Dyson SIS Code Tutorial and Reproducibility Notes

Tim Van Wesemael, Gilberto Nakamura

September 2025

Contents

1 Overview **2**

2 Quick start **2**
2.1 Prerequisites 2
2.2 Reproducing all figures 2

3 Core API and Usage **2**
3.1 API Overview 2
3.2 Workflow 4

4 Scripts and outputs **4**
4.1 Matrix structure 4
4.2 Entropy figures (Figure 1) 4
4.3 Statistics over time (Figure 3) 4
4.4 Transition curves (Figure 4) 5
4.5 Steady-state comparison (Figure 5) 5
4.6 Algorithm convergence (Figure 6) 5

5 Figure Gallery **5**

6 Produced Figures **6**

1 Overview

This document accompanies the *dyson-code* repository and explains how to reproduce the figures and explore some model variations. The code implements a Susceptible-Infected-Susceptible (SIS) model on a graph and computes a Dyson similarity transformation that approximately symmetrizes the generator matrix. The primary entry points are in `src/dyson.jl` and runnable scripts in `scripts/` that generate the paper figures into `results/figures/`.

Key components

- `default_initialisation()` prepares a default complete graph with 6 nodes, default rates, and either loads or computes the Dyson transform.
- `get_SIS_H(A, beta, gamma; incl_disease_free)` builds the SIS generator given adjacency matrix `A` and rates.
- `find_hermitian(H; tol, kmax)` iteratively constructs h, η with reduced asymmetry.
- `simulate(H, eta; symmetrize, tspan, saveat)` integrates $x'(t) = -H'x$ with optional similarity transform.
- Multiple functions to provide observables and statistics of the system.

2 Quick start

2.1 Prerequisites

- Julia 1.11.7 with the packages pinned by `Project.toml`

2.2 Reproducing all figures

From the repository root:

```
$ bash run_all.sh
```

This runs every script in `scripts/` and writes figures to `results/figures/`.

3 Core API and Usage

3.1 API Overview

The core functionality lives in `src/dyson.jl`. Below is a narrative overview of the exported functions used throughout the scripts, how to apply them in your own work, and when to prefer each. For a technical summary of arguments and return values, refer to the in-code documentation (docstrings) in `src/dyson.jl`.

`default_initialisation()` Loads cached defaults or generates them if missing: a complete graph with six nodes, SIS generator `H`, Hermitized matrix `h`, transformation η, rates β, γ, a flag `incl_disease_free`, and the convergence `errors`. Use this at the top of a script to get a reproducible baseline without waiting for recomputation. Artifacts are cached under `results/intermediate/`. If you want to repeat the experiments for an other system (dynamics, or network), this function serves as a great jumping-off point.

```
name, g, A, H, h, eta, beta, gamma, incl_disease_free, errors = default_initialisation()
```

get_SIS_H(A, beta, gamma; incl_disease_free=false) Constructs the SIS generator consistent with the chosen graph (represented by adjacency matrix `A`) and rates. Rows sum to zero by construction, and the disease-free state can be excluded, such that only a single steady-state exists. Use this when sweeping β/γ (see `scripts/transition.jl`) or when you want to try a different graph.

```
H = get_SIS_H(A, beta, gamma; incl_disease_free=false)
```

find_hermitian(H; tol, kmax, verbose=false) Iteratively finds a similarity transform that reduces the asymmetry of `H`, returning the transformed matrix `h`, the transform η, and an error history. Use when you need a near-Hermitian representation for analysis or optimization (cf. `scripts/steady_state.jl`).

```
h, eta, errors = find_hermitian(Matrix(H); kmax=5_000)
```

SIS_initial_state(n) Provides the default initial state used in simulations: one infected individual mapped by η. Override this if you need a different starting distribution (e.g., multiple infections or a distribution over states).

```
u0 = SIS_initial_state(size(H, 1))
```

simulate(H, eta; symmetrize, tspan, saveat, initial_func) Integrates the dynamics $x'(t) = -H'x$ under the similarity transform η. Set `symmetrize=true` to simulate the symmetrized transformed matrix. This changes the system depending on how non-Hermitian `H` is. However it ensures that the dynamics are reversible. Supply a custom `initial_condition` to explore different initial states.

```
sol = simulate(H, I; tspan=(0.0, 5.0), saveat=0.05)        # original system
sol_h = simulate(H, eta; symmetrize=true)                      # transformed
sol_custom = simulate(H, eta; initial_condition=(H,eta)->(eta*rand(size(H,1))))
```

get_steady_state(H; eta=I, symmetrize=false) Computes a steady state by forward integration. In the Hermitized frame, combine with η for the transformed steady state as in `scripts/steady_state.jl`.

```
P_ss = get_steady_state(H)
phi_ss = get_steady_state(h; symmetrize=false)  # pass eta via keyword in your code
```

infectious_proportion_O(n; incl_disease_free=false) Returns the observable measuring the infected fraction. Use with `observe` on either the original state distribution `p` or the transformed state ϕ with metric Ω.

```
  O = infectious_proportion_O(n; incl_disease_free=false)
  I_mean = observe(O, p)
  Omega = inv(eta*eta')
  I_mean_transformed = observe(eta*O*inv(eta), phi, Omega)
```

observe(O, p) and observe(O, phi, Omega) Two observation modes: apply `O` to the probability vector `p`, or to a transformed state ϕ with the appropriate metric Ω. See `scripts/statistics.jl` for mean/variance time series in both frames.

shannon_entropy(p) and renyi_entropy(p) Entropy measures used for the entropy figures and transition curves. Apply directly to state vectors from simulations or steady states. For details (e.g., numerical safeguards near zeros), see the function docstrings.

3.2 Workflow

In practice, the workflow is: build or load `H`, obtain h, η via `find_hermitian`, simulate either original or transformed dynamics using `simulate`, and compute observables with `observe` and entropies with `shannon_entropy/renyi_entropy`. For precise signatures and edge cases, consult the code documentation in `src/dyson.jl`.

4 Scripts and outputs

Each script starts by including `dyson-setup.jl`, which activates the project, brings Dyson into scope, seeds RNG for reproducibility, and ensures `results/figures/` exists. Every script is written to reproduce the figures of the paper, using the functionality of Dyson.

4.1 Matrix structure

Script: `scripts/H_figure.jl`

- Builds a color-coded visualization of the SIS generator's sparsity and transition types.
- Saves `SIS_H_heatmap.pdf` to `results/figures/`.

Adjustable parameters (Figure 2B)

- Color scaling and palette: edit the color ranges in the script for infections/healings/diagonals.
- Graph size/structure: modify `default_initialisation()` to change `complete_graph(6)`.

4.2 Entropy figures (Figure 1)

Script: `scripts/entropy_figures.jl`

- `generate_homotopy(eta, sim_f, agg_f; steps)` creates a homotopy $\eta(\alpha) = (1-\alpha)I + \alpha\,\eta$ and aggregates an observable (by default Rényi entropy).
- `create_heatmap` saves `<name>_entropy_heatmap.pdf` and `create_entropies_figure` saves `<name>_entropy_graphs.pdf`.

Adjustable parameters

- `sim_f`: adjust to perform a homotopy over other systems than the SIS one.
- `agg_f`: change to aggregate other observables, e.g., `mean_O` or `var_O`.
- `create_entropies_figure`: modify to plot other statistics or change figure layout.
- `create_heatmap`: change the look of the figure, e.g., colormap or axis limits.

4.3 Statistics over time (Figure 3)

Script: `scripts/statistics.jl`

- Computes mean infection proportion and variance over time for original and transformed systems.
- Saves `<name>_statistics_full.pdf`.

Adjustable parameters

- `tspan`, `saveat`: change integration horizon and sampling density in `simulate` calls.
- `symmetrize`: switch between raw and symmetrized dynamics in the transformed system.
- Observable: replace `infectious_proportion_O` with other operators to track different quantities.

4.4 Transition curves (Figure 4)

Script: `scripts/transition.jl`

- Sweeps ratios β/γ and records steady-state means, variances, and entropies (original and transformed).
- Saves `<name>_transition.pdf` and a JLD2 data file under `results/intermediate/`.

Adjustable parameters

- `beta_range`: adjust the logarithmic range of β/γ values.
- `kmax`, `tol`: control depth/precision of `find_hermitian` during the sweep.

4.5 Steady-state comparison (Figure 5)

Script: `scripts/steady_state.jl`

- Compares steady-states from Dyson-optimized distribution vs. dynamic evolution; annotates KS distance.
- Saves multiple files like `<name>_SIS_ss_b001.pdf`, `..._b2.pdf`, `..._b200.pdf`.

Adjustable parameters

- Rates: set specific β values relative to γ to probe different regimes.
- `kmax`: number of Hermitization iterations taken from `default_initialisation()`.
- Plot aesthetics: toggle legend, change colors, layout, and bar alignment.

4.6 Algorithm convergence (Figure 6)

Script: `scripts/convergence.jl`

- Plots the error trajectory from `find_hermitian` in log scale.
- Saves `<name>_convergence.pdf`.

Adjustable parameters

- `kmax`: increase/decrease the number of iterations to control plot length.
- Styling: modify line width, colors, and axis labels.

5 Figure Gallery

Below we embed the figures produced by the scripts. If a file is missing, a placeholder box appears.

Missing figure: `full-6-wodf_entropy_heatmap.pdf`

Figure 1: Entropy heatmap (Figure 1).

Missing figure: `full-6-wodf_entropy_graphs.pdf`

Figure 2: Entropy time series (Figure 1).

6 Produced Figures

The scripts in `scripts/` produce the following files in `results/figures/`:

- `full-6-wodf_entropy_heatmap.pdf` — Entropy heatmap (Figure 1).
- `full-6-wodf_entropy_graphs.pdf` — Entropy time series (Figure 2).
- `SIS_H_heatmap.pdf` — Structure of the SIS generator (Figure 3).
- `full-6-wodf_statistics_full.pdf` — Mean and standard deviation over time (Figure 4).
- `full-6-wodf_transition.pdf` — Transition curves over β/γ (Figure 5).
- `full-6-wodf_SIS_ss_b001.pdf` — Steady-state bar plot at low β (Figure 6).
- `full-6-wodf_SIS_ss_b2.pdf` — Steady-state bar plot at intermediate β (Figure 7).
- `full-6-wodf_SIS_ss_b200.pdf` — Steady-state bar plot at high β (Figure 8).
- `full-6-wodf_convergence.pdf` — Convergence of the Hermitization algorithm (Figure 9).

Missing figure: `SIS_H_heatmap.pdf`

Figure 3: Structure of the SIS generator (Figure 2B).

Missing figure: `full-6-wodf_statistics_full.pdf`

Figure 4: Mean and standard deviation over time (Figure 3).

Missing figure: `full-6-wodf_transition.pdf`

Figure 5: Transition curves over β/γ (Figure 4).

Missing figure: `full-6-wodf_SIS_ss_b001.pdf`

Figure 6: Steady-state comparison at low β (Figure 5 variant).

Missing figure: `full-6-wodf_SIS_ss_b2.pdf`

Figure 7: Steady-state comparison at intermediate β (Figure 5 variant).

Missing figure: `full-6-wodf_SIS_ss_b200.pdf`

Figure 8: Steady-state comparison at high β (Figure 5 variant).

Missing figure: `full-6-wodf_convergence.pdf`

Figure 9: Convergence of the Hermitization algorithm (Figure 6).