Baboon simulation report

Carpio Chicote, Álvaro
acarpio@student.ethz.ch

Cuesta Sierra, Pablo
cuestap@student.ethz.ch

Romero de Frutos, Lydia
romerol@student.ethz.ch

Schröer, Jannick
jschroeer@student.ethz.ch

May 11, 2025

This document is a report on the baboon simulation project. It contains the details of the simulation software and procedure that we have developed in our project to simulate the movements of a group of baboons in a 2 dimensional space according to the model proposed by [5].

The code can be found in our GitHub repository [1] and it is self-explanatory. Classes and functions are documented with docstrings and the `README` file in the repository has a guide to the code structure. This document is a curated compilation of the most relevant information from said docstrings.

Contents

1 Model description

We first present the main objects of the simulation: how we represent the baboons trajectories.

1.1 Baboon trajectories

- `baboons`: $(M, 2)$-(shaped-)`np.ndarray` (i.e., $M \times 2$ matrix)
  - M is the number of baboons.
  - 2 is the x_1 and x_2 coordinates of the baboon.
- $\mathtt{x} =$ `baboons_trajectory`: $(N, M, 2)$-`np.ndarray` (i.e., $N \times M \times 2$ matrix)
  - N is the number of steps.
  - M is the number of baboons.
  - 2 is the x_1 and x_2 coordinates of the baboon.

  For example, `baboons_trajectory[t, i]`$\in \mathbb{R}^2$ is the position of baboon i at time t.

1.2 SDE Model

The i-th baboon position is given by the following Stochastic Differential Equation (SDE):

$$d\mathtt{x}^i = f^i(\mathtt{x}[:t], \omega)dt + g^i(\mathtt{x}[:t], \omega) \cdot dW_t^i(\omega), \tag{1}$$

where:

- $\mathtt{x}^i$ is the full trajectory `x[:, i]` of baboon i.
- $\mathtt{x}[:t]$ is the full trajectory of all baboons up to time t. It is a $(t, M, 2)$-`np.ndarray`.
- $f(\mathtt{x}[:t], \omega) = (f^1(\mathtt{x}[:t], \omega), \ldots, f^M(\mathtt{x}[:t], \omega))$ is the **DRIFT** of the SDE. $f^i \in \mathbb{R}^2$ denotes the average change in position of baboon i. This term may include a random component (thus the dependency on ω). We can interpret this random component as the randomness included in the baboon decision-making process. f outputs an $(M, 2)$-`np.ndarray`.
- $g(\mathtt{x}[:t], \omega) = (g^1(\mathtt{x}[:t], \omega), \ldots, g^M(\mathtt{x}[:t], \omega))$ is the **DIFFUSION** of the SDE. $g^i \in \mathbb{R}^{2\times J}$ for each i. g outputs an $(M, 2, J)$-`np.ndarray`.
- W_t is an $M \times J$-dimensional Brownian motion. $W_t^i \in \mathbb{R}^J$ and $g^i \cdot W_t^i \in \mathbb{R}^2$ can be interpreted as the noisy component of the baboon's movement.

In practice, this will be implemented with an Euler-Maruyama scheme:

$$\mathtt{x}[t+1] = \mathtt{x}[:t] + f(\mathtt{x}[:t]) \cdot \Delta t + g(\mathtt{x}[:t]) \cdot \Delta W_t, \tag{2}$$

where Δt is the fixed time step size and ΔW_t is a normal random variable with mean 0 and variance Δt (each coordinate and each realization of the increments

$\Delta W_t, \Delta W_{t+\Delta t}, \ldots$ are all independent). Multiplication "$\cdot$" here is supposed to be in a matrix sense for each baboon:

$$g(\mathtt{x}[:t]) \cdot dW_t = \mathtt{np.einsum("mij,\ mj\ ->\ mi",}\ g(\mathtt{x}[:t]),\ \Delta W_t\mathtt{)}. \tag{3}$$

In general, we will choose $J = 2$ for the dimension of the Brownian motion driving each baboon's SDE. This is a reasonable choice as the baboons move in a 2D space. Moreover, we will in general have

$$g^i(\mathtt{x}[:t], \omega) = \underbrace{\sigma(\mathtt{x}[:t])}_{\in\mathbb{R}} I_{2\times 2},$$

so that the noisy term of the equation is isotropic. This means that the baboon's movement is equally likely in all directions. σ is a scalar function that depends on the baboons' trajectory up to time t.

We will use the following notation in the code:

- $f \equiv$ `drift`
- $g \equiv$ `diffusion`
- `x`[:t] $\equiv$ `baboons_trajectory[:t]`

Note We could be more general, and put the output of the diffusion function to be $(M, 2, J, 2)$-shaped and the BM, $(J, 2)$-shaped, and their multiplication is done with Einstein summation convention: `mijk, jk` $\rightarrow$ `mi`. This would allow for the driver (W_t) of the equation to affect different baboons jointly. This would overcomplicate the model in our case, but it could be explored for other models and maybe to apply neural networks to learn the SDE expression as in [2, 3].

2 Simulation

We have implemented an Euler-Maruyama solver for the SDE simulation (2). While formally the SDE that we are solving here is equivalent to (2), for convinience we will encode information of the baboons' trajectory and random choices into a "state" object. This state object will be passed to the drift and diffusion functions, which will be called at each time step and will also output the updated state. This allows us to keep track of the baboons' trajectory and random choices without having to re-compute trajectory information at each time step. This solver takes as input:

- `total_time_steps`: Total number of simulation steps.
- `initial_baboons`: Initial positions of baboons. Shape: $(n\_baboons, 2)$. We typically use a normal distribution with mean 0 and fixed standard deviation for the initial positions.

- `dt`: Time step size.
- `seed`: Random seed for reproducibility. This means that a call to the solver with exactly the same parameters will yield the same result.
- `drift_diffusion_with_state`: Callable function that takes the baboons trajectory, random generator, and state as input and returns the drift, diffusion, and next state. This function will be called at each time step of the solver.

  To be more precise, the `drift_diffusion_with_state` has the following signature (using the `typing` [4] notation):

```
from typing import Callable, Optional, Tuple
import numpy as np
import numpy.typing as npt
from sklearn.utils import Bunch

DriftDiffusionWithStateType = Callable[
    [
        # baboons_trajectory[:t], shape (t, n_baboons, 2)
        npt.NDArray[np.float64],
        # random generator (this is the omega)
        np.random.Generator,
        # state (if None, it will be initialized)
        Optional[Bunch],
    ],
    Tuple[
        # output of drift function, shape (n_baboons, 2)
        npt.NDArray[np.float64],
        # output of diffusion function, shape (n_baboons, 2, J)
        npt.NDArray[np.float64],
        # next state, shape (n_baboons, 2)
        Bunch,
    ],
]
```

In the following sections we will see the form that the state object takes.

3 Instances of drift and diffusion functions with state

We have coded two main versions of the drift and diffusion functions with state. The second one utilizes the concept of “targets”, which represent coordinates in the 2D space that the baboons are trying to reach (e.g. food or water sources). The first one is a simple version that does not use targets.

These functions are relatively complex, but their docstrings explain the behaviour comprehensively. Thus, we directly refer to the docstrings of the functions for a detailed explanation of their behaviour.

3.1 Drift and diffusion without targets

```
class State(Enum):
    """State of the baboon.
    The baboon can be in one of the following states:
        - following: The baboon is following another baboon.
        - group_influence: The baboon is influenced by the group
            (choose a random angle pointing to another).
        - still: The baboon is not moving (maybe only moving with
            a small perturbation).
        - random_walk: The baboon is doing a random walk (i.e.
            exploring on its own) with drift. The drift is randomly
            assigned at the beginning of the random walk and is
            kept until the baboon changes state.
    """
    following = 1
    group_influence = 2
    still = 3
    random_walk = 4


def state_driven_drift_diffusion_function(
    angle_std: float,
    group_influence_step_length: float,
    random_walk_step_length: float,
    random_walk_step_length_std: float,
    min_follow_distance: float,
    max_follow_distance: float,
    max_follow_step: float,
    state_diffusion_constants: dict[State, float],
    following_step_length_std: float = 0.2,
    following_step_length_proportion: float = 0.1,
    following_radius: float = 1.0,
    state_probabilities: Optional[dict[State, float]] = None,
    state_countdown_means: Optional[dict[State, float]] = None,
    probability_repeat_random_walk: float = 0.0,
    choose_drift_from_other_random_walkers: bool = True,
    new_random_walk_drift_angle_std: float = 10 * np.pi / 180,
) -> DriftDiffusionWithStateType:
    """
    Creates a drift + diffusion function where each baboon acts according to
    an internal state.

    Each baboon can be in one of four states: following, group_influence,
    still, or random_walk. State transitions occur after a countdown, drawn
```

```
from a Poisson distribution + 1.

Args:
    angle_std (float): Standard deviation for angular perturbations in
        group_influence state.
    group_influence_step_length (float): Base step length for
        group_influence.
    random_walk_step_length (float): base step length for random walk
        drift.
    random_walk_step_length_std (float): Standard deviation of noise added
        to random walk step size.
    min_follow_distance (float): Minimum distance between two baboons for
        one to follow the other.
    max_follow_distance (float): Maximum distance between two baboons for
        one to follow the other.
    max_follow_step (float): Maximum step size a baboon can take while
        following another baboon.
    following_radius (float): Radius up to which a baboon is satisfied with
        its following target. If the distance to the target is smaller than
        this radius, the baboon will move in the opposite direction.
    state_diffusion_constants (dict[State, float]): Diffusion coefficient
        for each state.
    following_step_length_std (float): Standard deviation of noise added to
        following step size.
    following_step_length_proportion (float): Proportion of the distance to
        the target used as a base for following step size.
    state_probabilities (Optional[dict[State, float]]):
        A dictionary mapping each State to its probability when sampling
        new states. If None, defaults to equal probability for all states.
    state_countdown_means (Optional[dict[State, float]]):
        Dictionary specifying Poisson means for each state.
        Defaults to 20 for all states.
    probability_repeat_random_walk (float): Probability of repeating the
        random walk state when transitioning from random_walk to another
        state.
    choose_drift_from_other_random_walkers (bool): If True, when a baboon
        transitions to the random_walk state, it will choose a drift from
        another baboon that is already in random_walk state plus some angle
        perturbation. If False, the baboon will choose a random drift
        direction.
    new_random_walk_drift_angle_std (float): Standard deviation of the
        angle perturbation for the new random walk drift. This is used when
        a baboon transitions to the random_walk state and is assigned a
        drift based on an existing random walker. The angle is perturbed
        by a normal distribution with this standard deviation.
Returns:
    DriftDiffusionWithStateType: A callable that computes the drift vector,
    diffusion matrix, and updated internal state for all baboons given
    their trajectory history and a random generator.

Notes:
    - In random_walk state, baboons move with a persistent random drift
        direction plus diffusion.
    - The internal state includes an additional field `random_walk_drift`
        to store assigned random walk drifts.
    - In the following state, baboons follow a target baboon which has to
        be far enough and in random_walk state.
"""
```

Here the state object has the form:

```
state_bunch: Bunch(
    state: npt.NDArray(shape=n_baboons, dtype=State),
    following_idx: npt.NDArray(shape=n_baboons, dtype=int),
    state_countdown: npt.NDArray(shape=n_baboons, dtype=int),
    random_walk_drift: npt.NDArray(shape=(n_baboons, 2), dtype=float),
)
```

3.2 Drift and diffusion with targets

This version of the drift and diffusion function is a modified version of the previous one. It includes the concept of targets, which represent coordinates in the 2D space that the baboons are trying to reach (e.g. food or water sources).

```
class State(Enum):
    """State of the baboon.
    The baboon can be in one of the following states:
        - following: The baboon is following another baboon.
        - group_influence: The baboon is influenced by the group (choose a
          random angle pointing to another).
        - still: The baboon is not moving (maybe only moving with a small
          perturbation).
        - random_walk: The baboon is doing a random walk (i.e. exploring on its
          own) with drift. The drift is randomly assigned at the beginning of
          the random walk and is kept until the baboon changes state.
        - target: The baboon is moving towards a target in the targets array.
    """
    following = 1
    group_influence = 2
    still = 3
    random_walk = 4
    target = 5


def state_driven_drift_diffusion_with_targets_function(
    angle_std: float,
    group_influence_step_length: float,
    random_walk_step_length: float,
    random_walk_step_length_std: float,
    min_follow_distance: float,
    max_follow_distance: float,
    max_follow_step: float,
    state_diffusion_constants: dict[State, float],
    targets: npt.NDArray[np.float64],
    following_step_length_std: float,
    following_step_length_proportion: float,
    following_radius: float,
    target_radius: float,
    n_max_targets: int = 7,
    state_probabilities: Optional[dict[State, float]] = None,
    state_countdown_means: Optional[dict[State, float]] = None,
    probability_repeat_random_walk: float = 0.0,
    choose_drift_from_other_random_walkers: bool = True,
    new_random_walk_drift_angle_std: float = 10 * np.pi / 180,
    new_target_noise_std: float = 5.0,
) -> DriftDiffusionWithStateType:
    """
    Creates a drift + diffusion function where each baboon acts according to
    an internal state.

    Each baboon can be in one of four states: following, group_influence,
    still, or random_walk. State transitions occur after a countdown, drawn
    from a Poisson distribution.

    Args:
        angle_std (float): Standard deviation for angular perturbations in
            group_influence state.
        group_influence_step_length (float): Base step length for
            group_influence.
        random_walk_step_length (float): base step length for random walk
            drift.
        random_walk_step_length_std (float): Standard deviation of noise added
            to random walk step size.
        min_follow_distance (float): Minimum distance between two baboons for
            one to follow the other.
        max_follow_distance (float): Maximum distance between two baboons for
            one to follow the other.
```

```
max_follow_step (float): Maximum step size a baboon can take while
    following another baboon.
following_radius (float): Radius up to which a baboon is satisfied with
    its following target. If the distance to the target is smaller than
    this radius, the baboon will move in the opposite direction.
state_diffusion_constants (dict[State, float]): Diffusion coefficient
    for each state.
following_step_length_std (float): Standard deviation of noise added to
    following step size.
following_step_length_proportion (float): Proportion of the distance to
    the target used as a base for following step size.
state_probabilities (Optional[dict[State, float]]):
    A dictionary mapping each State to its probability when sampling
    new states. If None, defaults to equal probability for all states.
state_countdown_means (Optional[dict[State, float]]):
    Dictionary specifying Poisson means for each state.
    Defaults to 20 for all states.
probability_repeat_random_walk (float): Probability of repeating the
    random walk state when transitioning from random_walk to another
    state.
choose_drift_from_other_random_walkers (bool): If True, when a baboon
    transitions to the random_walk state, it will choose a drift from
    another baboon that is already in random_walk state plus some angle
    perturbation. If False, the baboon will choose a random drift
    direction.
new_random_walk_drift_angle_std (float): Standard deviation of the
    angle perturbation for the new random walk drift. This is used when
    a baboon transitions to the random_walk state and is assigned a
    drift based on an existing random walker. The angle is perturbed
    by a normal distribution with this standard deviation.
targets (npt.NDArray[np.float64]): Array of target
    coordinates (n_targets, 2). If provided, target-state baboons will
    move towards these targets as a drift vector. The target is chosen
    if it is the closest to the baboon's current position. After a
    target is chosen by a target-state baboon, only following baboons
    which have not come close enough to the target will be able to
    choose this target-state baboon it as a following target.

    Logic is as follows:
    - If a baboon is in target-state, it will choose the first
        target and move towards it until it is close enough
        (determined by the baboon's target_radius).
        After it is close enough to the target, it will select next
        target from the list of targets and move towards it (and so
        on until the target-state runs out). Next time that the
        baboon is in target-state, it will choose the first target
```

```
            that it has not yet visited (i.e. gotten close to according
            to target_radius).
        - If a baboon is in following-state, it will choose to follow
            baboons that are in target-state or random_walk-state but
            discarding baboons in target-state which are going towards
            target that the follower has already yet visited (i.e. the
            follower did not yet get close enough to yet according to
            target_radius). If no baboon is available to follow, the
            baboon will become target-state.
        At the end fo the loop, the state will keep an array of
        visited_targets = np.array((n_baboons, n_targets), dtype=bool)
        where each baboon will have a boolean array of size n_targets
        indicating whether it has already visited the target or not (i.e.
        gotten closer to it than target_radius).

    Returns:
        DriftDiffusionWithStateType: A callable that computes the drift vector,
        diffusion matrix, and updated internal state for all baboons given
        their trajectory history and a random generator.
    """
```

Here the state object has more information to track targets and target visits:

```
state_bunch: Bunch(
    state: npt.NDArray(shape=n_baboons, dtype=State),
    following_idx: npt.NDArray(shape=n_baboons, dtype=int),
    state_countdown: npt.NDArray(shape=n_baboons, dtype=int),
    random_walk_drift: npt.NDArray(shape=(n_baboons, 2), dtype=float),
    targets: npt.NDArray(shape=(n_targets, 2), dtype=float),
    visited_targets: npt.NDArray(shape=(n_baboons, n_targets), dtype=bool),
)
```

For more details on the behaviour of this function, please refer to the code [1].

References

[1] Álvaro Carpio Chicote et al. *Baboon Simulation*. https://github.com/IQisMySenpai/baboon-simulation. Accessed: 2025-05-11. 2025.

[2] Ricky T. Q. Chen et al. *Neural Ordinary Differential Equations*. 2019. arXiv: 1806.07366 [cs.LG]. URL: https://arxiv.org/abs/1806.07366.

[3] Patrick Kidger et al. *Neural SDEs as Infinite-Dimensional GANs*. 2021. arXiv: 2102.03657 [cs.LG]. URL: https://arxiv.org/abs/2102.03657.

[4] Python Software Foundation. *typing — Support for type hints*. Python 3 Standard Library Documentation. Python Software Foundation. 2024. URL: `https://docs.python.org/3/library/typing.html`.

[5] Ariana Strandburg-Peshkin et al. "Shared decision-making drives collective movement in wild baboons". In: *Science* 348.6241 (2015), pp. 1358–1361. DOI: `10.1126/science.aaa5099`. URL: `https://www.science.org/doi/10.1126/science.aaa5099`.