HiSD Package: solscape V-1.0

A Python package for constructing solution landscapes using High-index Saddle Dynamics (HiSD). This toolkit enables numerical computation of saddle points and their hierarchical organization in dynamical systems. It simplifies the process of saddle point searching, offers flexible parameter settings and many visualization tools.

Installation & Import

Step 1: Download the Code First, download the code from GitHub:

- **GitHub Repository**: https://github.com/HiSDpackage/solscape

Step 2: Add the `solscape-1.0` Directory to the System Path After downloading the code, you need to add the path of the `solscape-1.0` directory to the system path. This will allow you to access the package from anywhere on your system.

Use the following Python code to add the directory to the system path:

```
import sys
sys.path.append('/path/to/solscape-1.0')
```

Replace `'/path/to/solscape-1.0'` with the actual path where the `solscape-1.0` directory is located.

Step 3: Import the Main Class Once the path is set, you can import the main class `Landscape` from the `solscape` package as follows:

```
from solscape import Landscape
```

This will allow you to use the `Landscape` class and other functionalities provided by the package.

Configuration Parameters

The configuration parameters for the `solscape` package are divided into different categories, each focusing on a specific part of the algorithm. Below is an overview of the different categories and the associated parameters.

System Parameters

These parameters are related to the system setup and its general properties: `Dim`, `EnergyFunction`, `Grad`, `AutoDiff`, `NumericalGrad`, `DimerLength`, `SymmetryCheck` and `GradientSystem`.

`Dim` **(Optional)**
Description

System dimension specification.

Data Type
`int` (positive)

Behavior

- Default: Inferred from `initial_point`
- Manual override must match dimensions of:
  - `initial_point`
  - `InitialSearchDirection`

`EnergyFunction` **(Conditional Required) Description**
Specifies the energy function for gradient systems.

Data Type

- Python function: `Callable[[np.ndarray], float]`
  - **Input**: `np.ndarray` with shape `(d, 1)` (column vector)
- Symbolic expression: `str`
  - **Specification**: Use `x1, x2, ..., xd` as variables & Supports full `sympy` syntax

Example

```
EnergyFunction = '''
0.4*x1**2 - 0.2*x1*x2 + 0.25*x2**2
- 5*(atan(x1-5) + atan(x2-5))
'''
```

`Grad` **(Conditional Required)**
Description
Specifies the gradient function for gradient systems or vector field for non-gradient systems.

Requirements

- For gradient systems: ∇E
- For non-gradient systems: Direct specification of F in $\dot{x}$ = -F(x)

Data Types

- Python function: `Callable[[np.ndarray], np.ndarray]`
  - **Input/Output**: `np.ndarray` with shape `(d, 1)` (column vector)
- Symbolic expression list: `list[str]` (e.g., `["2*x1", "cos(x2)"]`)

Example

```
# Gradient system
Grad = ["2*x1", "2*x2"]
```

```
# Non-gradient system
Grad = ["x2", "-x1 - 0.1*x2"]
```

`AutoDiff` **(Optional)**

Description

Enables automatic differentiation of the energy function.

Data Type

`bool`

Options

- `True`: Requires `EnergyFunction` parameter
- `False`: Requires manual specification of `Grad` parameter

Behavior

- Defaults to `True` if `EnergyFunction` is provided
- Defaults to `False` if `Grad` is provided

`NumericalGrad` **(Optional)**

Description

Enables numerical gradient approximation.

Applicability

Only active when `Grad` is unspecified

Data Type

`bool`

Options

- `True`: Use finite difference approximation
- `False`: (Default) Use analytical gradient from `EnergyFunction`

`DimerLength` **(Optional)**

Description

Displacement length for numerical gradient approximations.

Data Type

`float` (positive)

Default

`1e-5`

`SymmetryCheck` **(Optional)**

Description

Verifies gradient system properties via Hessian symmetry.

Data Type
`bool`

Behavior

- `True`: (default) Auto-detect
- `False`: Bypass checks (improves performance)

`GradientSystem` **(Optional)**
Description
Explicit declaration of gradient system nature.

Usage
Override automatic detection when known a priori.

Data Type
`bool`

Behavior

- missing: Auto-detect

Hessian Parameters

These parameters are related to the Hessian matrix: `ExactHessian` and `HessianDimerLength`.

`ExactHessian` **(Optional)**
Description
Controls Hessian matrix computation method.

Data Type
`bool`

Options

- `True`: Analytical Hessian from symbolic gradient
- `False`: (Default) Dimer-based approximation

Requirement
Requires `Grad` as symbolic expressions

`HessianDimerLength` **(Optional)**
Description
Displacement length for numerical Hessian-Vector product approximations.

Data Type
`float` (positive)

Default

`1e-5`

Eigen Parameters

These parameters control how the eigen pairs (eigenvalues and eigenvectors) are computed: `EigenMethod`, `EigenMaxIter`, `EigenStepSize` and `PrecisionTol`.

`EigenMethod` (Optional)

Description

Eigensolver selection for stability analysis.

Data Type

`str`

Options

Method	System Type	Description
`lobpcg`	Gradient (Default)	Locally Optimal Block PCG
`euler`	Gradient & Non-gradient	Explicit Euler Discretization
`power`	Non-gradient (Default)	Power Method

`EigenMaxIter` (Optional)

Description

Maximum iterations for eigenpair computation.

Data Type

`int` (positive)

Default

`10`

`EigenStepSize` (Optional)

Description

Discretization step for Euler/power methods.

Data Type

`float` (positive)

Default

`1e-7`

`PrecisionTol` (Optional)

Description

Precision tolerance for eigenvalues. (We treat eigenvalues as 0 if its absolute value less than tolerance.)

Data Type
`float` (non-negative)

Default
`1e-5`

Acceleration Parameters

These parameters are related to improving the speed and efficiency of the algorithm: `BBStep`, `Acceleration`, `NesterovChoice`, `NesterovRestart` and `Momentum`.

`BBStep` **(Optional)**
Description
Enables Barzilai-Borwein adaptive step sizing.

Data Type
`bool`

Default
`False`

Trade-off

- May accelerate convergence
- Can cause instability in stiff systems

`Acceleration` **(Optional)**
Description
Convergence acceleration technique.

Data Type `str`

Options

- `none`: (Default) No acceleration
- `heavyball`: Momentum-based acceleration
- `nesterov`: Nesterov-accelerated dynamics

`NesterovChoice` **(Optional)**
Description
Specifies the acceleration parameter sequence for Nesterov's method.

Data Type
`int`

Options

- `1` : (default) $\gamma_n = \frac{n}{n+3}$.
- `2` : $\gamma_n = \frac{\theta_n - 1}{\theta_{n+1}}$, with $\theta_{n+1} = \frac{1+\sqrt{1+4\theta_n^2}}{2}$, $\theta_0 = 1$.

Default
`1`

`NesterovRestart` **(Optional)**
Description
Iteration interval for Nesterov momentum reset.

Data Type
`int` (positive) | `None`

Behavior

- `None`: Disables momentum restart
- Integer `n`: Resets momentum every `n` iterations

Default
`None`

`Momentum` **(Optional)**
Description
Momentum coefficient for heavy ball acceleration.

Data Type
`float` (non-negative)

Default
`0.0`

Constraints

- `0.0 <= Momentum < 1.0`
- `0.0`: Equivalent to no acceleration

Solver Parameters

These parameters are related to the solver process and control the behavior of the HiSD process: `InitialPoint`, `Tolerance`, `SearchArea`, `TimeStep`, `MaxIter`, `SaveTrajectory`, `Verbose` and `ReportInterval`.

`InitialPoint` **(Required) Description**
The starting coordinates for saddle point search.

Data Types
`list` | `numpy.ndarray` (1D array)

Example
`initial_point = [0.5, -1.2]`

`Tolerance` **(Optional)**

Description

Convergence threshold for saddle point iterations.

Data Type

`float` (positive)

Default

`1e-6`

Stopping Criterion

$Vert$ gradient vector $Vert_2$ < Tolerance

`SearchArea` **(Optional)**

Description

Maximum exploration radius from initial point.

Data Type

`float` (positive)

Default

`1e3`

Effect

Terminates search if $||$ x - x_0 $||_2$ > SearchArea

`TimeStep` **(Optional)**

Description

Temporal discretization interval for dynamics.

Data Type

`float` (positive)

Default

`1e-4`

`MaxIter` **(Optional)**

Description

Maximum number of HiSD iterations permitted.

Data Type

`int` (positive)

Default

`1000`

`SaveTrajectory` **(Optional)**

Description

Records full optimization path during computation.

Data Type

`bool`

Default

`True`

`Verbose` **(Optional)**

Description

Controls real-time progress reporting.

Output

Prints iteration count and gradient norm.

Data Type

`bool`

Default

`False`

`ReportInterval` **(Optional)**

Description

Iteration frequency for console output when `Verbose=True`.

Data Type

`int` (positive)

Default

`100`

Landscape Parameters

These parameters are related to constructing and navigating the solution landscape: `MaxIndex`, `MaxIndexGap`, `SameJudgement`, `InitialEigenVectors`, `PerturbationMethod`, `PerturbationRadius`, `PerturbationNumber` and `EigenCombination`.

`MaxIndex` **(Optional) Description**

Maximum saddle index (k) to compute.

- Index 0: Uses standard SD (Steepest Descent) method
- Index >=1: Uses HiSD (High-index Saddle Dynamics) method

Data Type

`int` (non-negative)

Default

`6`

Constraints

`0 <= max_index <= Dim`

`MaxIndexGap` **(Optional) Description**

Maximum allowed index difference between parent and child saddle points during hierarchical search.

Data Type

`int` (positive)

Default

`1`

Example

If `MaxIndexGap = 2`:

- Parent (index 4) --> Children (indices 2, 3)
- Parent (index 3) --> Children (indices 1, 2)

`SameJudgement` **(Optional)**

Description

Saddle point equivalence criterion.

Data Types

- `float` (positive): Threshold for 2-norm distance (default: 1e-3)
- `Callable[[np.ndarray, np.ndarray], bool]`: Custom comparison function
  - **Input**: `np.ndarray` with shape `(d, 1)` (column vector)

Example

```
# Custom similarity check
def custom_judge(a, b):
    return np.linalg.norm(a - b) < 0.5 and abs(a[0] - b[0]) < 0.1
```

`InitialEigenVectors` **(Optional)**

Description

Initial guess for Hessian eigenvectors.

Data Types

- `None`: (Default) Auto-initialize with k smallest eigenvectors
- Manual input: `np.ndarray` of shape (d, k)
  where d is the dimension and k equals MaxIndex.

Requirement

Column vectors must be orthonormal

`PerturbationMethod` **(Optional)**

Description

Statistical distribution for saddle point exploration.

Data Type

`str`

Options

Value	Description
`uniform` (Default)	Uniform sampling
`gaussian`	Normally distributed displacements

`PerturbationRadius` **(Optional)**

Description

Displacement magnitude for saddle perturbations.

Data Type

`float` (positive)

Default

`1e-4`

`PerturbationNumber` **(Optional)**

Description

Number of directional probes per saddle point.

Note

Actual probes = `2 * PerturbationNumber` (bidirectional)

Data Type

`int` (positive)

Default

`2`

`EigenCombination` **(Optional)**

Description

Strategy for eigenvector utilization in perturbations.

Data Type

`str`

Options

Value	Computational Cost	Completeness
`all` (Default)	High	Exhaustive

Value	Computational Cost	Completeness
`min`	Low	Partial

Execution Interface

`.Run()`

Description
Initiates the solution landscape computation process.

Parameters
None

Usage

```
landscape = Landscape(
    MaxIndex=3,
    AutoDiff=True,
    EnergyFunction="1*(x1**2-1)**2+2*(x2**2-1)**2+3*(x3**2-1)**2",
    InitialPoint=np.array([0, 0, 0])
)
landscape.Run()  # Starts computation
```

Output
Access search trajectory by `instance.DetailRecord`. Each row of it is data for one successful search. sequentially contains ID of end saddle point, ID of start saddle point, ndarray of each step position and ndarray of each step time position.

Access saddle point data by `instance.SaddleList`. Each row of it is data for one saddle point. sequentially contains ID of the saddle point, position of the saddle point, Morse Index of the saddle point, the eigenvectors of negative eigenvalues and set of father saddles.

See `Output Objects` for more details.

`.RestartFromPoint(RestartPoint, MaxIndex)`

Description
Restarts computation from specified coordinates.

Parameters

Name	Type	Constraints	Description
`RestartPoint`	`list` or `numpy.ndarray` (1D)	Must match system dimension	Initial position vector

Name	Type	Constraints	Description
`MaxIndex`	`int`	`0 <= max_index <= dim`	Maximum saddle index to compute

Usage

```
landscape.RestartFromPoint(
    RestartPoint=np.array([[0.2], [-0.8]]),
    MaxIndex=2
)
```

`.RestartFromSaddle(BeginID, Perturbation, MaxIndex)`

Description

Restarts computation from existing saddle point.

Parameters

Name	Type	Constraints	Description
`BeginID`	`int`	`0 <= begin_id < len(SaddleList)`	Valid saddle point ID
`Perturbation`	`numpy.ndarray` (1D)	Must match system dimension	Initial perturbation vector
`MaxIndex`	`int`	`0 <= max_index <= dim`	Maximum saddle index to compute

Usage

```
landscape.RestartFromSaddle(
    BeginID=3,
    Perturbation=1e-3 * np.array([[-1], [0.5]]),
    MaxIndex=2
)
```

`.DrawTrajectory(**kwargs)`

Description

Visualizes search trajectories and energy landscapes for systems. Supports contour overlays, style customization, and high-D projections.

Parameters

Name	Type	Constraints	Description & Default Value
`DetailedTraj`	`bool`	Requires saved trajectory data	Show full iteration path (default: `False`)
`Contour`	`bool`	2D systems only	Enable contour lines (default: `True`)
`Contourf`	`bool`	`Contour=True`	Enable color-filled contours (default: `True`)
`ContourGridNum`	`int`	`>= 1`	Main grid divisions per axis (default: `50`)
`ContourGridOut`	`int`	`>= 0`	Extended grid divisions per axis (default: `10`)
`Title`	`str`		Figure title text (defalut: `"The Search Trajectory"`)
`TrajectorySet`	`dict`		Set the trajectory style (default: `{"linewidth": 0.4, "linestyle": "-", "color": "blue", "label": "Search Trajectory"}`)
`SaddlePointSet`	`list[dict]`		Set the saddle point style (default: `[{"marker": "o", "color": colors[20 * i + 20], "label": f"Index {i} Saddle Point"} for i in range(instance.MaxIndex + 1)]`)
`GridSet`	`dict`		Set the grid style (default: `{"visible": True, "linestyle": "--", "linewidth": 0.1, "color": "gray"}`)
`1DSamples`	`int`	`>= 1` (1D only)	Function sampling density (default: `1e3`)
`1DSamplesOut`	`int`	`>= 0` (1D only)	Extended sampling range (default: `1e2`)
`1DFunctionDraw`	`dict`	1D only	Set the 1D function style (default: `{"linewidth": 2, "linestyle": "-", "color": "red", "label": "Function Curve"}`)
`WhetherSave`	`bool`		Save to file (default: `False`)
`SaveFigurePath`	`str`	File extension determines format	Output path (default: `"Landscape_figure.png"`)

Name	Type	Constraints	Description & Default Value
`Projection`	`None` or `callable`	Required for dim > 2	Projection function (Signature: `((n*dim) array) --> ((n*2) array)`)

Usage

```
# 2D system without detailed trajectory
landscape.DrawTrajectory(
    Contour=True,
    Contourf=True,
    WhetherSave=True,
    ContourGridNum=100,
    ContourGridOut=25,
    SaveFigurePath="landscape.png"
)

# high-D system with detailed trajectory
import numpy as np
def proj_func(input):
    output = np.hstack((1.0 * input[:, [0]]+ 1.5 * input[:, [1]], 1.0 * input[:,
[0]]+ 2.5 * input[:, [2]]))
    return output

landscape.DrawTrajectory(
    DetailedTraj=True,
    ContourGridNum=100,
    ContourGridOut=25,
    Projection=proj_func
)
```

Visualization Rules
Dimensionality Handling:

- 1D: X-axis = iteration count, Y-axis = function value
- 2D: Natural coordinates
- =3D: Requires `Projection` to 2D plane

Data Requirements

- Detailed trajectories require `SaveTrajectory=True` in `.Run()` call

Output

- Matplotlib figure (interactive display)
- Image file when `WhetherSave=True` (PNG/PDF/SVG based on path)

`.DrawConnection(**kwargs)`

Description

Visualizes connectivity between saddle points in the solution landscape. Displays markers for saddle nodes and search paths connecting them. Supports custom styling of points and connection lines.

Parameters

Name	Type	Default Value	Description
`Title`	`str`	`"The Connection of Saddle Points"`	Figure title text
`SaddlePointSet`	`list[dict]`	`[{"node_shape": "o", "node_color": colors[i], "label": f"Index {i} Saddle Point"}for i in range(instance.MaxIndex + 1)]`	Set the saddle point style
`TrajectorySet`	`dict`	`{"width": 0.4, "style": "solid", "edge_color": "blue", "label": "Search Trajectory"}`	Set the connection path style
`WhetherSave`	`bool`	`False`	Save to file
`SaveFigurePath`	`str`	`"Landscape_Connection_figure.png"`	Output path (file extension determines format)

Usage

```
# Draw solution landscape
landscape.DrawConnection(
    Title="Solution Landscape",
    WhetherSave=True,
    SaveFigurePath="landscape.png"
)
```

`.Save(filepath, fileformat)`

Description

Persists calculation results to persistent storage with specified serialization format.

Parameters

Name	Type	Constraints	Description
`filepath`	`str`	Valid filesystem path	Target path without extension

Name	Type	Constraints	Description
`fileformat`	`str`	`"json" | "pickle" | "mat"` (default: `"json"`)	Serialization format

Usage

```
landscape.Save(filepath="/output/results", fileformat="pickle")
```

File Output
Generates:

- `filepath.json` (JSON text format)
- `filepath.pickle` (Python binary serialization)
- `filepath.mat` (MATLAB-compatible binary)

Contains dictionary with aligned indices for:

1. `SaddleID`: Unique identification numbers
2. `Position`: Spatial coordinates (N-dimensional array)
3. `MorseIndex`: Number of negative eigenvalues
4. `FatherSet`: Ancestry relationships (list-formatted references)

Index Consistency
All data structures maintain identical ordering:
`SaddleID[n]` - `Position[n]` - `MorseIndex[n]` - `FatherSet[n]`

Output Objects

`.SaddleList`

Description
Collection of discovered saddle points with topological metadata.

Type
`list[list]`

Node Structure

```
[
  [
    ID: int,               # Index 0 - Unique identifier
    Position: np.ndarray,  # Index 1 - Coordinate vector (d-dim)
    Morse Index: int,      # Index 2 - Number of negative eigenvalues
    Unstable Directions: np.ndarray,  # Index 3 - Eigenvectors (d x k)
    Parent IDs: list[int]   # Index 4 - Ancestor saddle IDs (where -1 indicates
initial point)
  ]
```

```
  ... # Other saddle points
]
```

Access

```
# Get all index-1 saddles
index1_saddles = [node for node in landscape.SaddleList if node[2] == 1]

# Get position of saddle ID 5
saddle5_pos = landscape.SaddleList[5][1]
```

.DetailRecord

Description
Complete search trajectory data.

Type
`list[list]`

Record Structure

```
[
  [
    End ID: int,            # Index 0 - Terminating saddle ID
    Start ID: int,          # Index 1 - Originating saddle ID
    Path Positions: np.ndarray,  # Index 2 - Trajectory points (N x d)
    Path Times: np.ndarray       # Index 3 - Temporal coordinates (N x 1)
  ]
  ... # Other search trajectory
]
```

Access

```
# Plot first search path
first_search = landscape.DetailRecord[0]
plt.plot(first_search[2][:,0], first_search[2][:,1])
```

Package Requirements

Overview

This component requires the following Python packages to be installed. Verify installations before usage.

Built-in Packages

The following standard library modules are required (no installation needed):

```
import copy
import sys
import warnings
import inspect
import json
import itertools
import math
import pickle
```

Third-party Dependencies

Required external packages with version specifications:

Package	Required Version	Installation Command
NumPy	2.2.3	`pip install numpy==2.2.3`
SciPy	1.15.2	`pip install scipy==1.15.2`
SymPy	1.13.3	`pip install sympy==1.13.3`
Matplotlib	3.10.0	`pip install matplotlib==3.10.0`
NetworkX	3.4.2	`pip install networkx==3.4.2`

Full Environment Setup

```
pip install "numpy>=2.2.3" \
    "scipy>=1.15.2" \
    "sympy>=1.13.3" \
    "matplotlib>=3.10.0" \
    "networkx>=3.4.2"
```

Version Verification

Check installed versions using:

```
import numpy, scipy, sympy, matplotlib, networkx

print(f"NumPy: {numpy.__version__}")       # Should show 2.2.3
print(f"SciPy: {scipy.__version__}")       # Should show 1.15.2
print(f"SymPy: {sympy.__version__}")       # Should show 1.13.3
print(f"Matplotlib: {matplotlib.__version__}")  # Should show 3.10.0
print(f"NetworkX: {networkx.__version__}") # Should show 3.4.2
```