Switching Paths

In this section we can now turn our attention to finding the switching paths connecting the stable fixed points to the saddle point. We will use our Stability Analysis from the previous section to set the boundary conditions for the switching paths. We will then use a collocation method implemented in SciPy [1] to solve the resulting boundary value problem and find the switching paths.

1. Problem Overview

1.1. Path

Our key objective is to find a path $\mathbf{Z}(t)$ that connects a **stable fixed point** $\mathbf{Z}_0$ (node or focus) to a **saddle point** $\mathbf{Z}_s$.

The key idea is to use the eigenvectors of the Jacobian at both points to set the boundary conditions. Small deviations from the fixed points can be expressed in the eigenvector basis of the Jacobian and we control the boundary conditions by specifying the coefficients of the eigenvectors in those deviations.

At each fixed point the Jacobian has a pair of incoming and outgoing eigenvectors. We simply apply the condition that at the start of the path the deviation is purely along the eigenvectors leaving the stable point and at the end of the path the deviation is purely along the eigenvectors arriving at the saddle point.

1.2. Action

With these paths we can finally calculate the corresponding actions using the formula shown in Keldysh Auxiliary Hamiltonian:

$$S_{\text{aux}} = -\int dt \Big[\dot{x}_c\, x_q + \dot{p}_c\, p_q - H(x_c, p_c, x_q, p_q)\Big].$$

Since the Hamiltonian is conservative (i.e. time independent) and the initial and final points are in the classical plane where $H = 0$, the action can be simplified to:

$$S_{\text{aux}} = -\int dt \Big[\dot{x}_c\, x_q + \dot{p}_c\, p_q\Big].$$

2. Mathematical Framework

2.1. At the Stable Fixed Point

Let $\mathbf{Z}_0$ denote the coordinates of the stable fixed point. Small deviations from $\mathbf{Z}_0$ can be expressed in the eigenvector basis of the Jacobian $J(\mathbf{Z}_0)$:

$$\Delta \mathbf{Z}(t) \equiv \mathbf{Z}(t) - \mathbf{Z}_0 = \sum_{i=1}^{4} c_i \, \mathbf{v}_i \, e^{\lambda_i t},$$

where: - $\mathbf{v}_i$ are the eigenvectors, - λ_i are the corresponding eigenvalues, ordered by their real parts with $\mathrm{Re}(\lambda_1), \mathrm{Re}(\lambda_2) < 0$ (stable) and $\mathrm{Re}(\lambda_3), \mathrm{Re}(\lambda_4) > 0$ (unstable), - c_i are coefficients determined by the initial displacement.

2.2. At the Saddle Point

- Let $\mathbf{Z}_s$ denote the coordinates of the saddle point.
- Near $\mathbf{Z}_s$, the deviation is similarly expressed in terms of the eigenvectors of the Jacobian $J(\mathbf{Z}_s)$:

$$\Delta \mathbf{Z}(t) \equiv \mathbf{Z}(t) - \mathbf{Z}_s = \sum_{j=1}^{4} d_j \, \mathbf{u}_j \, e^{\mu_j t},$$

where: - $\mathbf{u}_j$ are the eigenvectors at the saddle, - μ_j are the corresponding eigenvalues, ordered by their real parts with $\mathrm{Re}(\mu_1), \mathrm{Re}(\mu_2) < 0$ (stable) and $\mathrm{Re}(\mu_3), \mathrm{Re}(\mu_4) > 0$ (unstable), - d_j are coefficients that characterize the deviation.

---

3. Formulating the Boundary Value Problem

3.1. Physical Interpretation of Boundary Conditions

The switching trajectory represents the optimal (least-action) path connecting a metastable fixed point to the saddle point. It is a solution to the equations of motion derived from the auxiliary Hamiltonian as seen in FixedPoints. The eigenvector analysis from the Stability Analysis then provides natural boundary conditions:

Stable Fixed Point $\mathbf{Z}_0$

At $t \to -\infty$ (approaching the stable fixed point) we have:

$$\mathbf{Z}(t) \to \mathbf{Z}_0 \quad \Longrightarrow \quad \Delta \mathbf{Z}(t) \approx \sum_{i} c_i \, \mathbf{v}_i \, e^{\lambda_i t},$$

Since the trajectory must *depart* within the unstable subspace, the initial deviation $\Delta \mathbf{Z}(t)$ aligns with eigenvectors having $\mathrm{Re}(\lambda_i) > 0$ and we can set $c_0 = 0$ and $c_1 = 0$.

Depending on exactly which initial direction we choose, the system will follow a different path. Our goal is to the path which connects to the saddle point.

Saddle Point $\mathbf{Z}_s$

At $t \to +\infty$ (approaching the saddle point):

$$\mathbf{Z}(t) \to \mathbf{Z}_s \quad \Longrightarrow \quad \Delta\mathbf{Z}(t) \approx \sum_j d_j\, \mathbf{u}_j\, e^{\mu_j t},$$

Since the trajectory must *arrive* within the stable subspace, the final deviation $\Delta\mathbf{Z}(t)$ lies in the subspace of eigenvectors with $\mathrm{Re}(\mu_j) < 0$. Therefore we can set $d_2 = 0$ and $d_3 = 0$. If we have chosen the correct initial trajectory, the system will follow a path that naturally arrives at the saddle point within this subspace.

3.2 Projection

To enforce our asymptotic boundary conditions at finite times t_i and t_f, we must project the deviations onto the eigenbasis of the Jacobian at the fixed points. This is achieved by using the left eigenvectors to extract the coefficients in the expansion of the deviation in terms of the right eigenvectors.

Let $\Delta Z(t)$ be the deviation from a fixed point (either Z_0 or Z_s). Suppose the right eigenvectors are arranged in a matrix R and the corresponding left eigenvectors (normalized such that $L_i \cdot R_j = \delta_{ij}$) are arranged in a matrix L. Then the expansion

$$\Delta Z(t) = \sum_i c_i v_i \iff \Delta Z(t) = Rc$$

allows us to extract the coefficients by projecting onto the left eigenvectors:

$$c = L\Delta Z(t).$$

At the stable fixed point, for instance, we only want deviations along the unstable directions. This requirement translates into setting the coefficients corresponding to the stable modes to zero:

$$c_i = L_i \cdot [Z(t) - Z_0] = 0 \text{ for stable modes.}$$

Similarly, at the saddle point, to ensure the trajectory approaches along the stable manifold, the coefficients for the unstable directions must vanish:

$$d_j = L_j \cdot [Z(t) - Z_s] = 0 \text{ for unstable modes.}$$

By enforcing these constraints at t_i and t_f, the finite-time boundary value problem inherits the correct asymptotic behavior, ensuring that the switching trajectory departs and arrives in the proper subspaces.

3.3. Numerical Solution

The task of finding the switching paths is now cast as a boundary value problem (BVP). Our next task is apply a numerical method to find a solution. For this we will use SciPy's `solve_bvp` [1] implementation of a collocation method, meaning a numerical technique for solving differential equations by approximating the solution using a set of basis functions, such as cubic splines, and enforcing the differential equations are satisfied to a given tolerance at specific points called collocation points.

Convergence

When applying this solver to our problem there are two key considerations to check convergence of the solution:

1. **Finite Time Domain**: Since we cannot numerically integrate from $t \to -\infty$ to $t \to +\infty$, we must choose finite initial and final times t_i and t_f. These should be chosen such that the system is sufficiently close to the fixed points at the boundaries.
2. **Numerical Parameters**: The key parameter for solution quality is the error tolerance, which caps the allowed residuals at the collocation points [1]. Lower error tolerances can be reached by increasing the number of collocation points, at the cost of increased computational time and memory usage.

Whether or not they are we have converged to a desired solution can be judged by whether or not the action along the path has reached to a stable value. This can be checked by recalculating the paths and actions using different values of $t_f - t_i$, error tolerances and numbers of collocation points and checking if the resulting actions are consistent.

Initial Guess

The BVP solver requires an initial guess for the solution. This can be constructed by linear interpolation between the fixed points. This initial guess is most effective near the saddle-node bifurcations where the stable and saddle points are closest to each other. For more distant points with more complex paths, this linear guess may not converge to a solution, in which case we can reuse solutions from neighbouring points in the parameter space (numerical continuation).

3.4. Implementation Example

Here's an example of how to implement the switching paths calculation in Python using our codebase:

```
from pathlib import Path
from metastable.map.map import FixedPointMap, FixedPointType
from metastable.paths import (
    get_bistable_kappa_range,
    generate_sweep_index_pairs,
    map_switching_paths
)
from metastable.action.map import map_actions
from metastable.paths.boundary_conditions.boundary_conditions_lock import BoundaryLockParams

# Load the fixed point map
map_path = Path("fixed_points/examples/map-with-stability.npz")
fixed_point_map = FixedPointMap.load(map_path)

# Choose parameter values for the sweep
epsilon_idx = 380  # Fixed epsilon value
kappa_boundaries = get_bistable_kappa_range(fixed_point_map.bistable_region, epsilon_idx)

# Generate parameter sweeps
kappa_cuts = generate_sweep_index_pairs(
    kappa_boundaries,
    bright_sweep_fraction=0.4,  # Sweep 40% of the way from bright to saddle
    dim_sweep_fraction=0.95     # Sweep 95% of the way from dim to saddle
)

# Configure boundary conditions
bright_lock_params = BoundaryLockParams(
    stable_threshold=1e-2,      # Threshold for stable manifold
    stable_linear_coefficient=0.0,
    saddle_threshold=1e-2,      # Threshold for saddle manifold
    saddle_linear_coefficient=0.0
)

# Calculate switching paths
output_path = Path("sweep")
path_results_bright = map_switching_paths(
    fixed_point_map,
    kappa_cuts.bright_saddle,
    output_path,
    t_end=10.0,                # Integration time
    endpoint_type=FixedPointType.BRIGHT,
    lock_params=bright_lock_params,
    tol=1e-3,                  # Error tolerance
    max_nodes=1000000          # Maximum number of collocation points
)

# Calculate actions for the paths
fixed_point_map = FixedPointMap.load(output_path / "map.npz")
fixed_point_map_with_actions = map_actions(fixed_point_map)
```

```
fixed_point_map_with_actions.save(output_path / "map.npz")
```

Key aspects of the implementation:

1. **Parameter Selection**: We first choose a fixed value of ϵ and find the bistable range of κ values.
2. **Boundary Conditions**: The `BoundaryLockParams` class configures how we enforce the boundary conditions at the fixed points:
   - `stable_threshold`: Controls how close we need to be to the stable manifold
   - `saddle_threshold`: Controls how close we need to be to the saddle manifold
   - `stable_linear_coefficient` and `saddle_linear_coefficient`: Control the linear terms in the boundary conditions
3. **Path Calculation**: The `map_switching_paths` function:
   - Takes an initial guess (generated automatically)
   - Uses SciPy's `solve_bvp` internally
   - Handles the boundary value problem with our specified conditions
   - Returns the calculated paths
4. **Action Calculation**: After finding the paths, we calculate the corresponding actions using `map_actions`.

The code handles all the technical details of: - Setting up the boundary value problem - Managing the numerical integration - Handling the boundary conditions - Calculating the actions

This implementation allows us to systematically explore the switching paths across different regions of parameter space.

4. Results

We continue to examine the system studied in Stability Analysis. We begin by finding the switching paths and actions as a function of κ at fixed $\epsilon/\delta = 2.44$. The results are shown in the interactive visualization below, which consists of two panels:

1. **Upper Panel (Bifurcation Diagram)**: Shows the bifurcation structure in the $(\kappa/\delta, \epsilon/\delta)$ plane. The red and blue lines represent the unstable-bright and unstable-dim bifurcation boundaries respectively. The horizontal dashed black line indicates our chosen $\epsilon/\delta = 2.44$ cut, and the grey shaded region indicates the bistable region where switching paths exist.
2. **Lower Panel (Action Values)**: Displays the calculated actions for the switching paths:
   - Red line: Keldysh action $R_{b\to u}$ for the bright-to-unstable transition
   - Blue line: Keldysh action $R_{d\to u}$ for the dim-to-unstable transition

- Purple dash-dot line: Kramers (analytical) prediction for $R_{b\to u}$
- Green dash-dot line: Kramers (analytical) prediction for $R_{d\to u}$

The actions are scaled by λ and plotted with a negative sign to match conventional energy barrier representations. The close agreement between the numerically computed Keldysh actions and the analytical Kramers predictions validates our numerical approach.

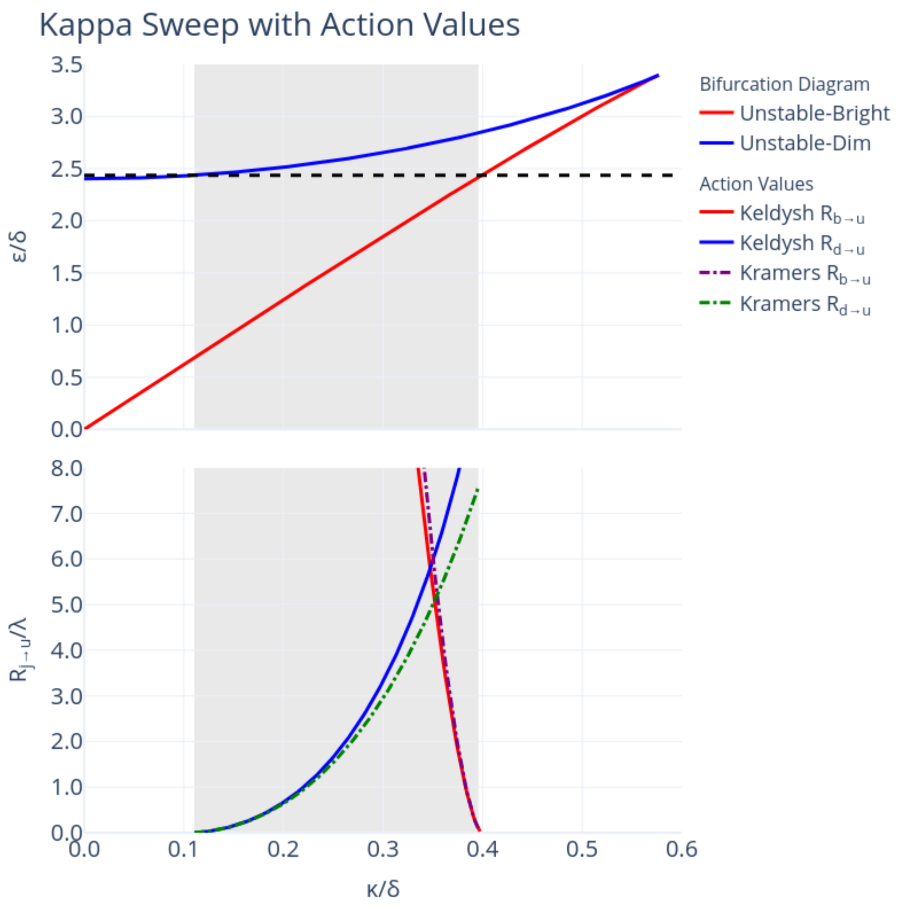

Figure 1: Kappa sweep with actions

The switching paths were computed using boundary conditions that ensure proper alignment with the stable and unstable manifolds at each fixed point, with thresholds set to 10^{-3} for both stable and saddle points. The numerical integration was performed over a finite time domain $[-5.0, 5.0]$, which proved sufficient for convergence of the action values.

Then as a function of ϵ at fixed $\kappa/\delta = 0.240$. The results are shown in the interactive visualization below, which consists of two panels:

1. **Upper Panel (Bifurcation Diagram)**: Shows the bifurcation structure in the $(\kappa/\delta, \epsilon/\delta)$ plane. The red and blue lines represent the unstable-bright and unstable-dim bifurcation boundaries respectively. The vertical dashed black line indicates our chosen $\kappa/\delta = 0.240$ cut, and the grey shaded region indicates the bistable region where switching paths exist.

2. **Lower Panel (Action Values)**: Displays the calculated actions for the switching paths:

    - Red line: Keldysh action $R_{b\to u}$ for the bright-to-unstable transition
    - Blue line: Keldysh action $R_{d\to u}$ for the dim-to-unstable transition
    - Purple dash-dot line: Kramers (analytical) prediction for $R_{b\to u}$
    - Green dash-dot line: Kramers (analytical) prediction for $R_{d\to u}$

The actions are scaled by λ and plotted with a negative sign to match conventional energy barrier representations. The close agreement between the numerically computed Keldysh actions and the analytical Kramers predictions validates our numerical approach.

The switching paths were computed using boundary conditions that ensure proper alignment with the stable and unstable manifolds at each fixed point, with thresholds set to 10^{-2} for both stable and saddle points. The numerical integration was performed over a finite time domain $[-5.5, 5.5]$, which proved sufficient for convergence of the action values close to the bifurcation points. However in the middle of the bistable regime convergence was not achieved.

References

[1] SciPy documentation, "scipy.integrate.solve_bvp", https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_bvp.html.

[2] J. Kierzenka, L. F. Shampine, "A BVP Solver Based on Residual Control and the Maltab PSE", ACM Trans. Math. Softw., Vol. 27, Number 3, pp. 299-316, 2001.

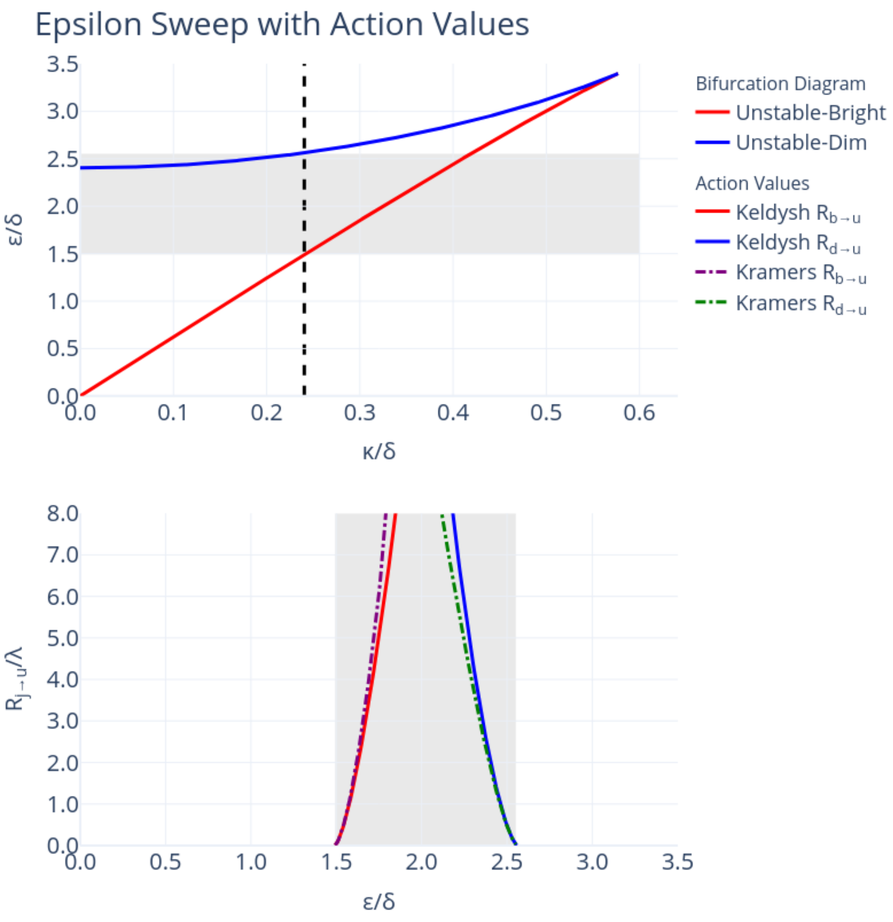

Figure 2: Epsilon sweep with actions