simlandr: Simulation-Based Landscape Construction for Dynamical Systems

by Jingmeng Cui, Merlijn Olthof, Anna Lichtwarck-Aschoff, Tiejun Li, and Fred Hasselman

Abstract We present the simlandr package for R, which provides a set of tools for constructing potential landscapes for dynamical systems using Monte Carlo simulation. Potential landscapes can be used to quantify the stability of system states. While the canonical form of a potential function is defined for gradient systems, generalized potential functions can also be defined for non-gradient dynamical systems. Our method is based on the potential landscape definition from the steady-state distribution, and can be used for a large variety of models. To facilitate simulation and computation, we introduce several novel features, including data structures optimized for batch simulations under varying conditions, an out-of-memory computation tool with integrated hash-based file-saving systems, and an algorithm for efficiently searching the minimum energy path. Using a multistable cell differentiation model as examples, we illustrate how simlandr can be used for model simulation, landscape construction, and barrier height calculation. The simlandr package is available at https://CRAN.R-project.org/package=simlandr, under GPL-3 license.

1 Introduction

To better understand a dynamical system, it is often important to know the stability of different states. The metaphor of a potential landscape consisting of hills and valleys has been used to illustrate differences in stability in many fields, including genetics (Wang et al., 2011; Waddington, 1966), ecology (Lamothe et al., 2019), and psychology (Olthof et al., 2023). In such a landscape, the stable states of the system correspond to the lowest points (minima) in the valleys of the landscape. Just like a ball that is thrown in such a landscape will eventually gravitate towards such a minimum, the dynamical system is conceptually more likely to visit its stable states in which the system is also more resilient to noise. For example, in the landscape metaphor of psychopathology (Figure 1), the valleys represent different mental health states, their relative depth represents the relative stability of the states, and the barriers between valleys represent the difficulty of transitioning between these states (Olthof et al., 2023, Hayes and Andrews (2020)). When the healthy state is more stable, the person is more likely to stay mentally healthy, whereas when the maladaptive state is more stable, the person is more likely to suffer from mental disorders.

Yet, formally quantifying the stability of states is a nontrivial question. Here we present an R package, **simlandr**, that can quantify the stability of various kinds of systems without many mathematical restrictions.

Dynamical systems are usually modeled by stochastic differential equations, which may depend on the past history (i.e., may be non-Markovian, Stumpf et al. (2021)). They take the general form of

$$\mathrm{d}\boldsymbol{X}_t = \boldsymbol{b}(\boldsymbol{X}_t, \boldsymbol{H}_t)\mathrm{d}t + \boldsymbol{\sigma}(\boldsymbol{X}_t, \boldsymbol{H}_t)\mathrm{d}\boldsymbol{W}, \tag{1}$$

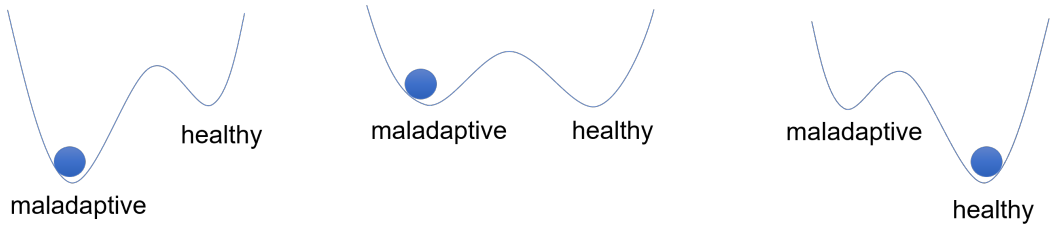

Figure 1: Illustration of the ball-and-landscape metaphor commonly used in the field of psychopathology.

where $\boldsymbol{X}_t$ is the random variable representing the current state of the system and $\boldsymbol{H_t}$ represents the past history of the system $\boldsymbol{H}_t = \{\boldsymbol{X_s} | s \in [0, t)\}$[1] The first term on the right-hand side of Eq (1) represents the deterministic part of the dynamics, which is a function of the system's current state $\boldsymbol{b}(\boldsymbol{X}_t, \boldsymbol{H}_t)$. The second term represents the stochastic part, which is standard white noise $\mathrm{d}\boldsymbol{W}$ multiplied by the noise strength $\boldsymbol{\sigma}(\boldsymbol{X}_t, \boldsymbol{H}_t)$.

If the dynamical equation (Eq (1)) can be written in the following form

$$\mathrm{d}\boldsymbol{X} = -\nabla U \mathrm{d}t + \sqrt{2}\mathrm{d}\boldsymbol{W}, \tag{2}$$

then U is the potential function of the system.[2]. However, this is not possible for general dynamical systems. The trajectory of such a system may contain loops which are not possible to be represented by a gradient system (this issue was compared to Escher's stairs by Rodríguez-Sánchez et al. (2020)). In this case, further generalization is needed. The theoretical background of **simlandr** is the generalized potential landscape by Wang et al. (2008), which is based on the Boltzmann distribution and the steady-state distribution of the system. The Boltzmann distribution is a distribution law in physics, which states the distribution of classical particles depends on the energy level they occupy. When the energy is higher, the particle is exponentially less likely to be in such states

$$P(\boldsymbol{x}) \propto \exp(-U). \tag{3}$$

This is then linked to dynamical systems by the steady-state distribution. The steady-state distribution of stochastic differential equations is the distribution that does not change over time, denoted by P_{SS} which satisfies

$$\frac{\partial P_{\mathrm{SS}}(\boldsymbol{x}, t)}{\partial t} = 0. \tag{4}$$

The steady-state distribution is important because it extracts time-invariant information from a set of stochastic differential equations. Substituting the steady-state distribution to Eq (3) gives Wang's generalized potential landscape function (Wang et al., 2008)

$$U(\boldsymbol{x}) = -\ln P_{\mathrm{SS}}(\boldsymbol{x}). \tag{5}$$

If the system has ergodicity (i.e., after sufficient time it can travel to all possible states in the state space), the long-term sample distribution can be used to estimate the steady-state distribution, and the generalized potential function can be calculated.

Our approach is not the only possible way for constructing potential landscapes. Many other theoretical approaches are available, including the SDE decomposition method by Ao (2004) and the quasi-potential by Freidlin and Wentzell (2012). Various strategies to numerically compute these landscapes have been proposed (see Zhou and Li (2016) for a review). However, available realizations are still scarce. To our knowledge, besides **simlandr**, there are two existing packages specifically for computing potential landscapes: the **waydown** package (Rodríguez-Sánchez, 2020) and the **QPot** package (Moore et al., 2016; Dahiya and Cameron, 2018). The **waydown** package uses the skew-symmetric decomposition of the Jacobian, which theoretically produces landscapes that are similar to Wang et al. (2008) (but see Cui et al. (2023a) for a potential technical issue with this package.). The **QPot** package uses a path integral method that produces quasi-landscapes following the definition by Freidlin and Wentzell (2012). Because of the analytical methods used by **waydown** and **QPot**, they both require the dynamic function to be Markovian and differentiable in the whole state space. Moreover, they can only be used for systems of up to two dimensions. **simlandr**, in contrast, is based on Monte Carlo simulation and the steady-state distribution. It does not have specific requirements for the model. Even for models that are not globally

[1]The corresponding variable representing positions in the state space is not a random variable, so we use lowercase $\boldsymbol{x}$ for it. This convention will be followed throughout this article.

[2]Under zero inertia approximation.

differentiable, have history-dependence, and are defined in a high-dimensional space, **simlandr** is still applicable (e.g., Cui et al. (2023b)). Therefore, **simlandr** can be applied to a much wider range of dynamical systems, illustrate a big picture of dominated attractors, and investigate how the stability of different attractors may be influenced by model parameters. As a trade-off, **simlandr** is not designed for rare events sampling in which the noise strength $\sigma(\boldsymbol{X})$ is extremely small, nor for the precise calculation of the tail probability and transition paths. Instead, it is better to view **simlandr** as a semi-quantitative tool that provides a broad overview of key attractors in dynamic systems, allowing for comparisons of their relative stability and the investigation of system parameter influences. We will show some typical use cases of **simlandr** in later sections. Some key terms used in this article are summarized in Table 1.

Table 1: Summary of key terms used in this article.

Term	Explanation
Potential landscape metaphor	A conceptual metaphor representing the stability of a complex dynamic system as an uneven landscape, with a ball on it representing the system's state. This can be quantitatively realized in various ways (Zhou and Li, 2016).
Gradient system	A system whose deterministic motion can be described solely by the gradient of a potential function.
Non-Markovian system	A system whose future evolution depends not only on its current state but also on its past history.
Steady-state distribution	The probability distribution of a dynamic system that remains unchanged over time.
Ergodic system	A dynamic system that, given enough time, will eventually pass through all possible states.
Minimum energy path (MEP)	A transition path linking two local minima and passing through a saddle point (Wan et al., 2024). It is always parallel to the gradient of the energy landscape, representing an efficient transition route.

2 Design and implementation

The general workflow of **simlandr** involves three steps: model simulation, landscape construction, and barrier height computation. See Figure 2 for a summary.

2.1 Step 1: model simulation

For the first step, a simulation function should be created by the user. This function should be able to simulate how the dynamical system of interest evolves over time and record its state in every time step. This can often be done with the Euler-Maruyama method. If the SDEs are up to three dimensions and Markovian, a helper function from **simlandr**, `sim_SDE()`, can also be used. This function is based on the simulation utilities from the **Sim.DiffProc** (Guidoum and Boukhetala, 2020) and the output can be directly used for later steps. Moreover, the `multi_init_simulation()` function can be used to simulate trajectories from various starting points, thus reducing the possibility for the system to be trapped in a local minimum. The `multi_init_simulation()` function also supports parallel simulation based on the **future** framework to improve time efficiency Bengtsson (2021).

For Monte Carlo methods, it is important that the simulation *converges*, which means the distribution of the system is roughly stable. The precision of the steady-state distribution estimation, according to Eq (5), determines the precision of the distribution estimation. **simlandr** provides a visual tool to compare the sample distributions in different stages

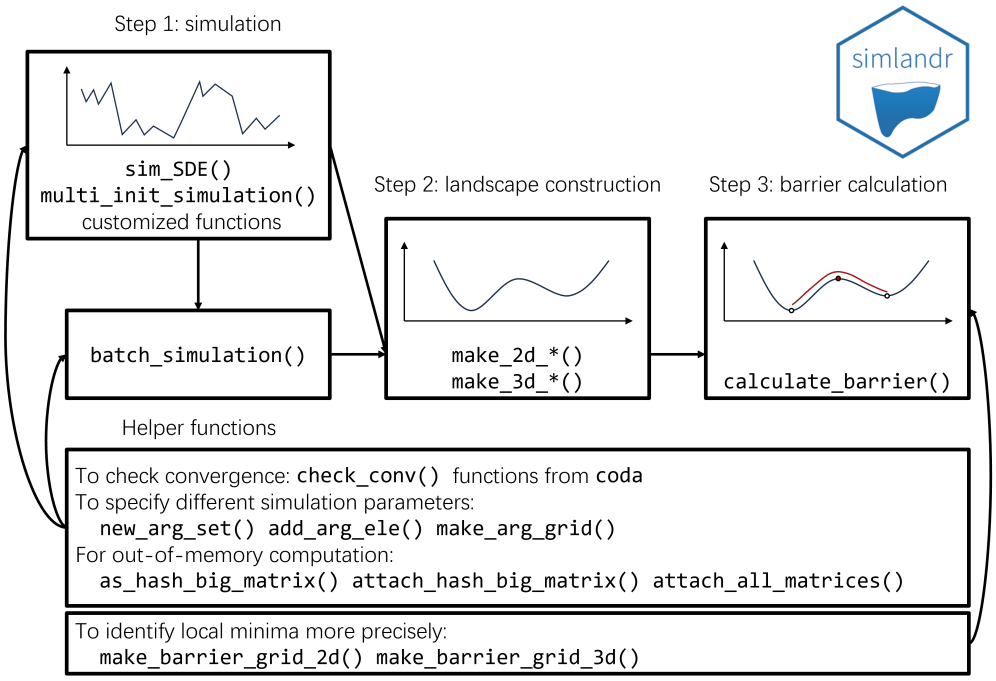

Figure 2: The structure and workflow of simlandr.

(`check_conv()`), whereas the **coda** package (Plummer et al., 2006) can be used for more advanced diagnostics. The output of the `sim_SDE()` and `multi_init_simulation()` functions also uses the classes from the **coda** package to enable easy convergence diagnosis. To achieve ergodicity in reasonable time, sometimes stronger noises need to be added to the system.

A simulation function is sufficient if the user is only interested in a single model setting. If the model is parameterized and the user wants to investigate the influence of parameters on the stability of the system, then multiple simulations need to be run with different parameter settings. **simlandr** provides functions to perform batch simulations and store the outputs for landscape construction as one object. This can later be used to compare the stability under different parameter settings or produce animations to show how a model parameter influences the stability of the model.

In many cases, the output of the simulation is so large that it cannot be properly stored in the memory. **simlandr** provides a hash_big_matrix class, which is a modification of the big.matrix class from the **bigmemory** package (Kane et al., 2013), that can perform out-of-memory computation and organize the data files in the disk. In an out-of-memory computation, the majority of the data is not loaded into the memory, but only the small subset of data that is used for the current computation step. Therefore, the memory occupation is dramatically reduced. The big.matrix class in the **bigmemory** package provides a powerful tool for out-of-memory computation. It, however, requires an explicit file name for each matrix, which can be cumbersome if there are many matrices to be handled, and this is likely to be the case in a batch simulation. The `hash_big_matrix` class automatically generates the file names using the md5 values of the matrices with the **digest** package(Eddelbuettel et al., 2021) and stores it within the object. Therefore, the file links can also be restored automatically.

2.2 Step 2: landscape construction

simlandr provides a set of tools to construct 2D, 3D, and 4D[3] landscapes from single or multiple simulation results. The steady-state distribution for selected variables of the system

[3]In this package, we use the number of dimensions in landscape plots (including U to define the dimension of landscapes. The x-, y-, z-, and color- axes can all be regarded as a dimension. Therefore, the dimension of a landscape can be one more than the dimension of the kernel smooth function.

is first estimated using the kernel density estimates (KDE). The density function in R is used for 2D landscapes, whereas the **ks** package (Chacón and Duong, 2018) is used by default for 3D and 4D landscapes because of its higher efficiency. Then the potential function U is calculated from Eq (5). The landscape plots without a z-axis are created with **ggplot2** (Wickham, 2016), and those with a z-axis are created with **plotly** (Sievert, 2020). These plots can be further refined using the standard **ggplot2** or **plotly** methods. See Table 2 for an overview for the family of landscape functions.

Table 2: Overview of various landscape functions provided by `simlandr`. Dimensions in bold represent the potential U calculated by the function. Dimensions in italic represent model parameters. Dimensions in parentheses are optional.

Type of Input	Function	Dimensions
Single simulation data	`make_2d_static()`	x, **y**
	`make_3d_static()`	1 x, y, **z+color**; (2) x, y, **color**
	`make_4d_static()`	x, y, z, **color**
Multiple simulation data	`make_2d_matrix()`	x, **y**, *cols*, *(rows)*
	`make_3d_matrix()`	x, y, **z+color**, *cols*, *(rows)*
	`make_3d_animation()`	1 x, y, **z+color**, *fr*; (2) x, y, **color**, *fr*; (3) x, y, **z+color**, *cols*

2.3 Step 3: barrier height calculation

An important property of the states in a landscape is their stability, which can be indicated by the barrier height that the system has to overcome when it transitions between one stable state to another adjacent state (see Cui et al. (2023b) for further discussions about different stability indicators). The barrier height is also related to the escape time that the system transitions from one valley to another, which can be tested empirically (Wang et al., 2008). **simlandr** provides tools to calculate the barrier heights from landscapes. These functions look for the local minima in given regions and try to find the saddle point between two local minima. The potential differences between the saddle point and local minima are calculated as barrier heights.

In 2D cases, there is only one possible path connecting two local minima. The point on the path with the highest U is identified as the saddle point. For 3D landscapes, there are multiple paths between two local minima. If we treat the system *as if* it is a gradient system with Brownian noise, then the most probable path (termed as the *minimum energy path*, MEP) that the system transitions is that it first goes along the steepest *ascent* path from the starting point, and then goes along the steepest *descent* path to the end point (E and Vanden-Eijnden, 2010). We find this path by minimizing the following action using the Dijkstra (1959) algorithm (Heymann and Vanden-Eijnden, 2008)

$$\varphi_{\text{MEP}} = \arg\min_{\varphi} \int_A^B |\nabla U||\mathrm{d}\varphi| \left(\approx \min_{\varphi} \Sigma_i |\nabla U_i||\Delta\varphi_i| \right), \tag{6}$$

where A and B are the starting and end points and φ is the path starting at A and ending in B. After that, the point with the maximum potential value on the MEP is identified as the saddle point. Note that while the barrier height still indicates the stability of local minima, the MEP may not be the true most probable path for a nongradient system to transition between stable states.

3 Examples

We use two dynamical systems to illustrate the usage of the **simlandr** package. The first one is a two-dimensional stochastic non-gradient gene expression model, which was used by

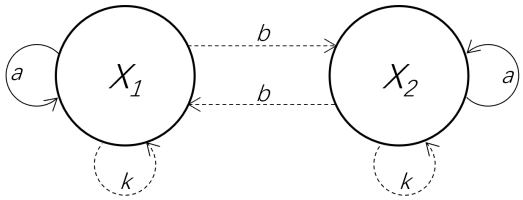

Figure 3: A graphical illustration of the relationship between the activation levels of the two genes. Solid arrows represent positive relationships (i.e., activation) and dashed arrows represent negative relationships (i.e., inhibition).

Wang et al. (2011) to represent cell development and differentiation. The second example is a dynamic model of panic disorder (Robinaugh et al., 2024) which contains many more variables and parameters, non-Markovian property, and non-differentiable formulas. We mainly use the first example to show the agreement of the results from **simlandr** with previous analytic results, and the second example as a typical use case of a complex dynamic model which is not treatable with other methods (also see Cui et al. (2023b) for more substantive discussions). Note that, both systems include more than two variables, making it impossible to perform the landscape analysis with other available R packages.

3.1 Example 1: the gene expression model

This model is built on the mutual regulations of the expressions of two genes, in which X_1 and X_2 represent the expression levels of two genes which activate themselves and inhibit each other. A graphical illustration is shown in Figure 3 (adapted from Wang et al. (2011)). Their dynamic functions can be written as

$$\frac{\mathrm{d}X_1}{\mathrm{d}t} = \frac{ax_1^n}{S^n + x_1^n} + \frac{bS^n}{S^n + x_2^n} - kx_1 + \sigma_1 \frac{\mathrm{d}W_1}{\mathrm{d}t}, \tag{7}$$

$$\frac{\mathrm{d}X_2}{\mathrm{d}t} = \frac{ax_2^n}{S^n + x_2^n} + \frac{bS^n}{S^n + x_1^n} - kx_2 + \sigma_2 \frac{\mathrm{d}W_2}{\mathrm{d}t}, \tag{8}$$

$$\frac{\mathrm{d}a}{\mathrm{d}t} = -\lambda a + \sigma_3 \frac{\mathrm{d}W_3}{\mathrm{d}t}, \tag{9}$$

where a represents the strength of self-activation, b represents the strength of mutual-inhibition, and k represents the speed of degradation. The development of an organism is modeled as a decreasing at a certain speed λ. In the beginning, there is only one possible state for the cell. After a certain milestone, the cell differentiates into one of the two possible states.

This model can be simulated using the `sim_SDE()` function in **simlandr**, with the default parameter setting $b = 1, k = 1, S = 0.5, n = 4, \lambda = 0.01$, and $\sigma_1 = \sigma_2 = \sigma_3 = 0.2$.

```
# Load the package.
library(simlandr)

# Specify the simulation function.
b <- 1
k <- 1
S <- 0.5
n <- 4
lambda <- 0.01

drift_gene <- c(rlang::expr(z * x^(!!n)/((!!S)^(!!n) + x^(!!n)) + (!!b) *
   (!!S)^(!!n)/((!!S)^(!!n) + y^(!!n)) - (!!k) * x), rlang::expr(z * y^(!!n)/((!!S)^(!!n) +
    y^(!!n)) + (!!b) * (!!S)^(!!n)/((!!S)^(!!n) + x^(!!n)) - (!!k) * y),
    rlang::expr(-(!!lambda) * z)) |>
```

```
    as.expression()

diffusion_gene <- expression(0.2, 0.2, 0.2)

# Perform a simulation and save the output.
set.seed(1614)
single_output_gene <- sim_SDE(drift = drift_gene, diffusion = diffusion_gene,
    N = 1e+06, M = 10, Dt = 0.1, x0 = c(0, 0, 1), keep_full = FALSE)
```

After the simulation, we perform some basic data wrangling to produce a dataset that can be used for further analysis. We create a new variable `delta_x` as the difference between X1 (X) and X2 (Y), and we rename the variable Z as a.

```
single_output_gene2 <- do.call(rbind, single_output_gene)
single_output_gene2 <- cbind(single_output_gene2[, "X"] - single_output_gene2[,
    "Y"], single_output_gene2[, "Z"])
colnames(single_output_gene2) <- c("delta_x", "a")
```

We then perform the convergence check on the simulation result. First, we convert the simulation output to the format for the coda package, and thin the output to speed up the convergence check.

```
single_output_gene_mcmc_thin <- as.mcmc.list(lapply(single_output_gene,
    function(x) x[seq(1, nrow(x), by = 100), ]))
```

We then show the convergence diagnosis plot to check the convergence of the simulation in Figure 4. The distribution of the two key variables in different simulation stages are converging, indicating that the simulation is long enough to provide reliable estimation of the steady-state distribution. Other convergence checks can also be readily performed using the **coda** package.

```
plot(single_output_gene_mcmc_thin)
```

We generated the 3D landscape for this model with `make_3d_single()`. Here, we use `x`, `y` to specify the variables of interest, and use `lims` to specify the limits of the x and y axes for the landscape. The `lims` argument can be left blank, then the limits will be automatically calculated.

```
l_single_gene_3d <- make_3d_single(single_output_gene2, x = "delta_x",
    y = "a", lims = c(-1.5, 1.5, 0, 1.5), Umax = 8)
```

The resulting landscape is shown in the left panel of Figure 5. In this plot, the x-axis represents $\Delta x (= x_1 - x_2)$, and the y-axis represents a. To compare with, the potential landscape obtained analytically by Wang et al. (2011) is shown in the right panel Figure 5. The result of **simlandr** appears to be very close to the result based on the analytical derivation. Note that because different normalization methods were used, the U values of the two landscapes are not directly comparable. Here, we are mainly interested in their relative shape.

```
plot(l_single_non_grad_3d)
```

We then calculate the barrier for the landscape using `calculate_barrier()`. The barrier is calculated by specifying the start and end locations, and the radii of the start and end locations. The height of the barrier from two sides can be calculated with `get_barrier_height()`.

```
b_single_gene_3d <- calculate_barrier(l_single_gene_3d, start_location_value = c(0,
    1.2), end_location_value = c(1, 0.2), start_r = 0.3, end_r = 0.3)

get_barrier_height(b_single_gene_3d)
```

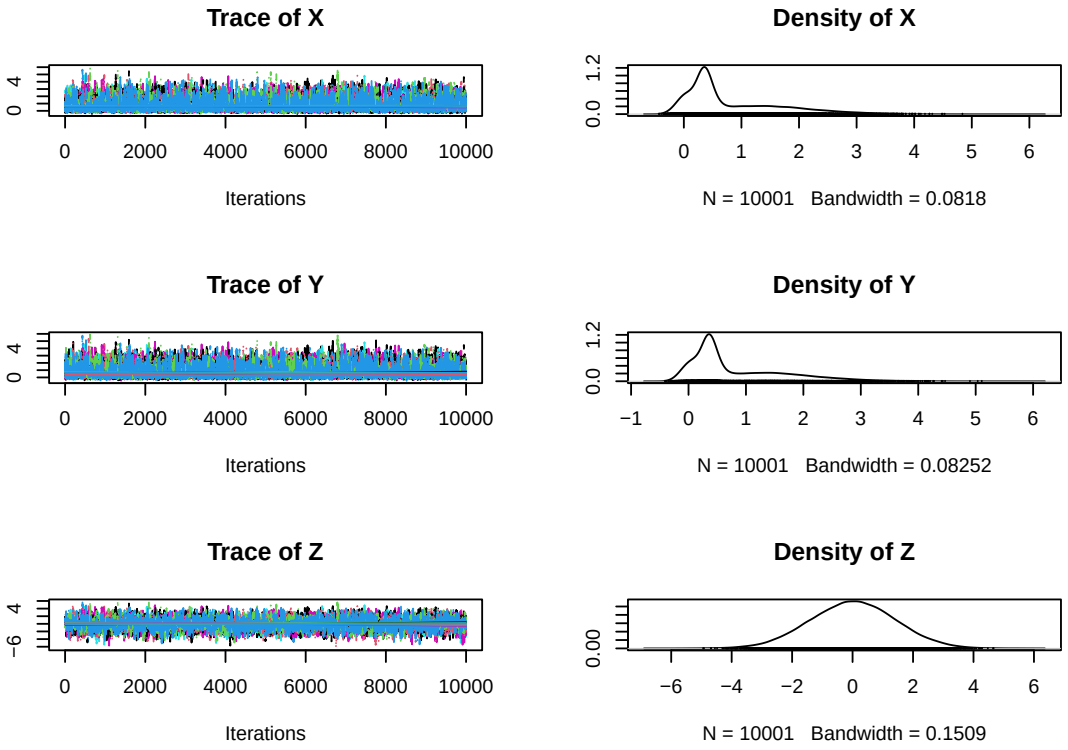

Figure 4: The convergence check result for the simulation of the gene expression model. The variables in different simulation stages did not show distributional differences, indicating that the simulation is long enough to provide a reliable estimation of the steady-state distribution.

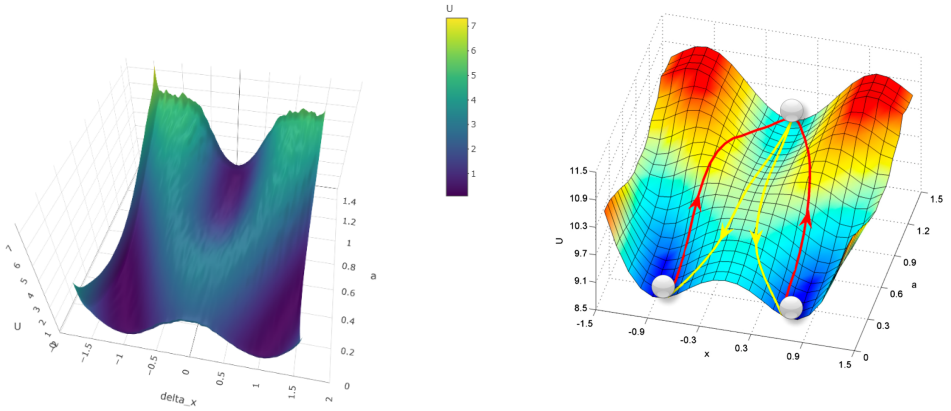

Figure 5: The 3D landscape (potential value as z-axis) for the gene expression model. The left panel is the plot produced by simlandr; the right panel is the potential landscape obtained analytically by Wang et al. (2008), reproduced with the permission of the authors and in accordance with the journal policy.

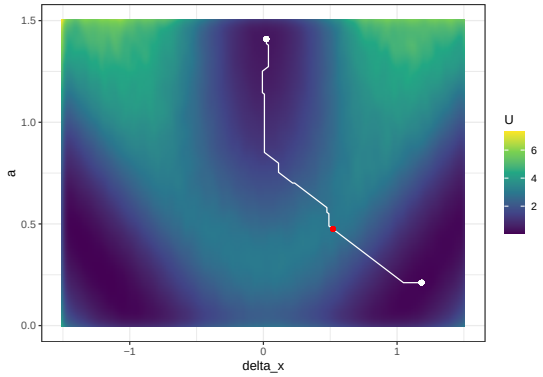

Figure 6: The landscape for the gene expression model. The local minima are marked as white dots, the saddle points are marked as red dots, and the MEPs are marked as white lines.

```
#> delta_U_start   delta_U_end
#>      2.506326      2.810604
```

The local minima, the saddle point, and the MEP can be added to the landscape with `autolayer()`, shown in Figure 6.

```
plot(l_single_gene_3d, 2) + autolayer(b_single_gene_3d)
```

Next, we use multiple simulations to investigate the influence of two parameters, k and b, on the stability of the system. As explained above, the parameter b represents the strength of mutual-inhibition between the two genes. Therefore, as b increases, we expect the differentiation is more extreme, that is, the cell is more likely to develop into one of the two cell types with very different gene expression levels. The valleys in the landscape representing the two types will become further apart and the barrier becomes clearer. The parameter k represents the speed of degradation of the gene products. As k increases, the gene products degrade faster, and this effect is more pronounced when the gene products are at high levels. Therefore, the dominant gene will express at a less extreme level, and we expect that the two valleys become closer, and the barrier will become less clear as k increases.

We use the batch simulation functions of **simlandr**. First, we create the argument set for the batch simulation. This specifies the parameters to be varied. We examined three b values, 0.5, 1, 1.5, and three k values, 0.5, 1, 1.5, which form 9 possible combinations.

```
batch_arg_set_gene <- new_arg_set()
batch_arg_set_gene <- batch_arg_set_gene |>
    add_arg_ele(arg_name = "parameter", ele_name = "b", start = 0.5, end = 1.5,
        by = 0.5) |>
    add_arg_ele(arg_name = "parameter", ele_name = "k", start = 0.5, end = 1.5,
        by = 0.5)
batch_grid_gene <- make_arg_grid(batch_arg_set_gene)
```

We then perform the batch simulation with the `batch_simulation()` function. Here, we specify the simulation function to be used, which is similar to the single simulation we showed above, together with the data wrangling procedure. The simulation function is defined with the `sim_fun` argument. We also use `bigmemory = TRUE` to store the simulation results in the `hash_big_matrix` format, which is more memory-efficient.

```
batch_output_gene <- batch_simulation(batch_grid_gene, sim_fun = function(parameter) {
    b <- parameter[["b"]]
    k <- parameter[["k"]]
    drift_gene <- c(rlang::expr(z * x^(!!n)/((!!S)^(!!n) + x^(!!n)) + (!!b) *
```

```
        (!!S)^(!!n)/((!!S)^(!!n) + y^(!!n)) - (!!k) * x), rlang::expr(z *
        y^(!!n)/((!!S)^(!!n) + y^(!!n)) + (!!b) * (!!S)^(!!n)/((!!S)^(!!n) +
        x^(!!n)) - (!!k) * y), rlang::expr(-(!!lambda) * z)) |>
        as.expression()
    set.seed(1614)
   single_output_gene <- sim_SDE(drift = drift_gene, diffusion = diffusion_gene,
        N = 1e+06, M = 10, Dt = 0.1, x0 = c(0, 0, 1), keep_full = FALSE)
    single_output_gene2 <- do.call(rbind, single_output_gene)
   single_output_gene2 <- cbind(single_output_gene2[, "X"] - single_output_gene2[,
        "Y"], single_output_gene2[, "Z"])
    colnames(single_output_gene2) <- c("delta_x", "a")
    single_output_gene2
}, bigmemory = TRUE)
```

If the output is saved in an RDS file, upon next use, it can be read as follows.

```
saveRDS(batch_output_gene, "batch_output_gene.RDS")
batch_output_gene <- readRDS("batch_output_gene.RDS") |>
    attach_all_matrices()
```

We then make the 3D matrix for the batch output, using `make_3d_matrix()`.

```
l_batch_gene_3d <- make_3d_matrix(batch_output_gene, x = "delta_x", y = "a",
    cols = "b", rows = "k", lims = c(-5, 5, -0.5, 2), h = 0.005, Umax = 8,
    kde_fun = "ks", individual_landscape = TRUE)
```

For the barrier calculation step, the start and end points of the barrier may be different for each landscape plot. The following code shows how to create a barrier grid for each landscape plot. First, we create a barrier grid template using the function `make_barrier_grid_3d()`. Next, we modify the barrier grid template to create a barrier grid for the landscape plot.

```
make_barrier_grid_3d(batch_grid_gene, start_location_value = c(0, 1.5),
   end_location_value = c(1, -0.5), start_r = 1, end_r = 1, print_template = TRUE)
```

```
#> structure(list(start_location_value = list(c(0, 1.5), c(0, 1.5
#> ), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0,
#> 1.5), c(0, 1.5)), start_r = list(c(1, 1), c(1, 1), c(1, 1), c(1,
#> 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1)), end_location_value = list(
#>     c(1, -0.5), c(1, -0.5), c(1, -0.5), c(1, -0.5), c(1, -0.5
#>     ), c(1, -0.5), c(1, -0.5), c(1, -0.5), c(1, -0.5)), end_r = list(
#>     c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1,
#>     1), c(1, 1), c(1, 1))), row.names = c(NA, -9L), class = c("arg_grid",
#> "data.frame"))
```

```
#>   ele_list   b   k start_location_value start_r end_location_value end_r
#> 1 0.5, 0.5 0.5 0.5            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 2 1.0, 0.5 1.0 0.5            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 3 1.5, 0.5 1.5 0.5            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 4 0.5, 1.0 0.5 1.0            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 5     1, 1 1.0 1.0            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 6 1.5, 1.0 1.5 1.0            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 7 0.5, 1.5 0.5 1.5            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 8 1.0, 1.5 1.0 1.5            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 9 1.5, 1.5 1.5 1.5            0.0, 1.5    1, 1          1.0, -0.5  1, 1
```

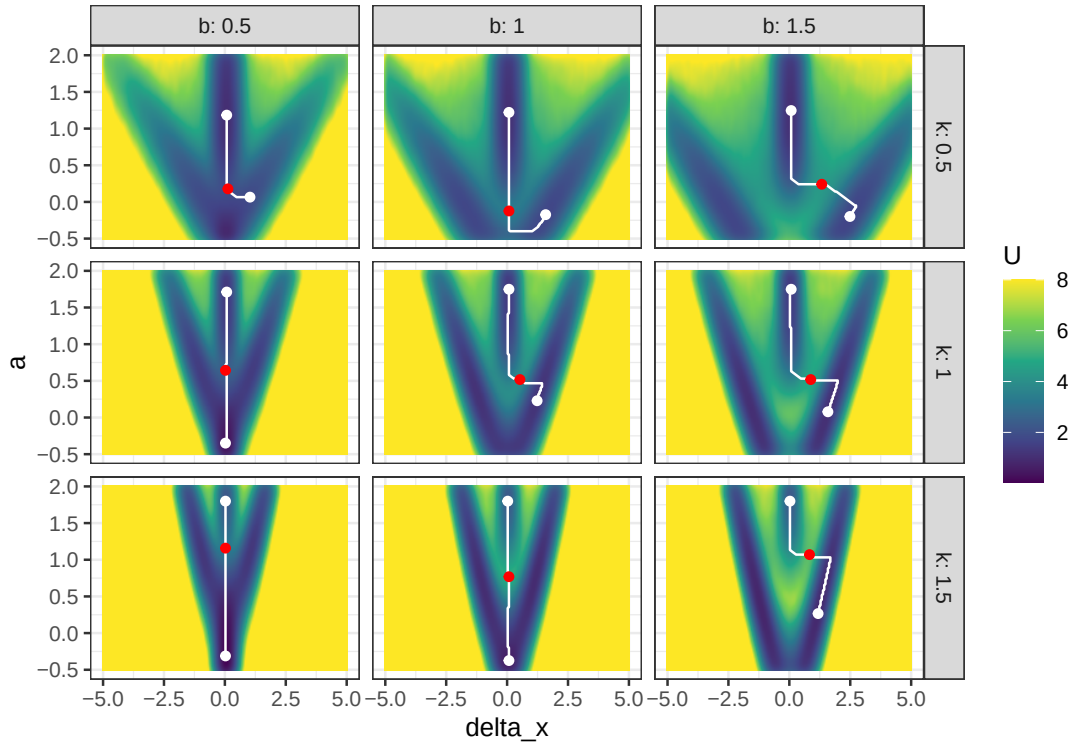

Figure 7: The landscape for the gene expression model of different b and k values. The local minima are marked as white dots, the saddle points are marked as red dots, and the MEPs are marked as white lines.

```
bg_gene <- make_barrier_grid_3d(batch_grid_gene, df = structure(list(start_location_value = list(c(0
    1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5),
    c(0, 1.5), c(0, 1.5)), start_r = list(c(0.2, 1), c(0.2, 1), c(0.2,
    1), c(0.2, 0.5), c(0.2, 0.5), c(0.2, 0.5), c(0.2, 0.3), c(0.2, 0.3),
    c(0.2, 0.3)), end_location_value = list(c(2, 0), c(2, 0), c(2, 0),
    c(1, 0), c(1, 0), c(1, 0), c(1, 0), c(1, 0), c(1, 0)), end_r = list(c(1,
    1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1),
    c(1, 1))), row.names = c(NA, -9L), class = c("arg_grid", "data.frame")))
```

With the barrier grid template, we can calculate the barrier for each landscape plot.

```
b_batch_gene_3d <- calculate_barrier(l_batch_gene_3d, bg = bg_gene)
```

If the barrier grid was not needed, the following code can be used to calculate the barrier.

```
b_batch_gene_3d <- calculate_barrier(l_batch_gene_3d, start_location_value = c(0,
    1.5), end_location_value = c(1, 0), start_r = 1, end_r = 1)
```

The resulting landscapes and the MEPs between states are shown in Figure 7.

```
plot(l_batch_gene_3d) + autolayer(b_batch_gene_3d)
```

From the landscapes, it is clear that increasing b, which represents higher strength of gene mutual prohibition, makes the two differentiated states further apart from each other. Increasing k, which represents faster degradation, makes the undifferentiated state disappear earlier, thus makes the differentiation earlier. When b is low enough and k is high enough, there is no differentiation anymore because the two differentiated states merge together and form a more stable state at $\Delta x = 0$. In this case, there is no actual differentiation in the system, but only a one-to-one conversion of cell types. Only when b is high enough and k is low enough is it possible for the cell to differentiate into two types.

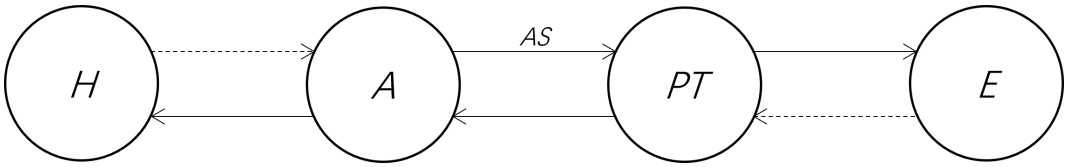

Figure 8: A graphical illustration of the relationships between several important psychological variables in the panic disorder model. Solid arrows represent positive relationships and dashed arrows represent negative relationships.

3.2 Example 2: panic disorder model

The second example we use is the panic disorder model proposed by Robinaugh et al. (2024). The model is implemented in the **PanicModel** package (`https://github.com/jmbh/PanicModel/`). It contains 12 variables and 33 parameters and also involves history dependency and non-differentiable formula (such as if-else conditions) to model the complex interplay of individual and environmental elements in different time scales. The most important variables of the model are the physical arousal (A) of a person (e.g., heart beat, muscle tension, sweating), the person's perceived threat (PT, how dangerous the person cognitively evaluates the environment), and the person's tendency to escape from the situation (E). The core theoretical idea of the model is that physical arousal and perceived threat of a person may strengthen each other in certain circumstances, leading to sudden increases in both variables, manifesting as panic attacks. The tendency that a person tends to use physical arousal as cognitive evidence of threat is represented by another variable, arousal schema S. A comprehensive introduction of the model is beyond the scope of the current article, and we would like to refer interested readers to Robinaugh et al. (2024). Here, to simplify the context, we assume that AS does not change over time, and no psychotherapy is being administered. We focus on the influence of AS on the system's stability represented by A and PT. A graphical illustration of several core variables of this model is shown in Figure 8 (adapted from Cui et al. (2023b)).

To construct the potential landscapes for this model, we first create a function that performs a simulation using the `simPanic()` function from **PanicModel**. This is required as we need to modify some default options for illustration.

```
library(PanicModel)

sim_fun_panic <- function(x0, par) {

    # Change several default parameters
    pars <- pars_default
    # Increase the noise strength to improve sampling efficiency
    pars$N$lambda_N <- 200
    # Make S constant through the simulation
    pars$TS$r_S_a <- 0
    pars$TS$r_S_e <- 0

    # Specify the initial values of A and PT according to the format
    # requirement by `multi_init_simulation()`, while the other
    # variables use the default initial values.
    initial <- initial_default
    initial$A <- x0[1]
    initial$PT <- x0[2]

    # Specify the value of S according to the format requirement by
    # `batch_simulation()`.
    initial$S <- par$S
```

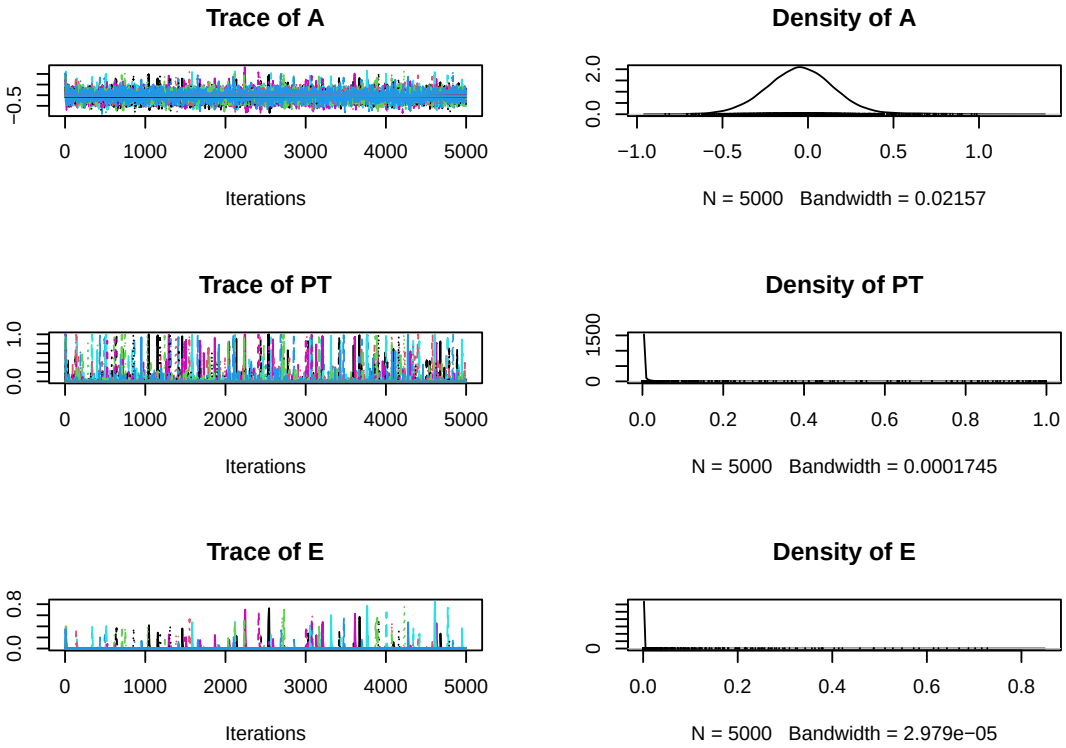

Figure 9: The convergence check result for the simulation of the panic disorder model.

```
    # Extract the simulation output from the result by simPanic().
    # Only keep the core variables.
   return(as.matrix(simPanic(1:5000, initial = initial, parameters = pars)$outmat[,
       c("A", "PT", "E")]))
}
```

We then perform a single simulation from multiple starting points. To speed up the simulation, we use parallel computing.

```
future::plan("multisession")
set.seed(1614, kind = "L'Ecuyer-CMRG")
single_output_panic <- multi_init_simulation(sim_fun = sim_fun_panic, range_x0 = c(0,
    1, 0, 1), R = 4, par = list(S = 0.5))
```

The convergence check results of the simulation, shown in Figure 9, indicate that the time series of the first 100 data points are strongly influenced by the choice of initial value. Therefore, we remove the first 100 data points in the following analysis.

```
plot(single_output_panic)
```

We then create the 3D landscape for the panic disorder model, shown in Figure 10. The landscape shows that the system has two stable states, which are represented by the valleys in the landscape. The system can be trapped in these valleys, leading to different levels of physical arousal and perceived threat. The valley with higher potential value represents a state with higher physical arousal and perceived threat, which corresponds to a panic attack. In contrast, the valley with lower potential value represents a state with lower physical arousal and perceived threat, which corresponds to a healthy state.

We now investigate the effect of the parameter S on the potential landscape. This parameter represents the tendency that a person considers physical arousal as a sign of danger. Therefore, we expect that higher S will stabilize the panic state and destabilize the healthy state.

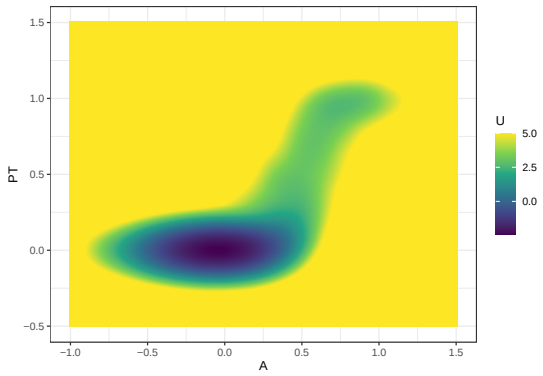

Figure 10: The 3D landscape (potential value as color) for the panic disorder model

We perform a batch simulation with varying S values to construct the potential landscapes for different S values. This, again, starts with the creation of a grid of parameter values.

```
batch_arg_grid_panic <- new_arg_set() |>
    add_arg_ele(arg_name = "par", ele_name = "S", start = 0, end = 1, by = 0.5) |>
     make_arg_grid()
```

We then perform the batch simulation using parallel computing.

```
future::plan("multisession")
set.seed(1614, kind = "L'Ecuyer-CMRG")
batch_output_panic <- batch_simulation(batch_arg_grid_panic, sim_fun = function(par) {
    multi_init_simulation(sim_fun_panic, range_x0 = c(0, 1, 0, 1), R = 4,
        par = par) |>
        window(start = 100)
})
```

The 3D landscapes for different S values are shown in Figure 11. The landscapes show that the system only has one stable state when S is low, but has two stable states when S is high. The stability of the panic state also increases when S is higher. This indicates that a higher S value corresponds to a higher risk of panic attacks.

```
l_batch_panic_3d <- make_3d_matrix(batch_output_panic, x = "A", y = "PT",
    cols = "S", h = 0.005, lims = c(-1, 1.5, -0.5, 1.5))
plot(l_batch_panic_3d)
```

4 Discussion

Potential landscapes can show the stability of states for a dynamical system in an intuitive and quantitative way. They are especially informative for multistable systems. In this article, we illustrated how to construct potential landscapes using **simlandr**. The potential landscapes generated by **simlandr** are based on the steady-state distribution of the system, which is in turn estimated using Monte Carlo simulation. Compared to analytic methods, Monte Carlo estimation is more flexible and thus more applicable for complex models. The flexibility comes together with a higher demand for time and storage, which is necessary to make the estimation precise enough. The hash_big_matrix class partly solved this problem by dumping the memory storage to hard disk space. Also, it is important that the simulation function itself is efficient enough. The functions `sim_SDE()` and `multi_init_simulation()` make use of the efficient simulations provided by **Sim.DiffProc** (Guidoum and Boukhetala, 2020) and the parallel computing with the **future** framework

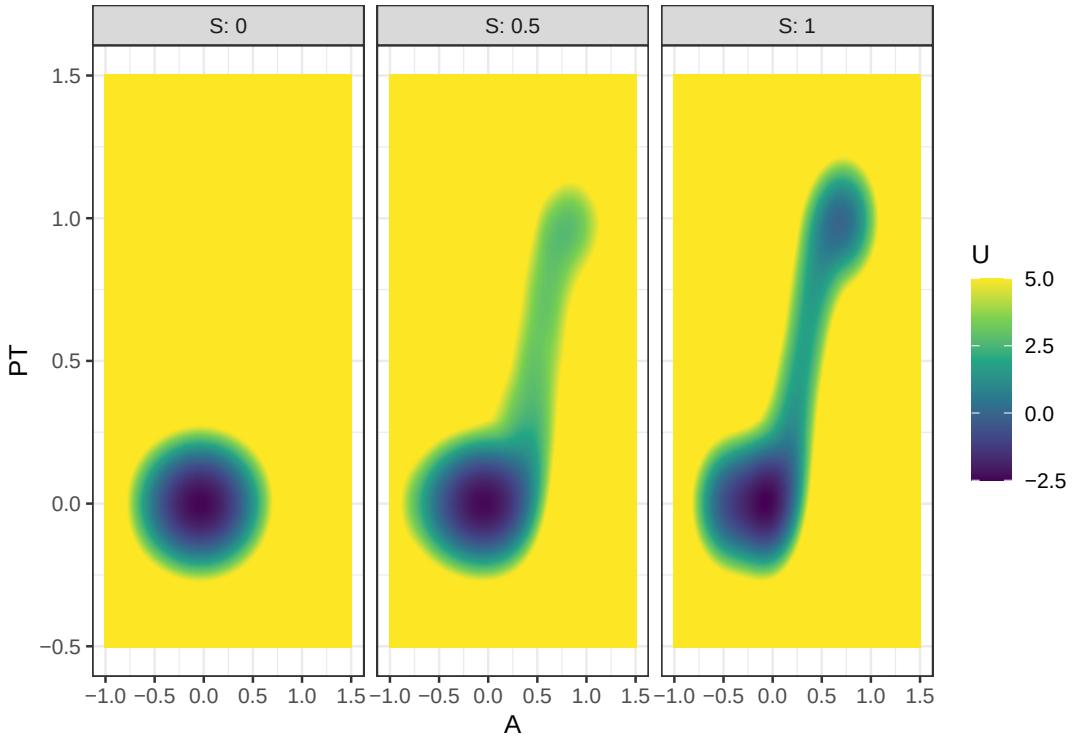

Figure 11: The landscape for the panic disorder model of different S values. Two landscapes are shown for different variable combinations, A and PT, or A and E.

(Bengtsson, 2021). For customized simulation functions, there are also multiple approaches that can be used to improve the performance, for which we refer interested readers to Wickham (2019). In Supplementary Materials A, we provide a benchmark of the typical time and memory usage of the procedures in **simlandr**. From there, we can see that time and memory usage are acceptable in most cases on a personal computer. When the transition between attractors is rare, the `multi_init_simulation()` function may help to speed up the convergence, and more advanced sampling methods like importance sampling or rare event sampling may be needed in more complex situations. The detailed ways to implement such methods are highly dependent on the specific model and are beyond the scope of this package. We direct interested readers to Rubino and Tuffin (2009) and Kloek and van Dijk (1978) for a comprehensive review of rare event simulation methods. Nevertheless, the landscape construction functions in **simlandr** allow users to provide weights for the simulation results, which can be used to adjust the sampling distribution.

In addition, the length of the simulation and the choice of noise strength may also have an important influence on the results. If the length is too short, the density estimation will be inaccurate, resulting in rugged landscapes. If the length is too long, the simulation part would require more computational resources, which is not always realistic. If the noise is too weak, the system may not be able to converge in a reasonable time, resulting in problems in convergence checks, overly noisy landscapes, or failure to show valleys that are theoretically present. If the noise is too strong, the simulation may be unstable and the boundaries between valleys may be blurry. In Supplementary Materials B, we showed the influence of simulation length and noise strength on the landscape output. With some theoretical expectation of the system's behavior, it is not difficult to spot that the simulation is too short, or the noise level might be unsuitable. In that case, some adjustments are required before the landscape can be well constructed.

All landscape construction and barrier calculation functions in **simlandr** contain both visual aids and numerical data that can be used for further processing. The html plots based on **plotly** are more suitable for interactive illustrations, while it is also possible to export them to static plots using `plotly::orca()`. The **ggplot2** plots are readily usable for flat printing.

We also want to note some limitations of the potential landscape generated by **simlandr**. First, the generalized potential landscape is not a complete description of all dynamics in a system. It emphasizes the stability of different states by filtering out other dynamical information. Some behaviors are not possible in gradient systems (e.g., oscillations and loops), thus cannot be shown in a potential landscape (Zhou and Li, 2016). Second, since the steady-state distribution is estimated using a kernel smoothing method, which depends on stochastic simulations, the resulting potential function may not be highly accurate. Its accuracy is further affected by the choice of kernel bandwidth and noise strength. This issue is particularly pronounced at valley edges, where fewer samples are available for estimation. Similar limitations apply to MEP calculations, as they are derived from the generalized landscape rather than the original dynamics. Therefore, we do not recommend directly interpreting the potential function or barrier height results for applications requiring high precision. Instead, the potential landscape is best used as a semi-quantitative tool to gain insights into the system's overall behavior, guide further analysis, and compare system behavior under different parameter settings, provided the same simulation and kernel estimation conditions are used. The examples in this article illustrated some typical use cases we recommend.

5 Availability and future directions

This package is publicly available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=simlandr`, under GPL-3 license. The results in the current article were generated with **simlandr** 0.4.0 version. R script to replicate all the results in this article can be found in the supporting information.

The barrier height data calculated by **simlandr** can also be further analyzed and visualized. For example, sometimes it is helpful to look into how the barrier height changes with varying parameters (e.g., Cui et al. (2023b)). We encourage users to explore other ways of analyzing and visualizing the various results provided by **simlandr**.

The method we chose for **simlandr** is not the only possible one. The generalized landscape by Wang et al. (2008), which we implemented, is more flexible and emphasizes the possibility that the system is in a specific state, while other methods may have other strengths (e.g., the method by Rodríguez-Sánchez (2020) emphasizes the gradient part of the vector field, and the method by Moore et al. (2016) emphasizes the possibility of transition processes under small noise). We look forward to future theoretical and methodological developments in this direction.

6 Acknowledgments

TL was supported by the NSFC under Grant No. 11825102 and the Beijing Academy of Artificial Intelligence (BAAI). ALA and MO were supported by an NWO VIDI grant, Grant No. VI.Vidi.191.178.

References

P. Ao. Potential in stochastic differential equations: Novel construction. *Journal of Physics A: Mathematical and General*, 37(3):L25–L30, 2004. ISSN 0305-4470. doi: 10.1088/0305-4470/37/3/L01. [p2]

H. Bengtsson. A unifying framework for parallel and distributed processing in R using futures. *The R Journal*, 13(2):208–227, 2021. doi: 10.32614/RJ-2021-048. [p3, 15]

J. E. Chacón and T. Duong. *Multivariate Kernel Smoothing and Its Applications*. Chapman and Hall/CRC, New York, May 2018. ISBN 978-0-429-48557-2. doi: 10.1201/9780429485572. [p5]

J. Cui, A. Lichtwarck-Aschoff, and F. Hasselman. Comments on "Climbing Escher's Stairs: A way to approximate stability landscapes in multidimensional systems", Dec. 2023a. [p2]

J. Cui, A. Lichtwarck-Aschoff, M. Olthof, T. Li, and F. Hasselman. From metaphor to computation: Constructing the potential landscape for multivariate psychological formal models. *Multivariate Behavioral Research*, 58(4):743–761, 2023b. doi: 10.1080/00273171.2022.2119927. [p3, 5, 6, 12, 16]

D. Dahiya and M. Cameron. Ordered line integral methods for computing the quasi-potential. *Journal of Scientific Computing*, 75(3):1351–1384, 2018. ISSN 1573-7691. doi: 10.1007/s10915-017-0590-9. [p2]

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 (1):269–271, 1959. doi: 10.1007/BF01386390. [p5]

W. E and E. Vanden-Eijnden. Transition-path theory and path-finding algorithms for the study of rare events. *Annual Review of Physical Chemistry*, 61(1):391–420, 2010. ISSN 0066-426X. doi: 10.1146/annurev.physchem.040808.090412. [p5]

D. Eddelbuettel, A. Lucas, J. Tuszynski, H. Bengtsson, S. Urbanek, M. Frasca, B. Lewis, M. Stokely, H. Muehleisen, D. Murdoch, J. Hester, W. Wu, Q. Kou, T. Onkelinx, M. Lang, V. Simko, K. Hornik, R. Neal, K. Bell, M. de Queljoe, I. Suruceanu, B. Denney, D. Schumacher, and W. Chang. *digest: Create Compact Hash Digests of R Objects*, 2021. [p4]

M. Freidlin and A. Wentzell. *Random Perturbations of Dynamical Systems*. Springer, Berlin, 3rd edition, 2012. doi: 10.1007/978-3-642-25847-3. [p2]

A. C. Guidoum and K. Boukhetala. Performing parallel Monte Carlo and moment equations methods for Itô and Stratonovich stochastic differential systems: R package Sim.DiffProc. *Journal of Statistical Software*, 96:1–82, Nov. 2020. ISSN 1548-7660. doi: 10.18637/jss.v096.i02. [p3, 14]

A. M. Hayes and L. A. Andrews. A complex systems approach to the study of change in psychotherapy. *BMC Medicine*, 18(1):197, July 2020. ISSN 1741-7015. doi: 10.1186/s12916-020-01662-2. [p1]

M. Heymann and E. Vanden-Eijnden. Pathways of maximum likelihood for rare events in nonequilibrium systems: Application to nucleation in the presence of shear. *Physical Review Letters*, 100(14):140601, 2008. doi: 10.1103/PhysRevLett.100.140601. [p5]

M. J. Kane, J. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. doi: 10.18637/jss.v055.i14. [p4]

T. Kloek and H. K. van Dijk. Bayesian estimates of equation system parameters: An application of integration by Monte Carlo. *Econometrica*, 46(1):1–19, 1978. ISSN 0012-9682. doi: 10.2307/1913641. [p15]

K. A. Lamothe, K. M. Somers, and D. A. Jackson. Linking the ball-and-cup analogy and ordination trajectories to describe ecosystem stability, resistance, and resilience. *Ecosphere*, 10(3):e02629, 2019. ISSN 2150-8925. doi: 10.1002/ecs2.2629. [p1]

C. M. Moore, C. R. Stieha, B. C. Nolting, M. K. Cameron, and K. C. Abbott. QPot: An R package for stochastic differential equation quasi-potential analysis. *The R Journal*, 8(2):19–38, 2016. doi: 10.32614/RJ-2016-031. [p2, 16]

M. Olthof, F. Hasselman, F. Oude Maatman, A. M. T. Bosman, and A. Lichtwarck-Aschoff. Complexity theory of psychopathology. *Journal of Psychopathology and Clinical Science*, 132 (3):314–323, 2023. doi: 10.1037/abn0000740. [p1]

M. Plummer, N. Best, K. Cowles, and K. Vines. Coda: Convergence diagnosis and output analysis for mcmc. *R News*, 6(1):7–11, 2006. URL https://journal.r-project.org/articles/RN-2006-002/RN-2006-002.pdf. [p4]

D. J. Robinaugh, J. M. B. Haslbeck, L. J. Waldorp, J. J. Kossakowski, E. I. Fried, A. J. Millner, R. J. McNally, O. Ryan, J. de Ron, H. L. J. van der Maas, E. H. van Nes, M. Scheffer, K. S. Kendler, and D. Borsboom. Advancing the network theory of mental disorders: A computational model of panic disorder. *Psychological Review*, 131(6):1482–1508, 2024. doi: 10.1037/rev0000515. [p6, 12]

P. Rodríguez-Sánchez. Pabrod/rolldown: Post-publication update, 2020. [p2, 16]

P. Rodríguez-Sánchez, E. H. van Nes, and M. Scheffer. Climbing Escher's stairs: A way to approximate stability landscapes in multidimensional systems. *PLOS Computational Biology*, 16(4):e1007788, 2020. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1007788. [p2]

G. Rubino and B. Tuffin. *Rare Event Simulation Using Monte Carlo Methods*. John Wiley & Sons, Incorporated, Newark, UNITED KINGDOM, 2009. ISBN 978-0-470-74541-0. doi: 10.1002/9780470745403. [p15]

C. Sievert. *Interactive Web-Based Data Visualization with R, Plotly, and Shiny*. Chapman and Hall/CRC, 2020. ISBN 978-1-138-33145-7. URL https://plotly-r.com. [p5]

P. S. Stumpf, F. Arai, and B. D. MacArthur. Modeling stem cell fates using non-Markov processes. *Cell Stem Cell*, 28(2):187–190, 02 2021. doi: 10.1016/j.stem.2021.01.009. [p1]

C. H. Waddington. *Principles of Development and Differentiation*. Macmillan, New York, 1966. [p1]

G. Wan, S. J. Avis, Z. Wang, X. Wang, H. Kusumaatmaja, and T. Zhang. Finding transition state and minimum energy path of bistable elastic continua through energy landscape explorations. *Journal of the Mechanics and Physics of Solids*, 183:105503, Feb. 2024. ISSN 0022-5096. doi: 10.1016/j.jmps.2023.105503. [p3]

J. Wang, L. Xu, and E. Wang. Potential landscape and flux framework of nonequilibrium networks: Robustness, dissipation, and coherence of biochemical oscillations. *Proceedings of the National Academy of Sciences*, 105(34):12271–12276, 2008. ISSN 0027-8424. doi: 10.1073/pnas.0800579105. [p2, 5, 16]

J. Wang, K. Zhang, L. Xu, and E. Wang. Quantifying the Waddington landscape and biological paths for development and differentiation. *Proceedings of the National Academy of Sciences*, 108(20):8257–8262, 2011. ISSN 0027-8424. doi: 10.1073/pnas.1017017108. [p1, 6, 7]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p5]

H. Wickham. Improving performance. In *Advanced R*, pages 531–546. Chapman and Hall/CRC, Boca Raton, 2nd edition, 2019. URL https://adv-r.hadley.nz/. [p15]

P. Zhou and T. Li. Construction of the landscape for multi-stable systems: Potential landscape, quasi-potential, a-type integral and beyond. *The Journal of Chemical Physics*, 144(9): 094109, 2016. ISSN 0021-9606. doi: 10.1063/1.4943096. [p2, 3, 16]

Jingmeng Cui
University of GroningenRadboud University
Faculty of Behavioural and Social Sciences, Groningen, the Netherlands
Behavioural Science Institute, Nijmegen, the Netherlands
ORCiD: 0000-0003-3421-8457
jingmeng.cui@rug.nl

Merlijn Olthof
University of GroningenRadboud University

Faculty of Behavioural and Social Sciences, Groningen, the Netherlands
Behavioural Science Institute, Nijmegen, the Netherlands
ORCiD: 0000-0002-5975-6588
merlijn.olthof@ru.nl

Anna Lichtwarck-Aschoff
University of GroningenRadboud University
Faculty of Behavioural and Social Sciences, Groningen, the Netherlands
Behavioural Science Institute, Nijmegen, the Netherlands
ORCiD: 0000-0002-4365-1538
a.lichtwarck-aschoff@rug.nl

Tiejun Li
Peking University
(corresponding author) LMAM and School of Mathematical Sciences, Beijing, China
ORCiD: 0000-0002-2086-1620
tieli@pku.edu.cn

Fred Hasselman
Radboud University
(corresponding author) Behavioural Science Institute, Nijmegen, the Netherlands
ORCiD: 0000-0003-1384-8361
fred.hasselman@ru.nl