while overgeneralizers show minimal difference.

All response-pattern plots are stored in the `Plots/Response_Patterns/` directory. Despite some noticeable group-level allocation shifts, the `Python` and `R` code produce highly comparable pictures of how participants learn and generalize in both the single-CS and two-CS settings.

0.3.4 Simulation

In the original study, a simulation-based analysis was performed to illustrate how varying parameter values in the model (learning rates, generalization rates, perceptual variability, and logistic scaling parameters) can influence observed behaviors and affect model performance. The second part of the simulation uses Bayesian analysis and served to check how accurately the model could recover these "true" parameters values from synthetic data, and to examine how latent group misallocations might alter inference.

We replicate those simulations in Python, largely following the same logic as the `R+JAGS` scripts but using PyMC-style sampling and NumPy-based random generation for the pseudo-data.

0.3.4.1 Part 1: Illustrating Parameter Effects (`Simulation/simulation_1.py`)

The module `simulation_1.py` corresponds to the first part of the simulation study. Here, the focus is on how individual parameters (learning rate α, generalization rate λ, perceptual noise, etc.) shape the outcome patterns in a controlled synthetic dataset. Specifically:

Key Aims.

1. *Demonstrate* the influence of different parameter settings e.g. "high" vs. "low" α, "high" vs. "low" λ, or differing perceptual variability on trial-by-trial associative learning and stimulus generalization.

2. *Visualize* how logistic scaling with different `w0`/`w1` parameters affects the models's predicted response function.

Key Functionalities and Corresponding Functions.

`lg_fun`
: This "generative" function simulates a learning experiment with a user-specified parameters. It produces trial-level data: similarity measures and distances from `CS+`, whether reinforcement occurred, associative strengths, generalization gradients and final model-based responses. By tweaking α, λ, `persd` (perceptual standard deviation), and `w1`, `w0` scaling factors one can observe distinct patterns in the simulated data.

`sim_lr_plot`, `sim_lr_comp_plot`

These functions focus on *learning rate* comparisons:

- `sim_lr_plot` draws the associative strength across learning trials for given parameters setting, highlighting how quickly `v` rises when α is high vs. remains slow when α is low.
- `sim_lr_comp_plot` plots and saves the results of two α values side by side in a unified figure, making it easy to see how "fast learners" differ from "slow learners" in the same environment.

`sim_ge_plot`, `sim_ge_comp_plot`

These focus on *generalization rate* comparisons, showing how λ generalization rate influences the stimulus similarity to `CS+` $s^{+} = e^{-\lambda distance^{+}}$:

- `sim_ge_plot` visualizes a single dataset's predicted similarity across various stimuli sizes and trials.
- `sim_ge_comp_plot` contrasts "high λ" vs. "low λ" simulations by generating and saving an unified figure for both scenarios, highlighting how quickly similarity decays with distance for different participants.

`sim_pervar_comp_plot`

Demonstrates the effect of *perceptual variability* (`persd`): large vs. small standard deviations in perceived distance from `CS+`. By plotting simulated data for "high `persd`" and "low `persd`" in a single unified figure it shows how participants' perceived distance from `CS+` might scatter more widely in the high-variability scenario, leading to more unpredictable responding.

`sim_logit`

Illustrates the logistic transformation used within the models. Different sets of `w0` and `w1` parameters are passed in to see how the predicted output saturates, shifts, or scales with the internal simulated **generalized strength g** measure. It clarifies how logistic parameters shape the final "response" from the generalized associative strength.

Comparison with Original `R` Scripts. The overall logic-construct trial data, vary parameters, and generate plots of associative strengths and generalization remains the same as in the `R` code. The discrepancies that appear in the plots are caused by differences in Python and `R` random number generation.

All simulation plots for `simulation_1.py` are saved to the `Plots/Simulation/` directory. Although they may not match the exact same curves shapes of the original `R` code, they achieve the same goal: demonstrating how each model parameter influences *simulated* learning and generalization behavior.

0.3.4.2 Part 2: Parameter recovery with 200 synthetic participants (`Simulation/simulation_2.py`)

`simulation_2.py` is a *faithful translation* of the original study's `R` code of the second part of the simulation into Python. While `simulation_1.py` changed *one* parameter at a time, the secondsimulation reproduces the *full* recovery experiment reported in the original paper. Four latent groups are created, each with 50 "virtual" participants ($N = 4 \times 50 = 200$). For every participant the code draws a complete set of $(\alpha, \lambda, w_0, w_1, \sigma)$ values from the same group-specific generative rules used by the Bayesian models. The synthetic data follow exactly the trial-structure of the *first participant of Experiment 2*, so that the resulting data set can be analysed with the very same models that were designed to fit to the real differential-conditioning experimental dataset.

Key aims.

1. **Identifiability check** verify that the chosen model(s) can recover the "true" α and λ values when they are known by construction.
2. **Group-allocation sanity** confirm that the four latent groups (Non Learners, Overgeneralizers, Physical and Perceptual Generalizers) remain distinguishable in the recovered posterior.
3. **Benchmark for simplified models** compare recovery quality of the full CLG2D model with the two simplified alternatives (`LGPHY` and `G2D`).

Key Functionalities and Corresponding Functions.

`decay_plot`

A quick diagnostic that shows how different values of λ translate into similarity-decay curves. It also picks a "reasonable" range for λ ($\approx$ 0.002–0.30) so that similarity stays between 0.7 and 0.95 at the extreme `CS+` distances.

`extract_variable`

Helper that tiles the fixed stimulus order (and US schedule) from the real experiment dataset so each of the 200 simulated participants sees the *same* sequence.

`lg_fun`

The core generator: for every participant and trial it updates excitatory/inhibitory associative strengths (v_+ , v_-), computes physical/perceptual similarity, passes the total generalised strength g through the logistic mapping, and samples the noisy response y. Returns a long-format data frame with 200 participants x 200 trials.

`lr_plot`, `sim_plot`

Visual sanity checks: (i) associative-strength trajectories during learning; (ii) similarity to CS+ across physical and perceptual distances on generalization trials.

`ge_plot` & `ge_gr_plot`

Show overall and group specific generalization gradients of the synthetic US expectancy ratings.

`prepare_data_for_pymc`

Reshapes the simulated data into PyMC input dictionaries: one *with* learning (CLG) and one *without* learning (G).

`sampling_fun`

Called three times to fit `CLG2D` (full model) and the two simplified models (`LGPHY`, `G2D`) to the synthetic data; each run is saved as a NetCDF trace in `Fitting_results/`.

`recovery_plot`, `all_recovery_plot`, `vs_recovery_plot`

Build the "true vs inferred" scatter plots once per model and then side-by-side to quantify how faithfully λ and α values are recovered.

Parity with the original `R` simulation. All three simulation figures the learning (associative strengths), the similarity profile, and the generalized response gradients are visually *near-identical* to their original `R` counterparts. This close match supports that the Python translation reproduces the intended generative process and trial logic.

Group allocation on simulated data. When fitting the full `LG2D` model to the simulated dataset, the posterior group-allocation plot shows that *most* participants are reassigned to their generating group (with only a few mismatches), indicating that the four latent classes remain identifiable under the PyMC implementation.

Empirical recovery: PyMC vs. original JAGS. With the Python translations of `CLG2D`, `LGPHY`, and `G2D` now fitted, we compare λ recovery:

- **CLG2D (λ recovery):** the original JAGS fit places points *closer to the diagonal* than PyMC for essentially all participants, indicating better λ recovery in JAGS.
- **LGPHY (λ recovery):** as expected (no perceptual distances), JAGS recovers λ for Physical Generalizers, Overgeneralizers, and Non-Learners and not for the Perceptual Generalizers. The PyMC `LGPHY` fit shows the same qualitative pattern (perceptual generalizers are worse) but overall the recovered values sits further from the diagonal than JAGS.
- **G2D (λ recovery) and the JAGS bug:** the original study's JAGS `G2D` code contains a double-exponential term for the stimulus similarity,

  $$\exp\big(-\lambda[i]\cdot\exp(-\lambda[i]\cdot d)\big),$$

  which severely harms λ recovery in their G2D results. The PyMC `G2D` implementation corrects this, yielding λ recovery *similar to the PyMC `LG2D`* fit rather than the degraded JAGS `G2D` result from the original figure.

Outputs and reproducibility. All simulation figures are saved under `Plots/Simulation/Simulation_part_2/`; fitted traces are in `Fitting_results/`, and the PyMC input dictionaries in `PYMC_input_data/`. The traces can be inspected with `arviz.summary` and fed to the recovery plotting utilities to replicate all the analysis.

References

Webler, Ryan D. et al. (2021). "The neurobiology of human fear generalization: meta-analysis and working neural model." In: *Neuroscience & Biobehavioral Reviews* 128, pp. 421–436. DOI: 10.1016/j.neubiorev.2021.06.035. URL: https://www.sciencedirect.com/science/article/pii/S0149763421002773.

Yu, Kenny et al. (2023). "Humans display interindividual differences in the latent mechanisms underlying fear generalization behaviour." In: *Communications Psychology* 1.5. DOI: 10.1038/s44271-023-00005-0. URL: https://www.nature.com/articles/s44271-023-00005-0.

PyMC Discourse Forum (n.d.). URL: https://discourse.pymc.io/.

PyMC Development Team (2025). *PyMC Documentation*. Version 5.x. URL: https://www.pymc.io/projects/docs/en/stable/api.html.

Gelman, Andrew (2006). "Prior distributions for variance parameters in hierarchical models." In: *Bayesian Analysis* 1, pp. 515–534. DOI: 10.1214/06-BA117A. URL: https://projecteuclid.org/journals/bayesian-analysis/volume-1/issue-3/Prior-distributions-for-variance-parameters-in-hierarchical-models-comment-on/10.1214/06-BA117A.full.

OpenAI (n.d.). *ChatGPT*. URL: https://chat.openai.com/.

Plummer, Martyn (2017). *JAGS user manual*. Version 4.3.0. URL: https://people.stat.sc.edu/hansont/stat740/jags_user_manual.pdf.

Stan Development Team (2025). *Stan User's Guide*. Version 2.37. URL: https://mc-stan.org/docs/stan-users-guide/index.html.

Sourceforge Forum (n.d.). URL: https://sourceforge.net/.

Betancourt, Michael (2020). *Identity Crisis*. URL: https://betanalpha.github.io/assets/case_studies/identifiability.html#Acknowledgements.