Exploring the Limits of the Threshold Rule: Noise, Variability, and Collective Nest Choice

Efe Mirkan Guner
Intelligence in Animals and Machines
University of Sussex

November 2025

Contents

Abstract

Collective nest-site selection in ants has been proposed to arise from simple individual rules rather than complex comparison strategies. In this study, the threshold-based model developed by Robinson et al. (2011) was reproduced and extended to examine how uncertainty influences its performance. A Monte Carlo simulation was implemented in which agents moved probabilistically between a home nest, a poor nest and a good nest, sampling noisy estimates of nest quality and committing to a site once the perceived quality exceeded an internal acceptance threshold. Two sources of variability were manipulated, namely noise in perceived nest quality and heterogeneity in acceptance thresholds. The effects of these factors on collective accuracy, decision time and final nest choices were then evaluated. Accuracy declined as nest-quality noise increased, and all conditions converged to near-random performance under high noise. Decision times decreased with noise, reflecting premature acceptance rather than more efficient search. Threshold variability reduced accuracy when noise was low but provided limited resilience at intermediate noise levels. These results indicate that the threshold rule functions effectively only under relatively low levels of uncertainty and becomes increasingly fragile when perceptual estimates are highly variable, highlighting constraints on its explanatory scope.

1 Introduction

Collective decision-making in social insects has often been presented as an example of how complex group-level behaviour can emerge from simple individual rules. Robinson et al. (2011) showed that colonies of Temnothorax albipennis can select high-quality nest sites even when scouts do not explicitly compare options. Their model demonstrated that a basic threshold rule, combined with stochastic searching and repeated encounters, is sufficient to reproduce the colony's empirically observed accuracy. By treating nest choice as a Monte-Carlo process governed by quality assessments, discovery probabilities and individual acceptance thresholds, the authors provided a mechanistic explanation for an apparently sophisticated group outcome. This work challenged earlier assumptions that direct comparison or specialised cognitive abilities are required for reliable collective choices, and instead suggested that robust decision-making can arise from simple embodied agents operating under uncertainty.

Although the threshold model has been influential, its simplifications raise questions about how sensitive the predicted colony-level performance is to variability in the ants' perceptual and internal processes. Empirical studies have shown that assessment of nest quality can be noisy and that searching behaviour is shaped by situated factors such as spatial arrangement and movement constraints (E. J. Robinson et al. 2009). Moreover, work on animal cognition has emphasised that behaviour emerges from interactions between an agent's sensory limitations and its environment rather than from idealised information processing (Guest and Martin 2021; Shettleworth 2010). In the context of nest selection, this means that variability in how scouts perceive nest quality, as well as heterogeneity in their acceptance thresholds, may influence the dynamics of switching, commitment and recruitment. Understanding these influences is important both for interpreting the biological system and for evaluating the scope of simple threshold-based models.

Further modelling is therefore valuable, as it allows systematic manipulation of parameters that are difficult to control experimentally. Exploring how the Robinson model behaves under different levels of noise can test the robustness of the threshold mechanism, reveal conditions under which accuracy declines, and indicate whether the model reproduces the empirical resilience of ant colonies. Parameter sweeps are also consistent with the wider argument that computational modelling forces explicit reasoning about theoretical assumptions and their implications for behaviour (Guest and Martin 2021). By reproducing the threshold-based decision model and systematically varying both nest-quality noise and threshold variability, the present study investigates how uncertainty influences collective accuracy and decision dynamics. This approach allows an examination of whether the model's predictions remain robust when agents are treated as noisy and environmentally situated decision-makers with minimal embodiment, rather than as idealised samplers.

2 Methods

2.1 Model Overview

A stochastic Monte Carlo simulation was implemented to reproduce and extend the threshold-based decision model developed by Robinson et al. (2011). This type of modelling was selected because nest-site choice in *Temnothorax albipennis* involves uncertainty at both the sensory and behavioural levels. Individual scouts vary in how they perceive nest quality and differ in the thresholds they apply when deciding whether to commit to a site. A deterministic approach would be unable to capture the variability in decision times and final nest choices observed in empirical studies. Monte Carlo simulation, by contrast, allows repeated sampling of perceptual noise, heterogeneous thresholds and probabilistic movement through the environment, enabling an examination of how these sources of uncertainty influence collective-level accuracy.

The simulated environment consisted of three discrete nest states: the original home nest, which was assigned a quality of negative infinity so that it could never be selected; a poor nest of fixed quality four; and a good nest of fixed quality six. In all conditions, the simulation modelled five hundred independent ants acting as scouts.

2.2 Simulation Algorithm

Each agent began at the home nest. At every visited site, the perceived quality was generated by sampling from a normal distribution centred on the nest's true quality. The standard deviation of this distribution, denoted as the nest-quality noise parameter, controlled the amount of sensory uncertainty. For the home nest, this variance was set to zero to maintain its non-selectable role.

At the start of each simulation, every agent was assigned an internal acceptance threshold drawn from a normal distribution with mean 4.5 and a specified standard deviation. This threshold represented the minimum perceived quality required for commitment. When an agent visited a site, it compared the sampled perceived quality against its threshold. If the perceived quality met or exceeded the threshold, the site was accepted and the decision time was recorded. If the perceived quality fell below the threshold, the agent moved to a new site.

Site transitions were governed by a fixed 3×3 probability matrix:

$$P = \begin{pmatrix} 0.91 & 0.15 & 0.03 \\ 0.06 & 0.80 & 0.06 \\ 0.03 & 0.05 & 0.91 \end{pmatrix}$$

where column j represents the current site and row i the next site. Travel times between nests were drawn from normal distributions with means given by a corresponding matrix of travel times and standard deviations equal to one-fifth of the respective mean:

$$T_{mean} = \begin{pmatrix} 1 & 36 & 143 \\ 36 & 1 & 116 \\ 143 & 116 & 1 \end{pmatrix}$$

Travel time was constrained to a minimum of one time unit to avoid negative durations. The simulation for each ant continued until a nest was accepted or until one thousand movement steps were completed, preventing indefinite looping.

Algorithm 1: Simulation of a single ant in the threshold model

---

1. Initialise the agent at the home nest.
2. Sample an acceptance threshold from $\mathcal{N}(\mu, \sigma_{\text{threshold}})$.
3. **While** no nest has been accepted and the step limit has not been reached:
   - 3.1 Sample a perceived quality from $\mathcal{N}(Q_{\text{true}}, \sigma_{\text{quality}})$ for the current nest.
   - 3.2 **If** the perceived quality is greater than or equal to the threshold:
     - Accept the current nest and record the decision time.
   - **Else**:
     - Select the next nest using the transition probability matrix.
     - Sample a travel time and update the accumulated decision time.
   - **End if**
4. **End while**

2.3 Parameter Sweep and Experimental Design

The study examined how two distinct sources of uncertainty influenced collective accuracy and decision dynamics:

1. **Nest-quality noise ($\sigma_{\mathbf{quality}}$)**

   Six levels were tested: $0.1, 0.5, 1.0, 1.5, 3.0, 4.0$. These values determined the standard deviation of the perceived-quality distribution for the poor and good nests. Larger values represented stronger sensory noise and reduced reliability of quality assessment.

2. **Threshold variability ($\sigma_{\mathbf{threshold}}$)**

   Three levels were tested: $0.1, 0.6, 1.2$. These values controlled the standard deviation of the population-wide threshold distribution. Higher variability produced greater heterogeneity in the scouts' acceptance criteria.

For each combination of the two parameters, five hundred independent ants were simulated. This fully crossed design allowed an assessment of how sensory uncertainty and internal variability interact, and whether they exert independent or compounding effects on collective nest choice.

The mean acceptance threshold was set to 4.5 so that it lay midway between the poor nest (quality 4) and the good nest (quality 6), ensuring that low-noise decisions reflected biologically plausible discrimination between suboptimal and high-quality sites rather than trivial acceptance or rejection of both options.

2.4 Output Measures and Visualisation

Three summary measures were produced for each condition.

First, **decision accuracy** was calculated as the proportion of ants that accepted the good nest. This measure reflected the collective's ability to identify the higher-quality option under varying levels of uncertainty. Accuracy values were plotted against nest-quality noise for each threshold-variability condition to reveal changes in performance as noise increased.

Second, **mean decision time** was computed by averaging the total time accumulated by ants before accepting a site. This measure captured the dynamics of commitment and offered an indication of potential speed–accuracy trade-offs. Mean decision times were plotted against nest-quality noise to examine whether sensory uncertainty produced slower or faster decisions.

Finally, the **distribution of final nest selections** across the three nest types (Home, Poor, Good) was recorded. Stacked bar charts were generated to illustrate how the colony's consensus shifted as sensory noise increased and to identify conditions where poor-nest selection or failure to leave the home nest became more common.

All simulations were run using a fixed random seed to ensure reproducibility. Data processing and visualisation were carried out using functions in *PlotSummaryDataRobinson.py*, producing the accuracy, decision-time and selection-distribution plots used in the Results section.

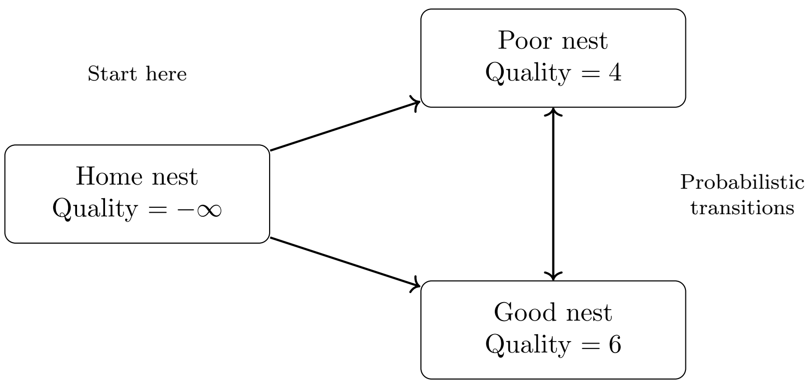

Figure 1: Schematic of the environment used in the Monte Carlo simulation. Agents begin at the home nest, which cannot be selected, and move probabilistically between the poor and good nests following the transition structure described by Robinson et al. (2011). Perceived quality at each visit is sampled with noise, and a nest is accepted once this perceived value exceeds the agent's internal threshold.

3 Results

3.1 Accuracy Across Noise Conditions

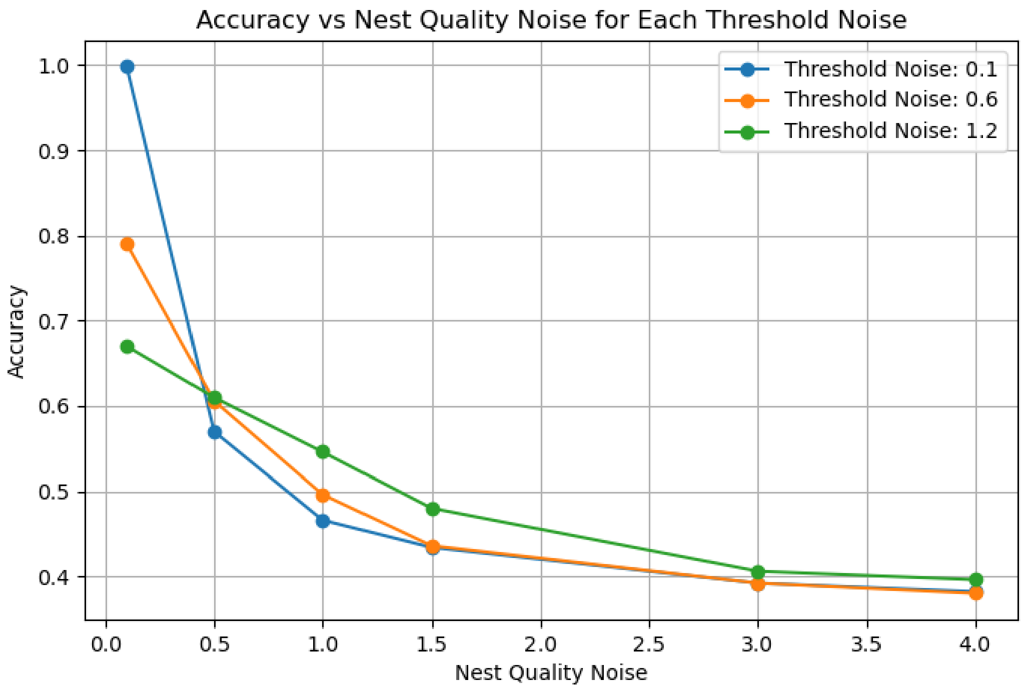

Figure 2: Accuracy across levels of nest-quality noise for different threshold variabilities. Accuracy decreased with increasing noise, and all conditions converged at high noise.

The effect of increasing nest-quality noise on accuracy was examined across three levels of threshold variability **Figure 2**. Accuracy declined in all conditions as nest-quality noise σ_{quality} increased, because higher noise made the perceived qualities of the two nests overlap more often. Under minimal noise $\sigma_{\text{quality}} = 0.1$, the lowest threshold variability condition $\sigma_{\text{threshold}} = 0.1$ achieved accuracy close to one, reproducing the near-ceiling accuracy reported in the original Robinson et al. (2011) model. In contrast, the highest variability condition $\sigma_{\text{threshold}} = 1.2$ began with substantially lower accuracy, since its wider distribution of thresholds included many agents whose acceptance criteria did not align well with the true nest qualities.

As nest-quality noise increased, accuracy in the low-variability condition declined more sharply than in the high-variability condition. This occurred because homogeneous thresholds were more vulnerable to misleading perceived-quality samples, whereas heterogeneity provided partial protection by maintaining some thresholds that remained effective despite degraded quality assessments.

At the highest level of nest-quality noise $\sigma_{\text{quality}} = 4.0$, accuracy converged to around 0.40 for all threshold conditions. This reflects that the increased variability in perceived quality effectively masked the difference between the nests, resulting in behaviour close to random choice. The convergence indicates that threshold variability had little influence once nest-quality noise became the dominant limiting factor.

3.2 Decision Time Under Increasing Noise

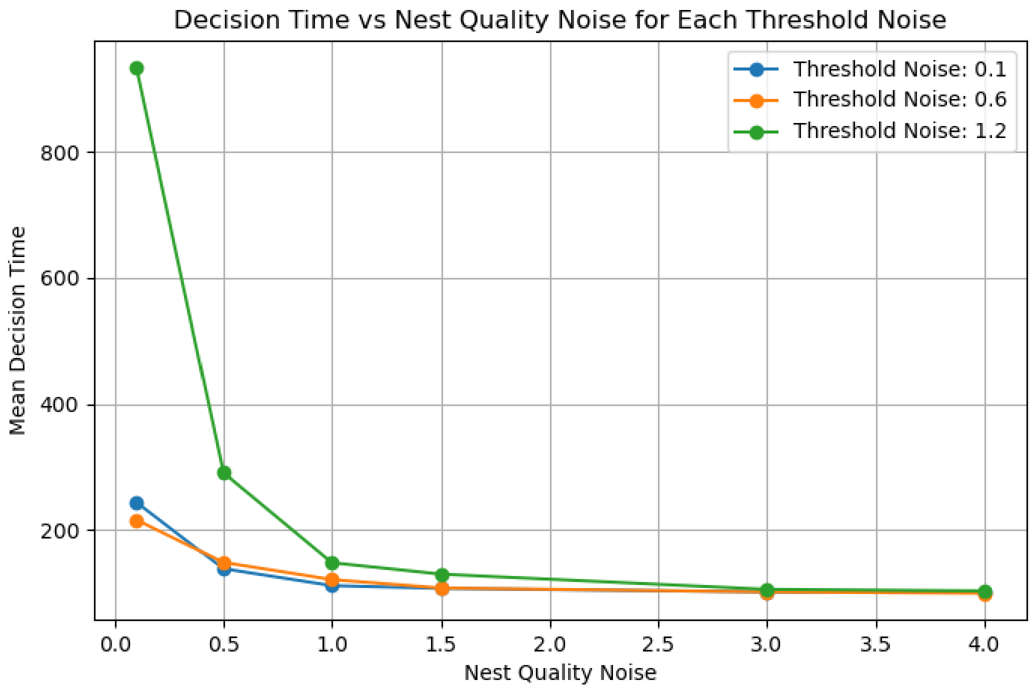

Figure 3: Mean decision time declined with increasing nest-quality noise $\sigma_{quality}$. High threshold variability $\sigma_{threshold} = 1.2$ produced long decision times at low noise, while all conditions converged at higher noise due to earlier, noise-driven acceptances.

Mean decision time decreased as nest-quality noise σ_{quality} increased across all levels of threshold variability **Figure 3**. When nest-quality noise was minimal, decisions required more time, particularly in the high threshold variability condition $\sigma_{\text{threshold}} = 1.2$. In this case, mean decision time exceeded 900 time units at $\sigma_{\text{quality}} = 0.1$, consistent with the prolonged decision latencies reported by Robinson et al. (2011) when thresholds were high relative to available nest quality. These prolonged searches resulted in slow convergence and occasional cases where no decision was made within the step limit.

As nest-quality noise increased, decision time fell sharply in all threshold conditions. This occurred because higher noise produced larger fluctuations in perceived quality, including occasional positive deviations that made a nest appear to exceed the threshold earlier in the search. Such deviations led to earlier commitments, even when the chosen nest was not the optimal one. The reduction in decision time therefore, reflected premature acceptance driven by misestimation, rather than more efficient evaluation.

Agents in the low threshold variability condition $\sigma_{\text{threshold}} = 0.1$ showed shorter decision times, as their thresholds were generally low enough to be met by the true qualities of the candidate nests. Their decline in decision time with increasing noise followed the same pattern as the other conditions, though with less pronounced changes. Overall, decision latency was shaped by the combination of threshold height and the magnitude of nest-quality noise, with high noise accelerating commitment while simultaneously reducing accuracy.

3.3 Distribution of Final Nest Choices

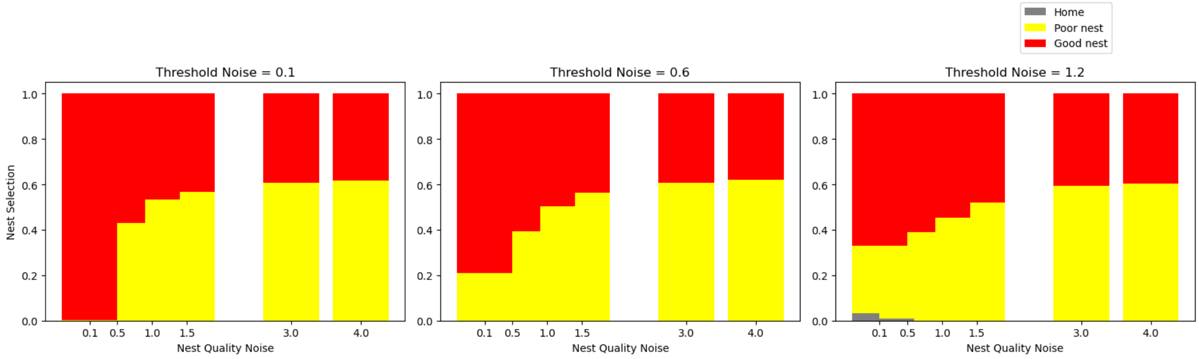

Figure 4: Final nest choices under different nest-quality noise levels. Increased noise led to more poor-nest selections, while rare home-nest outcomes occurred only at low noise with high threshold variability.

As in Robinson et al. (2011), the distribution of accepted nests reflected reliable good-nest selection at low noise. However, the outcomes of the decision shifted systematically as nest-quality noise increased **Figure 4**.. At low noise, most agents selected the good nest, as shown by the dominance of the red bars across all conditions. As nest-quality noise increased, the proportion of poor nest selections rose steadily, demonstrating that many agents were unable to reliably distinguish between the two candidate nests when perceived-quality samples fluctuated widely.

A small proportion of agents selected the home nest at the lowest noise level in the high variability condition $\sigma_{threshold} = 1.2$, $\sigma_{quality} = 0.1$. This occurred because some agents sampled thresholds so high that neither candidate nest produced a perceived-quality sample exceeding the acceptance criterion before the step limit was reached. As nest-quality noise increased, these non-selections disappeared, since noise created large positive fluctuations in perceived quality that pushed even very high thresholds to be exceeded. However, these early acceptances were often made for the poor nest, which contributed to the decline in accuracy described in Section 3.1.

The distribution of final choices therefore reflected the combined influence of threshold heterogeneity and nest-quality noise. High threshold variability increased the likelihood of no decision in low-noise settings, whereas high nest-quality noise increased the likelihood of incorrect decisions in all settings.

3.4 Summary of Patterns

Across all simulations, nest-quality noise exerted the strongest influence on decision outcomes. Accuracy declined steadily as noise increased, and decision times decreased as premature acceptances became more frequent. Threshold variability had a noticeable effect when noise was low, reducing accuracy and increasing the likelihood of prolonged searches, yet it provided some resilience at intermediate noise levels by maintaining diversity in acceptance criteria. At high noise, both accuracy and decision distributions converged across threshold conditions, indicating that nest-quality noise overwhelmed the threshold mechanism. Overall, the results indicate that the threshold rule described by Robinson et al. (2011) performs reliably only under relatively low uncertainty and becomes increasingly fragile as variability in perceived nest quality increases.

4 Discussion

The simulation examined how nest-quality noise and threshold variability shape the reliability of threshold-based decision-making. Accuracy was highly sensitive to variability in perceived nest quality, declining steadily as noise increased. Under low-noise conditions, colonies with low threshold variability performed well, supporting the claim by Robinson et al. (2011) that a simple acceptance rule can generate accurate collective decisions. However, as noise increased, accuracy declined sharply and all conditions converged at near-random performance, indicating that the threshold mechanism becomes fragile when perceptual uncertainty is high.

Analysis of decision times showed that increased noise produced earlier commitments rather than slower searches. Large positive fluctuations in perceived quality allowed thresholds to be exceeded prematurely, leading to shorter decision times that often reflected false positives rather than improved evaluation. This finding demonstrates how a mechanism that functions reliably under low noise can generate systematic errors when sensory estimates become highly variable.

The distribution of accepted nests further illustrated how errors emerged under uncertainty. Rising noise caused a shift from consistent selection of the good nest toward frequent acceptance of the poor nest. High threshold variability introduced additional failures at low noise, where some agents held thresholds too high to be exceeded, but it offered limited robustness at intermediate noise levels by preventing uniform failure across the population.

Overall, these results suggest that the threshold model explains accurate collective choices only under relatively low uncertainty. This aligns with broader arguments in comparative cognition that behaviour reflects interactions between sensory limitations and environmental structure rather than idealised optimisation (Guest and Martin 2021; Shettleworth 2010). The present analysis therefore extends Robinson et al.'s account by identifying clear limits to the robustness of the threshold rule under substantial noise.

A limitation of the present implementation is the absence of explicit spatial structure or recruitment dynamics. Transitions between nests are governed by a fixed probability matrix rather than by spatial distances, trail formation, or social interactions. Consequently, the model captures independent threshold-based sampling but does not represent the positive feedback mechanisms, such as tandem running, that drive emergent colony-level coordination in real Temnothorax emigrations. Future work could incorporate spatially embedded movement and recruitment to examine how positive feedback interacts with perceptual noise and threshold variability during collective nest-site selection.

References

Guest, Olivia and Andrea E. Martin (2021). "How computational modeling can force theory building in psychological science". In: *Perspectives on Psychological Science* 16.4, pp. 789–802. DOI: 10.1177/1745691620970585.

Robinson, Elva J.H. et al. (July 22, 2009). "Do ants make direct comparisons?" In: *Proceedings of the Royal Society B: Biological Sciences* 276.1667, pp. 2635–2641. ISSN: 0962-8452, 1471-2954. DOI: 10.1098/rspb.2009.0350. URL: https://royalsocietypublishing.org/doi/10.1098/rspb.2009.0350 (visited on 11/26/2025).

Robinson et al. (May 24, 2011). "A Simple Threshold Rule Is Sufficient to Explain Sophisticated Collective Decision-Making". In: *PLoS ONE* 6.5. Ed. by Frederick R. Adler, e19981. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0019981. URL: https://dx.plos.org/10.1371/journal.pone.0019981 (visited on 11/26/2025).

Shettleworth, Sara J. (2010). "Clever animals and killjoy explanations in comparative psychology". In: *Trends in Cognitive Sciences* 14.11, pp. 477–481. DOI: 10.1016/j.tics.2010.07.002.

A Program Code

main.py

```
import numpy as np
import PlotSummaryDataRobinson as psdr
import RobinsonCode as rc
import os
import yaml


def main():
    results = []

    # Plot folder
    plot_folder = "plots"
    os.makedirs(plot_folder, exist_ok=True)

    # Read Parameters
    with open("params.yaml", "r") as f:
        params = yaml.safe_load(f)

    qual_stddev = params["qual_stddev"]
    threshold_stddev = params["threshold_stddev"]
    n = params["n"]
    threshold_mean = params["threshold_mean"]


    for th_noise in threshold_stddev:

        for qual_noise in qual_stddev:

            print(f"Nest Quality Noise={qual_noise}, Threshold Noise={th_noise}")

            np.random.seed(42) # Set random.seed for reproducible runs

            # Run RObinsonCode and take accuracy, mean_time, accepts values
            accuracy, mean_time, accepts = run_robinson(qual_val=qual_noise,n=n,
                                                        threshold_mean=threshold_mean, threshold_stddev=th_noise)

            # Append values in results
            results.append((qual_noise, th_noise, accuracy, mean_time, accepts))


    # Accuracy vs Noise Plot
    accuracy_plot_path = os.path.join(plot_folder, "accuracy_vs_noise.png")
    acc_results = [(r[0], r[1], r[2]) for r in results]  # qual_noise, th_noise, accuracy
    psdr.PlotAccuracy(acc_results, threshold_stddev, accuracy_plot_path)

    # Decision time vs Noise Plot
    decision_time_plot_path = os.path.join(plot_folder, "decision_time_vs_noise.png")
    time_results = [(r[0], r[1], r[3]) for r in results] # qual_noise, th_noise, mean_time
    psdr.PlotDecisionTime(time_results, threshold_stddev, decision_time_plot_path)
```

```python

    distribution_plot_path = os.path.join(plot_folder, "selection_distribution.png")
    psdr.PlotSelectionDistribution(results, threshold_stddev,
        distribution_plot_path)


def run_robinson(qual_val, n, threshold_mean, threshold_stddev):

    # probabilities of visiting each site from each other
    probs = np.array([[0.91, 0.15, 0.03], [0.06, 0.80, 0.06], [0.03, 0.05,
        0.91]])

    # mean time to get between each nest
    time_means = np.array([[1, 36, 143], [36, 1, 116], [143, 116, 1]])

    # standard deviation of time to get between each nest
    time_stddevs = time_means / 5

    # mean quality of each nest. Note home is -infinity so it never gets picked
    quals = np.array([-np.inf, 4, 6])

    # standard deviation of quality: essentially this controls
    # how variable the ants assessment of each nest is.
    qual_stddev = np.array([0.0, qual_val, qual_val]) # set home nest
        qual_stddev=0 because quality already -np.inf

    # Run RobinsonCode function
    (accuracy, mean_decision_time, accepts) = rc.RobinsonCode(n, quals, probs,
        threshold_mean, threshold_stddev,
                                                              qual_stddev,
                                                                  time_means,
                                                                  time_stddevs)

    # Take returns of accuracy, mean_decision_time, accepts
    return accuracy, mean_decision_time, accepts

if __name__ == "__main__":
    main()
```

RobinsonCode.py

```python
import numpy as np


def RobinsonCode(n, quals, probs, threshold_mean, threshold_stddev, qual_stddev
    , time_means, time_stddevs):
    #   n = number of replicates (>=1)
    #   quals = row vector of m site qualities
    #           (quals(1) = home site
    #           quality: -Inf for no effect of home site quality on searching)
    #   discovery_probabilities = m * m matrix of discovery probabilities from
    #     column site to row site (N.B. columns should sum to 1)

    #   threshold_mean: mean population threshold for site acceptability
    #   threshold_stddev: standard deviation in population thresholds

    #   qual_stddev: standard deviation in quality assessments: **AOP**
    #   time_means: m * m matrix of mean travel times from column site to row
    #     site (N.B. should probably be symmetric)
    #   time_stddevs: m * m matrix of travel time standard deviations, from
    #     column site to row site (N.B. should probably be symmetric)
```

```python

    #   quora: 1 * m matrix of quorum times for each nest site    #
    #   times = row vector of times to first recruitment (i.e. nest acceptance)
    #   discovers = matrix (m x i) of times of first visit to each site
    #   visits = matrix (m x i) of numbers of visits to each site
    #   accepts = row vector of ids of accepted sites (indexed from 1 (for home nest) to m)
    #   the equivalents prefixed 'preq' are the pre-quorum equivalents of these

    # MATLAB allows variables to be created (and to grow) implicitly when elements are assigned
    # - Python doesn't play that foolish game, so we have to assign some variables here which we didn't in MATLAB
    nestNum = probs.shape[0]
    accepts = np.zeros([n], dtype=int)
    current_time = np.zeros([n])
    discovers = np.zeros([nestNum, n])
    visits = np.zeros([nestNum, n])

    Ants = []
    for i in range(n):
        ant = {'path': [],
               't': [],
               'thresh': 0,
               'selected': 0}
        Ants.append(ant)

    # corresponds to line 27 in m-code
    # Set the maximum number of steps for each ant
    Max_num_steps = 1000

    # corresponds to line 34 in m-code
    for i in range(n): # note that Python indexing is from 0, whereas MATLAB is from 1

        # Monte Carlo simulation of one ant
        # this sets up the output variables **AOP**

        # This holds the time it has taken before an ant has made the first
        # recruitment **AOP**
        current_time[i] = 0

        # this is a variable which holds where ant i currently is **AOP**
        accepts[i] = 0 # ant starts in home site

        # this is a matrix which holds the time at which the ant discovers
        # sites 1 to N where N= number of sites;
        # As ant is currently in the home site it never 'discovers' it
        # so set this (arbitrarily) to -1 **AOP**
        discovers[0,i] = -1 # ant is already in home site

        # this is a matrix which holds the number of times an ant visits
        # sites 1 to number of sites. As it is already in the home site the
        # 1st element is set to 1 **AOP**
        visits[0,i] = 1 # ant is already in home site

        # initialise the variables for the other home sites. it hasn't been to
        # any of the others so the time to 1st discovery is 0 and the number
        # of visits = 0  **AOP**
        for j in range(1, nestNum): # could be a problem here?
            discovers[j,i] = 0 # ant has not discovered or visited other sites
            visits[j,i] = 0
            # preqdiscovers[j,i] = 0;
```

```
            # preqvisits[j,i] = 0;

        # sample and set the ant's acceptance threshold **AOP**
        thresh = threshold_stddev * np.random.randn() + threshold_mean

        # set up some output variables *** AOP
        num_step = 0
        Ants[i]['path'].append(accepts[i])
        Ants[i]['t'].append(current_time[i])
        Ants[i]['thresh'] = thresh
        Ants[i]['selected'] = 0


        # corresponds to line 87 in m-code
        # this is now the main loop of the program. Essentially it says:
        # 1. for the current site, check to see if the ant accepts it based on
        #    it's threshold and a randomly selected quality based on the quality
        #    of the site **AOP**
        # 2. Do this until a site is accepted
        while Ants[i]['selected'] == 0:

            # check the quality of the current nest
            perceivedQuality = qual_stddev[accepts[i]] * np.random.randn() + quals[accepts[i]]

            # if the perceived nest quality is above the threshold, select it
            if perceivedQuality >= Ants[i]['thresh']:
                Ants[i]['selected'] = 1
                break


            # if you have exceeded the max number of steps without stopping
            # break out of the algorithm
            if num_step > Max_num_steps:
                Ants[i]['selected'] = 0
                break

            # probablistically pick one of the new sites to go to
            # unifrnd(0,1) generates a uniformly distributed number
            # between 0 and 1
            ran = np.random.uniform()
            # this then does the site picking. Looks complicated but is standard
            # and it works
            newsite = 0 # NOTE: THIS IS 0 INSTEAD OF 1 DUE TO PYTHON INDEXING!
            while ran > probs[newsite, accepts[i]]:
               ran = ran - probs[newsite, accepts[i]]
               newsite = newsite + 1

            # update the time taken with normally-distributed time-step size
            # (>=1) **AOP**
            delta = max(1, time_stddevs[newsite, accepts[i]] * np.random.randn() + time_means[newsite, accepts[i]])
            current_time[i] = current_time[i] + delta

            # update ant's current site, accepts, **AOP**
            # discovers, and the number of times it has been visited, visits **AOP**
            accepts[i] = newsite

            # if it hasn't discovered this site before, update the time that it
            # 1st discovered it, in discovers **AOP**
```

```
            if discovers[newsite, i] == 0:
                discovers[newsite, i] = current_time[i]

            # update the number of times it has visited this site **AOP**
            visits[newsite, i] = visits[newsite, i] + 1

            # Update the output variables **AOP**
            num_step = num_step + 1
            Ants[i]['path'].append(accepts[i])
            Ants[i]['t'].append(current_time[i])

        # corresponds to line 176 in m-code
        # record number of steps taken
        Ants[i]['numSteps'] = num_step

    # Calculate mean decision time
    mean_decision_time = float(np.mean(current_time))

    # Calculate accuracy by ants chose the good nest
    accuracy = np.sum(accepts == 2) / len(accepts)

    # Take returns of accuracy, mean_decision_time, accepts
    return accuracy, mean_decision_time, accepts
```

PlotSummaryDataRobinson.py

```
import matplotlib.pyplot as plt
import numpy as np

def PlotAccuracy(results, threshold_stddev, out_path):

    # Convert results into numpy array
    data = np.array(results, dtype=[('qual_stddev', float), ('threshold_stddev', float), ('accuracy', float)])

    # Create plot
    plt.figure(figsize=(8, 5))

    # Iterate over each threshold standard deviation
    for th_noise in threshold_stddev:

        # Pick the rows where the threshold matches th_noise
        mask = (data['threshold_stddev'] == th_noise)
        subset = data[mask]

        # Sort by increasing noise
        order = np.argsort(subset['qual_stddev'])

        # Plot accuracy vs Nest Quality Noise for each threshold
        plt.plot(subset['qual_stddev'][order], subset['accuracy'][order],
                 marker='o', label=f"Threshold Noise: {th_noise}")

    plt.xlabel("Nest Quality Noise") # label x-axis
    plt.ylabel("Accuracy") # label y-axis
    plt.title("Accuracy vs Nest Quality Noise for Each Threshold Noise") # set title
    plt.grid(True) # add grid
    plt.legend() # add lagend
    plt.savefig(out_path) # save figure the folder
    plt.close()

def PlotDecisionTime(results, threshold_stddev, out_path):
```

```python

    # Convert results into numpy array
    data = np.array(results, dtype=[('qual_stddev', float), ('threshold_stddev'
        , float), ('mean_time', float)])

    # Create plot
    plt.figure(figsize=(8, 5))

    # Iterate over each threshold standard deviation
    for th_noise in threshold_stddev:

        # Pick the rows where the threshold matches th_noise
        mask = (data['threshold_stddev'] == th_noise)
        subset = data[mask]

        # Sort by increasing noise
        order = np.argsort(subset['qual_stddev'])

        # Plot mean decision time vs Nest Quality Noise for each threshold
        plt.plot(subset['qual_stddev'][order], subset['mean_time'][order],
                 marker='o', label=f"Threshold Noise: {th_noise}")

    plt.xlabel("Nest Quality Noise") # label x-axis
    plt.ylabel("Mean Decision Time") # label y-axis
    plt.title("Decision Time vs Nest Quality Noise for Each Threshold Noise") #
         set title
    plt.grid(True) # add grid
    plt.legend() # add lagend
    plt.savefig(out_path) # save figure the folder
    plt.close()


def PlotSelectionDistribution(results, threshold_stddev, out_path):

    # Convert results into numpy array
    data = np.array(results,dtype=[('qual_stddev', float), ('threshold_stddev',
         float), ('accuracy', float),
                                         ('mean_time', float), ('accepts', object)])

    # Sorted quality noise
    qual_stddev = sorted(list(set(data['qual_stddev'])))

    # Create one subplot for each threshold noise value
    fig, axes = plt.subplots(1, len(threshold_stddev), figsize=(16, 5))

    if len(threshold_stddev) == 1:
        axes = [axes]

    # Iterate over each threshold standard deviation
    for i, th_noise in enumerate(threshold_stddev):
        ax = axes[i]

        # Pick the rows where the threshold matches th_noise
        subset = data[data['threshold_stddev'] == th_noise]

        # Store selections for each nest
        home_visits = []
        poor_visits = []
        good_visits = []

        # Iterate over each nest quality standard deviation
        for qual_noise in qual_stddev:

```

```
            # Pick the rows where the quality matches qual_noise
            row = subset[subset['qual_stddev'] == qual_noise][0]
            accepts = row['accepts'] # store accepted nest indices

            total = len(accepts) # total number of decisions

             # Calculate selections for each nest type
            home_visits.append(np.sum(accepts == 0) / total)
            poor_visits.append(np.sum(accepts == 1) / total)
            good_visits.append(np.sum(accepts == 2) / total)

         # Plot stacked bar chart
        ax.bar(qual_stddev, home_visits, label="Home", color="gray")

        # Stack poor nest selections
        ax.bar(qual_stddev, poor_visits, bottom=home_visits, label="Poor␣nest",
             color="yellow")
        # Stack good nest selections
        ax.bar(qual_stddev, good_visits, bottom=np.array(home_visits) + np.
            array(poor_visits),
               label="Good␣nest", color="red")

        ax.set_title(f"Threshold␣Noise␣=␣{th_noise}") # set subplots ttitles
        ax.set_xlabel("Nest␣Quality␣Noise") # label x-axis
        ax.set_xticks(qual_stddev) # # Set x as nest quality noises

        # Label y-axis onl at the first
        if i == 0:
            ax.set_ylabel("Nest␣Selection")

    plt.legend(bbox_to_anchor=(0.5, 1.10)) # add lagend
    plt.tight_layout()
    fig.savefig(out_path, bbox_inches="tight") # save figure the folder
    plt.close(fig)
```

params.yaml

```
qual_stddev: [0.1, 0.5, 1.0, 1.5, 3.0, 4.0] # Nest quality noise
threshold_stddev: [0.1, 0.6, 1.2] # Threshold noise
n: 500 # number of ants
threshold_mean: 4.5 # threshold
```