Leniax

Morgan Giraud

Oct 06, 2022

QUICKSTART

Leniax is a Lenia simulation library powered by JAX. It can efficiently simulate and render Lenia worlds and can also be used to search for creatures, initial conditions, kernels and growth functions. It is also fully differentiable!

For a quick introduction and short example snippets, see our README.

CHAPTER

ONE

OVERVIEW

1.1 Background: JAX

JAX is NumPy + autodiff + GPU/TPU

It allows for fast scientific computing and machine learning with the normal NumPy API (+ additional APIs for special accelerator ops when needed).

1.2 Leniax

Leniax is a high-performance CA simulator library supporting variations like: - Lenia - Multi-neighbourhood CA - Neural CA - Hopefully even more variations in the future

Leniax comes with everything you need to simulate, evolve and differentiate Cellular Automatata. It includes:

- **Evolution API** (`leniax.qd`): You can thousands of simulations in parallel and compute statistics to apply heuristics.
- **Differentiability**: Thanks to JAX, all the core components are differentiable making it easy to compute the gradients of any part of your CA.
- **Educational examples** See our examples.

1.3 CPU/GPU/TPU support

All of our examples can run on CPU, GPU or TPU.

Following is an example of TPU and GPU scripts to look for interesting initialization conditions:

- Initialization search - GPU
- Initialization search - TPU

CHAPTER

TWO

INSTALLATION

2.1 From source

To install a version from source, clone the repo

```
git clone https://github.com/morgangiraud/leniax
cd leniax
```

Install Leniax library with conda (make sure you have it before typing the following command): `make install`

Then activate the environment: `conda activate leniax`

Finally, install the lib itself: `pip install .`

2.2 Verification

You can make sure that everything is working fine by running the following command: `make ci`

CHAPTER

THREE

THE LENIAX PHILOSOPHY

(Taken from the very good philosophy of Flax, in no particular order)

- Library code should be easy to read and understand.
- Prefer duplicating code over a bad abstraction.
- Generally, prefer duplicating code over adding options to functions.
- Unit test-driven design: If it's hard to test your code, consider changing the design.
- People start projects by copying an existing implementation – make base implementations excellent.
- If we expose an abstraction to our developers, we own the mental overhead.
- Developer-facing functional programming abstractions confuse some users, expose them where the benefit is high.
- "Read the manual" is not an appropriate response to developer confusion. The framework should guide developers towards good solutions, e.g. through assertions and error messages.
- An unhelpful error message is a bug.
- "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." -Brian Kernighan

CHAPTER

FOUR

HOW TO CONTRIBUTE

Everyone can contribute to Leniax, and we value everyone's contributions.

We also appreciate if you spread the word, for instance by starring our Github repo, or referencing Leniax in blog posts of projects that used it.

CHAPTER

FIVE

LENIAX.CORE PACKAGE

Leniax core simulation functions

leniax.core.**update**(*rng_key*, *state*, *K*, *gf_params*, *kernels_weight_per_channel*, *dt*, *get_potential_fn*, *get_field_fn*, *get_state_fn*)

Update the cells state

Jitted function with static argnums. Use functools.partial to set the different function. Avoid changing non-static argument shape for performance.

Parameters

- **rng_key** (*PRNGKeyArray*) – JAX PRNG key.
- **state** (*Array*) – cells state `[N, nb_channels, world_dims...]`
- **K** (*Array*) – Kernel `[K_o=nb_channels * max_k_per_channel, K_i=1, kernel_dims...]`
- **gf_params** (*Array*) – Growth function parmaeters `[nb_kernels, params_shape...]`
- **kernels_weight_per_channel** (*Array*) – Kernels weight used in the averaginf function `[nb_channels, nb_kernels]`
- **dt** (*Array*) – Update rate `[N]`
- **get_potential_fn** (*Callable*) – **(jit static arg)** Function used to compute the potential
- **get_field_fn** (*Callable*) – **(jit static arg)** Function used to compute the field
- **get_state_fn** (*Callable[[PRNGKeyArray, Array, Array, Array], Array]*) – **(jit static arg)** Function used to compute the new cell state

Returns

A tuple of arrays representing a jax PRNG key and the updated state, the used potential and used field.

Return type

Tuple[*Array*, *Array*, *Array*]

leniax.core.**get_potential_fft**(*state*, *K*, *tc_indices=None*, *channel_first=True*)

Compute the potential using FFT

The first dimension of cells and K is the vmap dimension

Parameters

- **state** (*Array*) – cells state `[N, nb_channels, world_dims...]`
- **K** (*Array*) – Kernels `[1, nb_channels, max_k_per_channel, world_dims...]`

- **tc_indices** (*Optional[Tuple]*) – Optional 1-dim array channels indices to keep
- **channel_first** (*bool*) – see https://jax.readthedocs.io/en/latest/_autosummary/jax.lax.conv_general_dilated.html

Returns
An array containing the potential

Return type
Array

leniax.core.**get_potential**(*state*, *K*, *padding*, *tc_indices=None*, *channel_first=True*)

Compute the potential using lax.conv_general_dilated

The first dimension of cells and K is the vmap dimension

Parameters

- **state** (*Array*) – cells state `[N, nb_channels, world_dims...]`
- **K** (*Array*) – Kernels `[K_o=nb_channels * max_k_per_channel, K_i=1, kernel_dims...]`
- **padding** (*Tuple*) – array with padding informations, `[nb_world_dims, 2]`
- **tc_indices** (*Optional[Tuple]*) – Optional 1-dim array channels indices to keep
- **channel_first** (*bool*) – see https://jax.readthedocs.io/en/latest/_autosummary/jax.lax.conv_general_dilated.html

Returns
An array containing the potential

Return type
Array

leniax.core.**get_field**(*potential*, *gf_params*, *kernels_weight_per_channel*, *growth_fn_t*, *weighted_fn*)

Compute the field

Jitted function with static argnums. Use functools.partial to set the different function. Avoid changing non-static argument shape for performance.

Parameters

- **potential** (*Array*) – `[N, nb_kernels, world_dims...]`
- **gf_params** (*Array*) – `[nb_kernels, nb_gf_params]`
- **kernels_weight_per_channel** (*Array*) – Kernels weight used in the averaginf function `[nb_channels, nb_kernels]`
- **growth_fn_t** (*Tuple[Callable, ...]*) – **(jit static arg)** Tuple of growth functions. `length: nb_kernels`
- **weighted_fn** (*Callable*) – **(jit static arg)** Function used to merge fields linked to the same channel

Returns
An array containing the field

Return type
Array

`leniax.core.`**`weighted_sum`**(*fields*, *weights*)

Compute the weighted sum of sub fields

Parameters

- **`fields`** (*Array*) – Raw sub fields `[N, nb_kernels, world_dims...]`
- **`weights`** (*Array*) – Weights used to compute the sum `[nb_channels, nb_kernels]` 0. values are used to indicate that a given channels does not receive inputs from this kernel

Returns

The unnormalized field

Return type

Array

`leniax.core.`**`weighted_mean`**(*fields*, *weights*)

Compute the weighted mean of sub fields

Parameters

- **`fields`** (*Array*) – Raw sub fields `[N, nb_kernels, world_dims...]`
- **`weights`** (*Array*) – Weights used to compute the sum `[nb_channels, nb_kernels]` 0. values are used to indicate that a given channels does not receive inputs from this kernel

Returns

The normalized field

Return type

Array

CHAPTER

SIX

LENIAX.RUNNER PACKAGE

`leniax.runner.run`(*rng_key*, *cells*, *K*, *gf_params*, *kernels_weight_per_channel*, *T*, *max_run_iter*, *R*, *update_fn*, *compute_stats_fn*, *stat_trunc=False*)

Simulate a single configuration

It uses a python `for` loop under the hood.

Parameters

- **rng_key** (*PRNGKeyArray*) – JAX PRNG key.
- **cells** (*Array*) – Initial cells state `[N_init=1, nb_channels, world_dims...]`
- **K** (*Array*) – Stacked Kernels (shape depends on convolution implementation)
- **gf_params** (*Array*) – Growth function parameters `[nb_kernels, params_shape...]`
- **kernels_weight_per_channel** (*Array*) – Kernels weight used in the average function `[nb_channels, nb_kernels]`
- **dt** – Update rate `[1]`
- **max_run_iter** (*int*) – Maximum number of simulation iterations
- **R** (*float*) – Main kernel Resolution
- **update_fn** (*Callable*) – Function used to compute the new cell state
- **compute_stats_fn** (*Callable*) – Function used to compute the statistics
- **stat_trunc** (*bool*) – Set to `True` to truncate run based on its statistics
- **T** (*Array*) –

Returns

A 5-tuple of arrays representing a jax PRNG key, the updated cells state, the used potential and used field and statistics

Return type

Tuple[*Array*, *Array*, *Array*, *Dict*[str, *Array*]]

`leniax.runner.run_scan`(*rng_key*, *cells0*, *K*, *gf_params*, *kernels_weight_per_channel*, *T*, *max_run_iter*, *R*, *update_fn*, *compute_stats_fn*)

Simulate a single configuration

This function is jitted, it uses jax.lax.scan function under the hood. It can be used to simulate a single configuration with multiple initialization.

Parameters

- **rng_key** (*PRNGKeyArray*) – JAX PRNG key.

- **cells0** (*Array*) – Initial cells state [N_init, nb_channels, world_dims...]
- **K** (*Array*) – Stacked Kernels [kernel_shape...]
- **gf_params** (*Array*) – Growth function parameters [nb_kernels, params_shape...]
- **kernels_weight_per_channel** (*Array*) – Kernels weight used in the average function [nb_channels, nb_kernels]
- **dt** – Update rate [1]
- **max_run_iter** (*int*) – Maximum number of simulation iterations
- **R** (*float*) – Main kernel Resolution
- **update_fn** (*Callable*) – Function used to compute the new cell state
- **compute_stats_fn** (*Callable*) – Function used to compute the statistics
- **T** (*Array*) –

Returns
A 4-tuple of arrays representing the updated cells state, the used potential and used field and simulations statistics

Return type
Tuple[*Array*, *Array*, *Array*, *Dict*[str, *Array*]]

leniax.runner.**run_scan_mem_optimized**(*rng_key*, *cells0*, *K*, *gf_params*, *kernels_weight_per_channel*, *T*, *max_run_iter*, *R*, *update_fn*, *compute_stats_fn*)

Vectorized version of run_scan_mem_optimized. Takes similar arguments as run_scan_mem_optimized but with additional array axes over which run_scan_mem_optimized is mapped.

Original documentation:

Simulate multiple configurations

This function is jitted, it uses jax.lax.scan function under the hood. It can be used to simulate multiple configurations with multiple initialization.

Args:
rng_key: JAX PRNG key. cells0: Initial cells state [N_sols, N_init, nb_channels, world_dims...] K: Stacked Kernels [N_sols, kernel_shape...] gf_params: Growth function parameters [N_sols, nb_kernels, params_shape...] kernels_weight_per_channel: Kernels weight used in the average function [N_sols, nb_channels, nb_kernels] T: Update rate [N_sols] max_run_iter: Maximum number of simulation iterations R: Main kernel Resolution update_fn: Function used to compute the new cell state compute_stats_fn: Function used to compute the statistics

Returns:
A 3-tuple representing a jax PRNG key, the simulations statistics and final cells states

Parameters

- **rng_key** (*PRNGKeyArray*) –
- **cells0** (*Array*) –
- **K** (*Array*) –
- **gf_params** (*Array*) –
- **kernels_weight_per_channel** (*Array*) –
- **T** (*Array*) –

- **max_run_iter** (*int*) –
- **R** (*float*) –
- **update_fn** (*Callable*) –
- **compute_stats_fn** (*Callable*) –

Return type
Tuple[*Dict*[str, *Array*], *Array*]

leniax.runner.**run_scan_mem_optimized_pmap**(*rng_key*, *cells0*, *K*, *gf_params*, *kernels_weight_per_channel*, *T*, *max_run_iter*, *R*, *update_fn*, *compute_stats_fn*)

Vectorized version of run_scan_mem_optimized_pmap. Takes similar arguments as run_scan_mem_optimized_pmap but with additional array axes over which run_scan_mem_optimized_pmap is mapped.

Original documentation:

Simulate multiple configurations on multiple devices

This function is jitted, it uses jax.lax.scan function under the hood. It can be used to simulate multiple configurations with multiple initialization on multiple devices.

Args:
rng_key: JAX PRNG key. cells0: Initial cells state `[N_device, N_sols, N_init, nb_channels, world_dims...]` K: Stacked Kernels `[N_device, N_sols, kernel_shape...]` gf_params: Growth function parameters `[N_device, N_sols, nb_kernels, params_shape...]` kernels_weight_per_channel: Kernels weight used in the average function `[N_device, N_sols, nb_channels, nb_kernels]` T: Update rate `[N_device, N_sols]` max_run_iter: Maximum number of simulation iterations R: Main kernel Resolution update_fn: Function used to compute the new cell state compute_stats_fn: Function used to compute the statistics

Returns:
A 3-tuple representing a jax PRNG key, the simulations statistics and final cells states

Parameters

- **rng_key** (*PRNGKeyArray*) –
- **cells0** (*Array*) –
- **K** (*Array*) –
- **gf_params** (*Array*) –
- **kernels_weight_per_channel** (*Array*) –
- **T** (*Array*) –
- **max_run_iter** (*int*) –
- **R** (*float*) –
- **update_fn** (*Callable*) –
- **compute_stats_fn** (*Callable*) –

Return type
Tuple[*Dict*[str, *Array*], *Array*]

CHAPTER

SEVEN

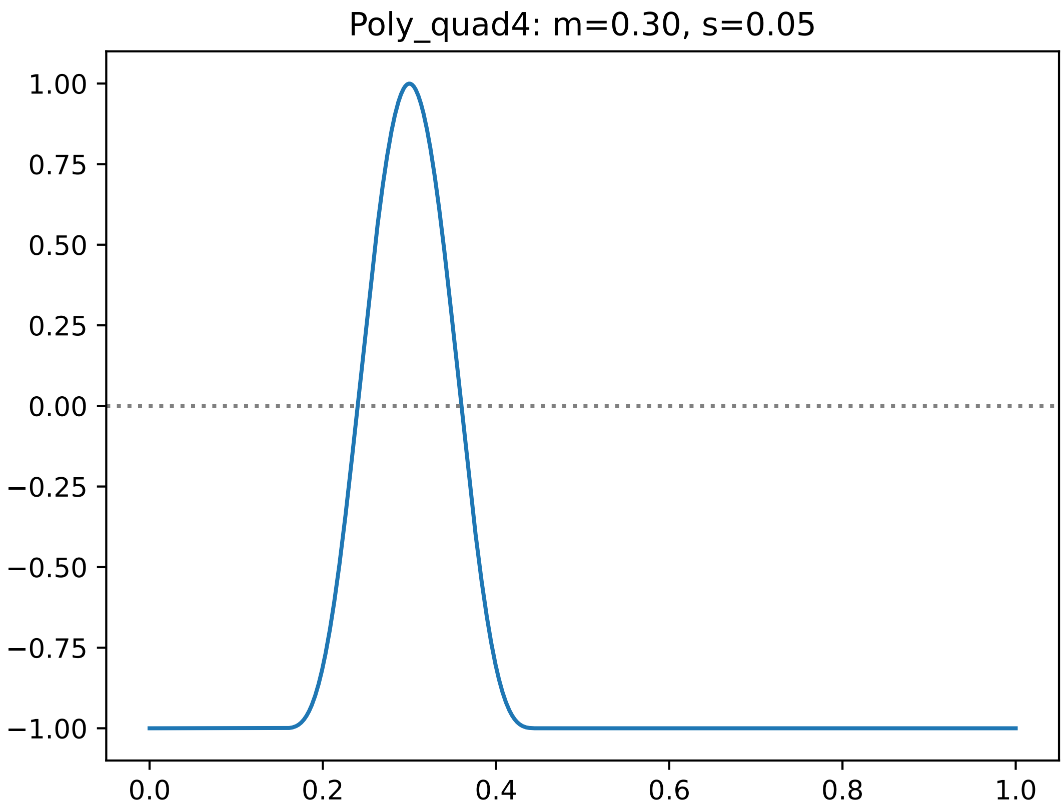
LENIAX.GROWTH_FUNCTIONS PACKAGE

Leniax growth functions

`leniax.growth_functions.poly_quad4(params, X)`

Quadratic polynomial growth function

$$y = 2 * \max\left[1 - \left(\frac{X - m}{3s}\right)^2, 0\right]^4 - 1$$



Parameters

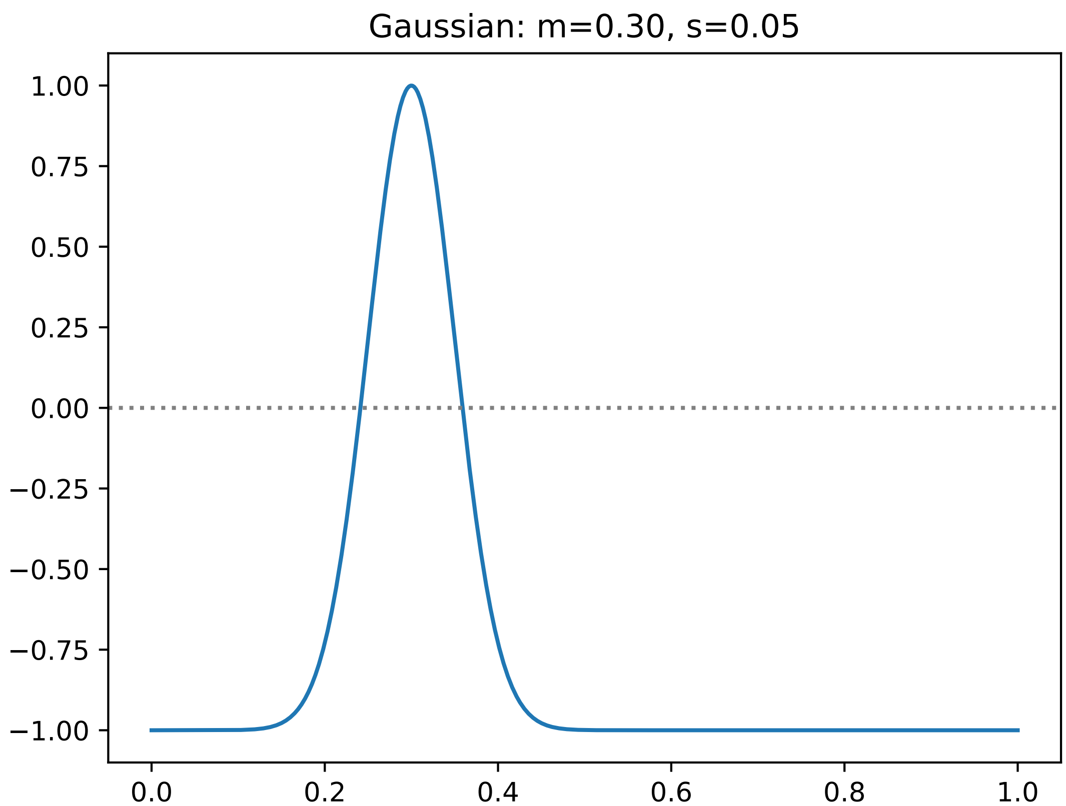
- **params** (*Array*) – parameters of the growth function
- **X** (*Array*) – potential

Returns
A field

`leniax.growth_functions.`**`gaussian`**`(`*params*, *X*`)`

Gaussian growth function

$$y = 2 * e^{-\frac{1}{2}\left(\frac{X-m}{s}\right)^2} - 1$$



Parameters

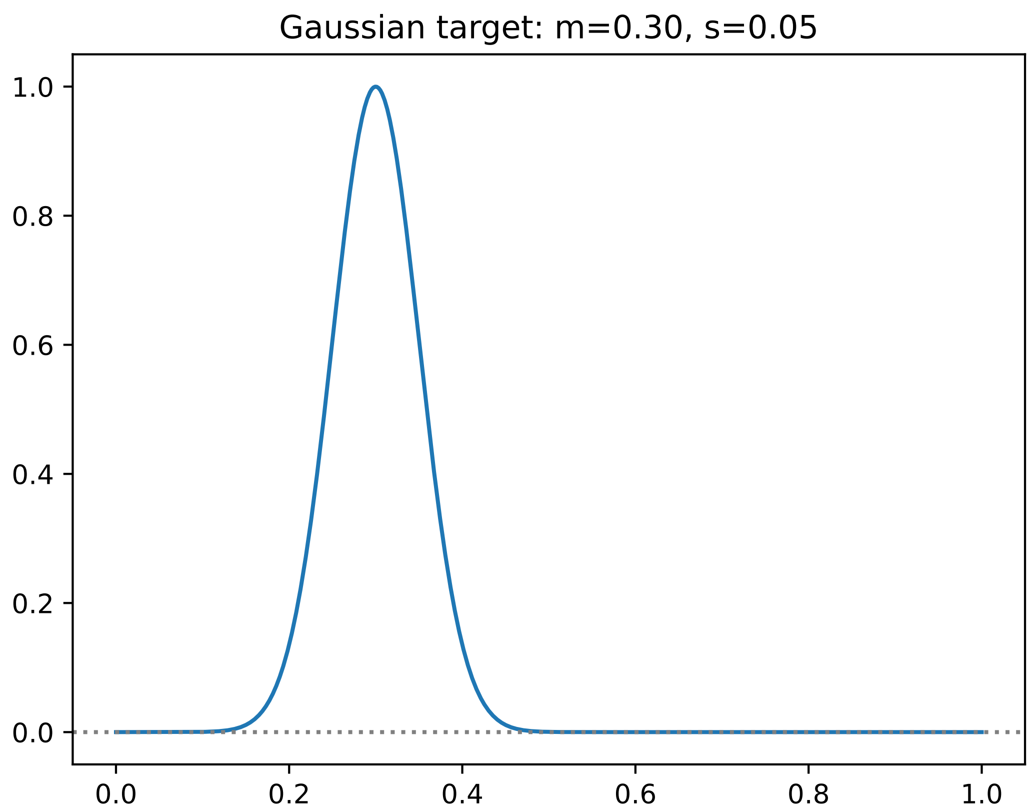
- **params** (*Array*) – parameters of the growth function
- **X** (*Array*) – potential

Returns
A field

`leniax.growth_functions.`**`gaussian_target`**`(`*params*, *X*`)`

Gaussian growth function for aymptotic Lenia

$$y = e^{-\frac{1}{2}\left(\frac{X-m}{s}\right)^2}$$


Gaussian target: m=0.30, s=0.05
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Parameters

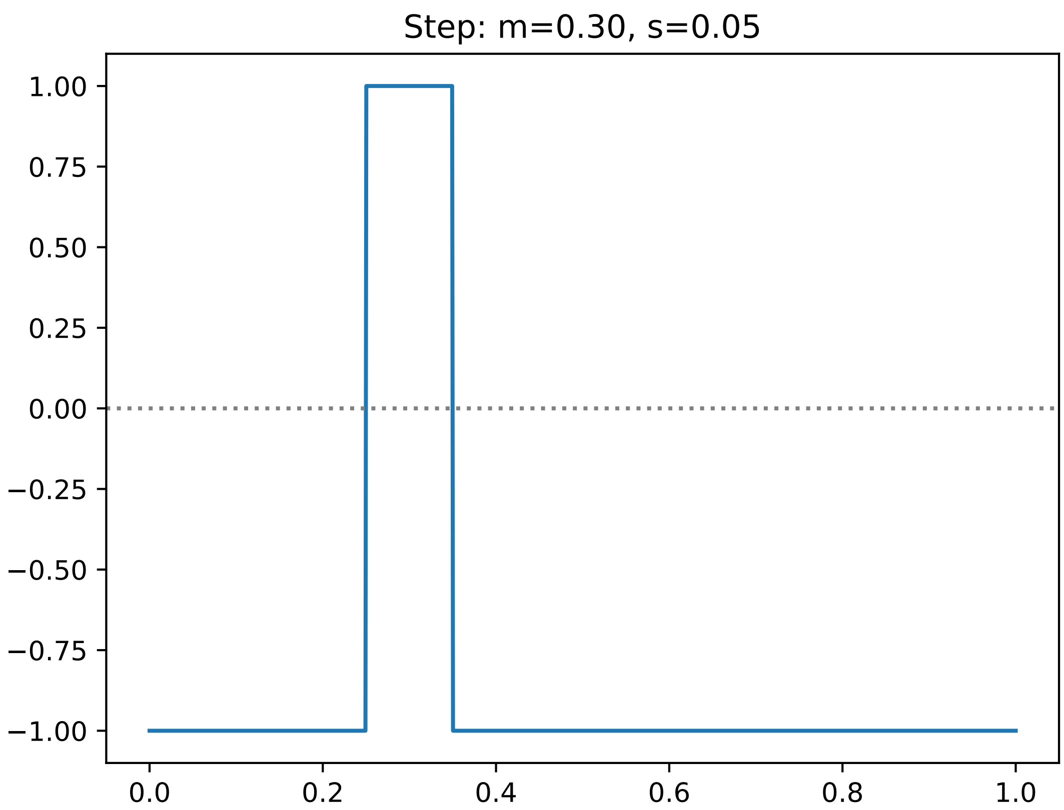
- **params** (*Array*) – parameters of the growth function
- **X** (*Array*) – potential

Returns
A field

`leniax.growth_functions.step`(*params*, *X*)

Step growth function

$$y = \begin{cases} 1, & \text{if } |X - m| \leq s \\ -1, & \text{otherwise} \end{cases}$$



Parameters

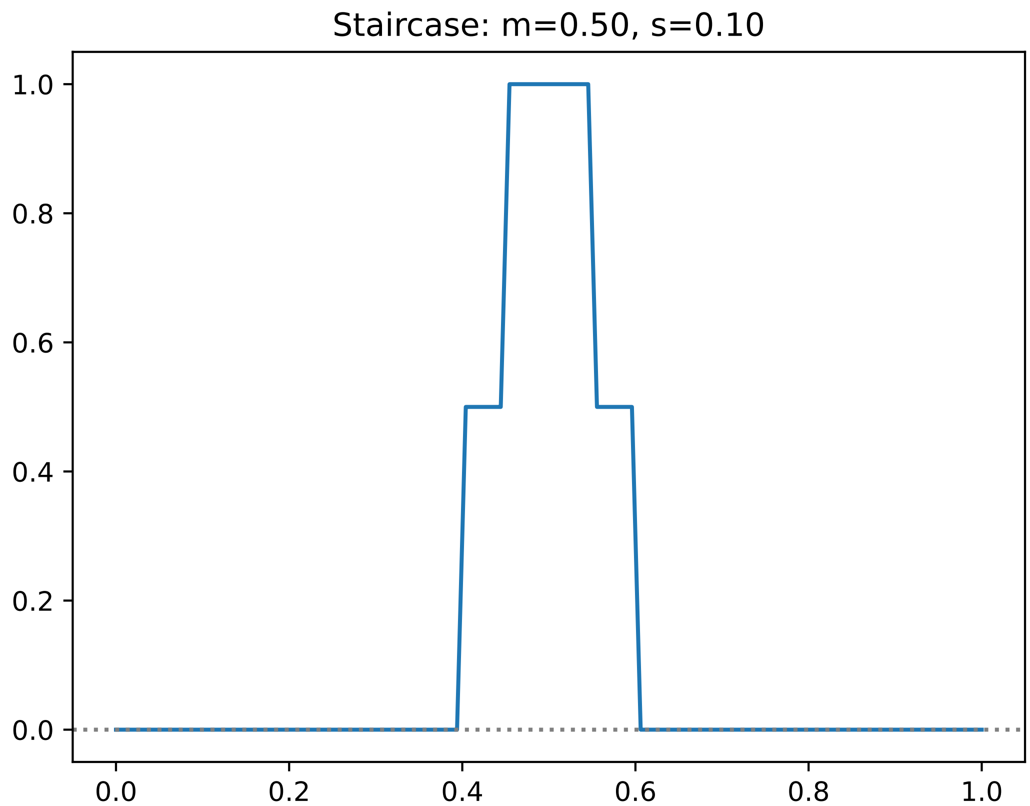
- **params** (*Array*) – parameters of the growth function
- **X** (*Array*) – potential

Returns
A field

`leniax.growth_functions.staircase(params, X)`

Staircase function

$$y = \begin{cases} 0.5, & \text{if } X \geq m - s \text{ and } X < m - \frac{s}{2} \\ 1, & \text{if } X \geq m - \frac{s}{2} \text{ and } X \leq m + \frac{s}{2} \\ 0.5, & \text{if } X > m + \frac{s}{2} \text{ and } X \leq m + s \\ 0, & \text{otherwise} \end{cases}$$



Parameters

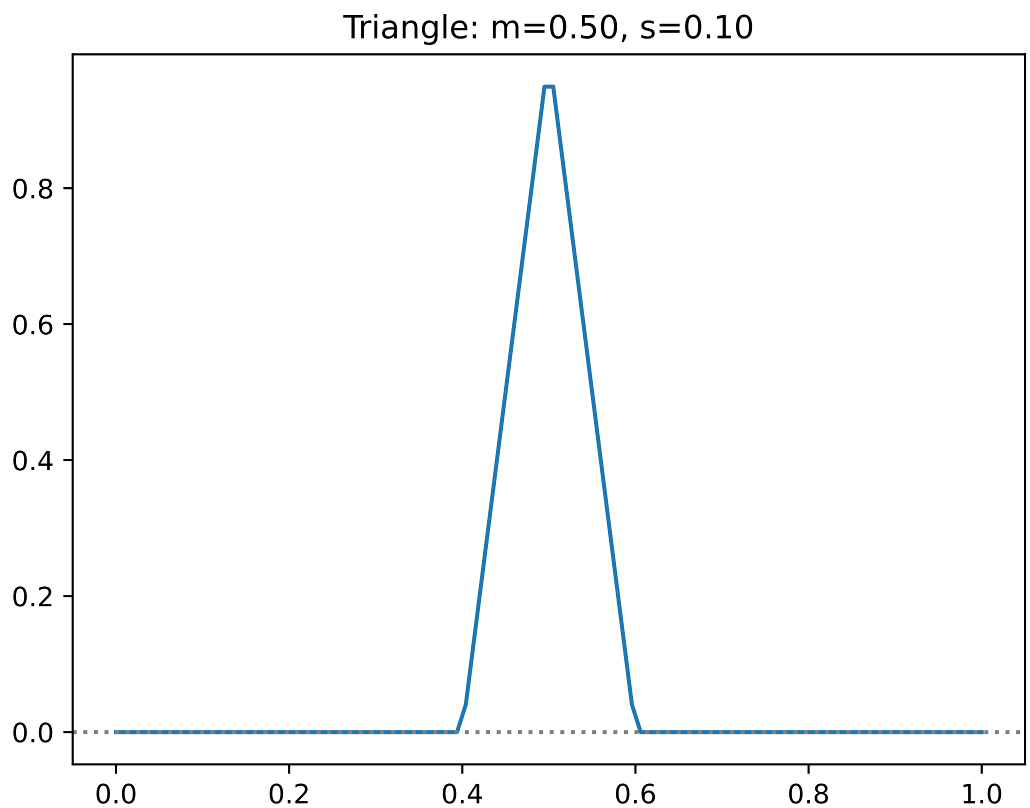
- **params** (*Array*) – Kernel function parameters
- **X** (*Array*) – Raw kernel

`leniax.growth_functions.triangle(params, X)`

Gauss function

$$y = \begin{cases} \frac{X-(m-s)}{m-(m-s)}, & \text{if } X \geq m - s \text{ and } X < m \\ \frac{X-(m+s)}{m-(m+s)}, & \text{if } X \geq m \text{ and } X \leq m + s \\ 0, & \text{otherwise} \end{cases}$$

Parameters


Triangle: m=0.50, s=0.10
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

- **params** (*Array*) – Kernel function parameters
- **X** (*Array*) – Raw kernel

CHAPTER

EIGHT

LENIAX.KERNELS PACKAGE

class leniax.kernels.**KernelMapping**(*nb_channels*, *nb_kernels*)

Explicit mapping of the computation graph

Parameters

- **nb_channels** (*int*) –
- **nb_kernels** (*int*) –

cin_kernels

list of kernel indexes grouped by channel of shape `[nb_channels, max_nb_kernels]`

Type

List[List[int]]

cin_k_params

list of kernel parameters grouped by channel of shape `[nb_channels, max_nb_kernels]`

Type

List[List]

cin_kfs

list of kernel functions grouped by channel of shape `[nb_channels, max_nb_kernels]`

Type

List[List[str]]

cin_gfs

list of growth function slugs grouped by channel of shape `[nb_channels, max_nb_kernels]`

Type

List[List[str]]

gf_params

list of growth function slug grouped by channel of shape `[nb_channels, max_nb_kernels]`

kernels_weight_per_channel

list of weights grouped by channel of shape `[nb_channels, nb_kernels]`

Type

List[List[float]]

leniax.kernels.**raw**(*R*, *k_params*, *kf_slug*, *kf_params*)

Returns k_params as an array

Parameters

- **R** – World radius **(not used)**

- **k_params** (*List*) – Kernel parameters
- **kf_slug** (*str*) – Kernel function slug
- **kf_params** (*Array*) – Kernel function params

Returns
A kernel of shape `k_params`

Return type
Array

`leniax.kernels.circle_2d`(*R*, *k_params*, *kf_slug*, *kf_params*)

Build a circle kernel

Parameters

- **R** – World radius
- **k_params** (*List*) – Kernel parameters
- **kf_slug** (*str*) – Kernel function slug
- **kf_params** (*Array*) – Kernel function params

Returns
A kernel of shape `[1, world_dims...]`

Return type
Array

CHAPTER

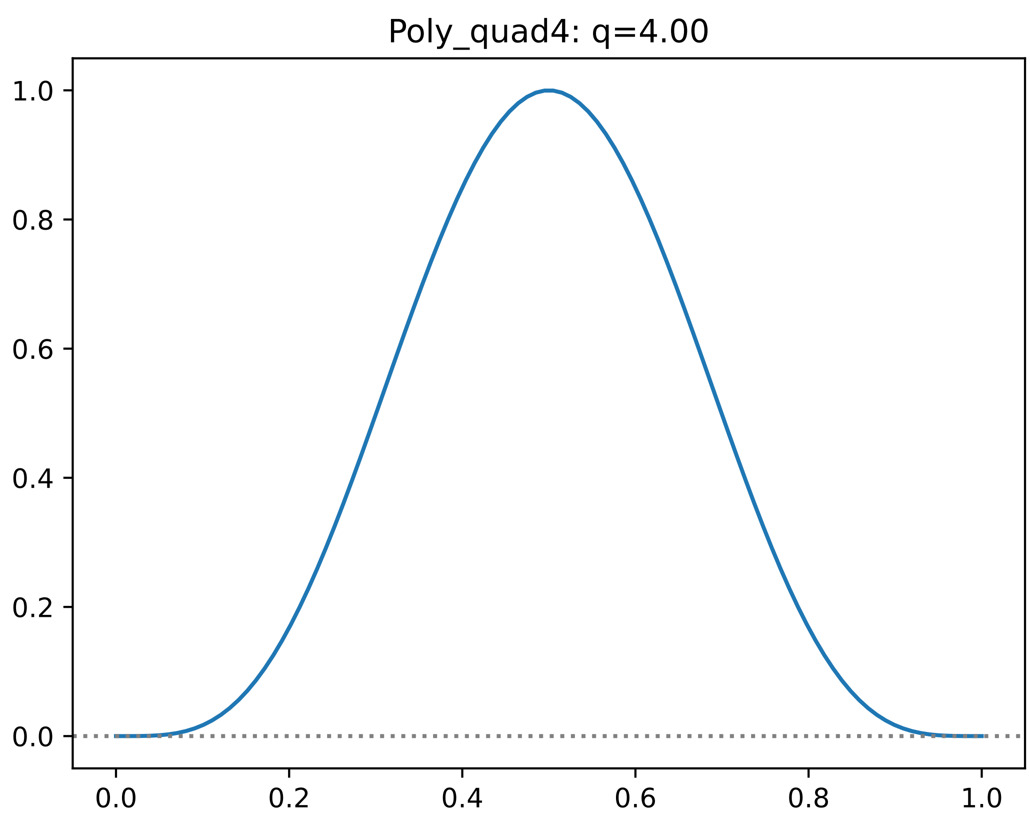
NINE

LENIAX.KERNEL_FUNCTIONS PACKAGE

`leniax.kernel_functions.`**`poly_quad`**(*params*, *X*)

Quadratic polynomial

$$y = (q * X * (1 - X))^q$$



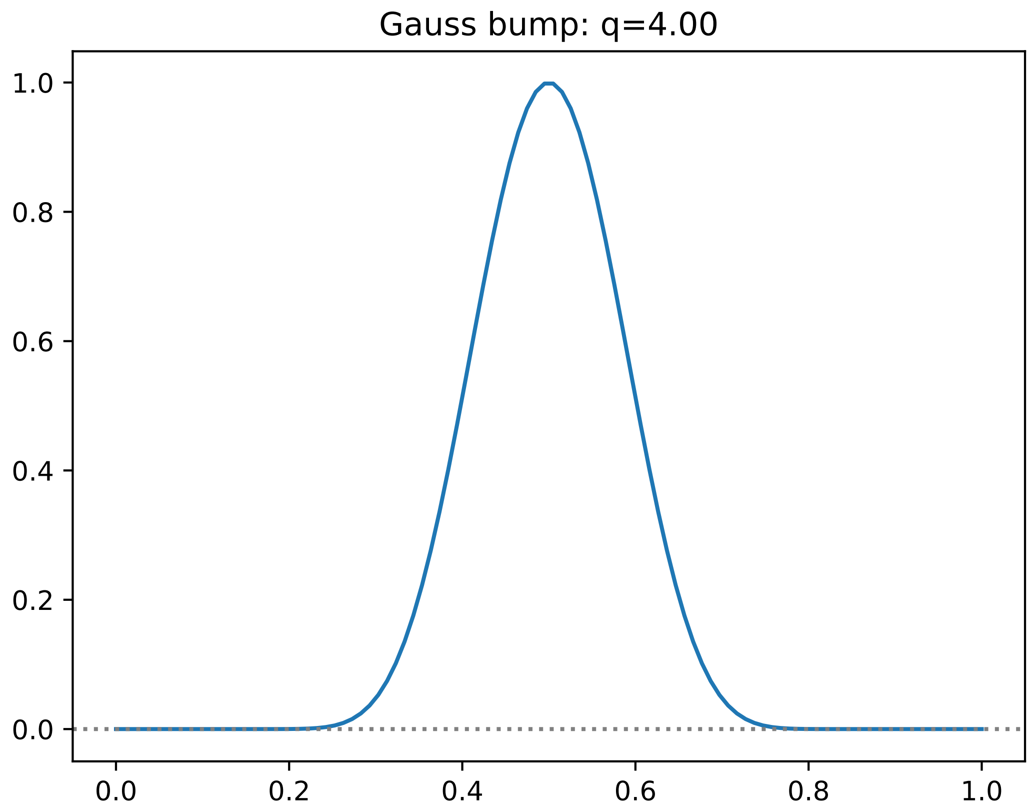
Parameters

- **`params`** (*`Array`*) – Kernel function parameters
- **`X`** (*`Array`*) – Raw kernel

leniax.kernel_functions.**gauss_bump**(*params*, *X*)

Gaussian bump function

$$y = e^{q*[q-\frac{1}{X*(1-X)}]}$$



Parameters

- **params** (*Array*) – Kernel function parameters
- **X** (*Array*) – Raw kernel

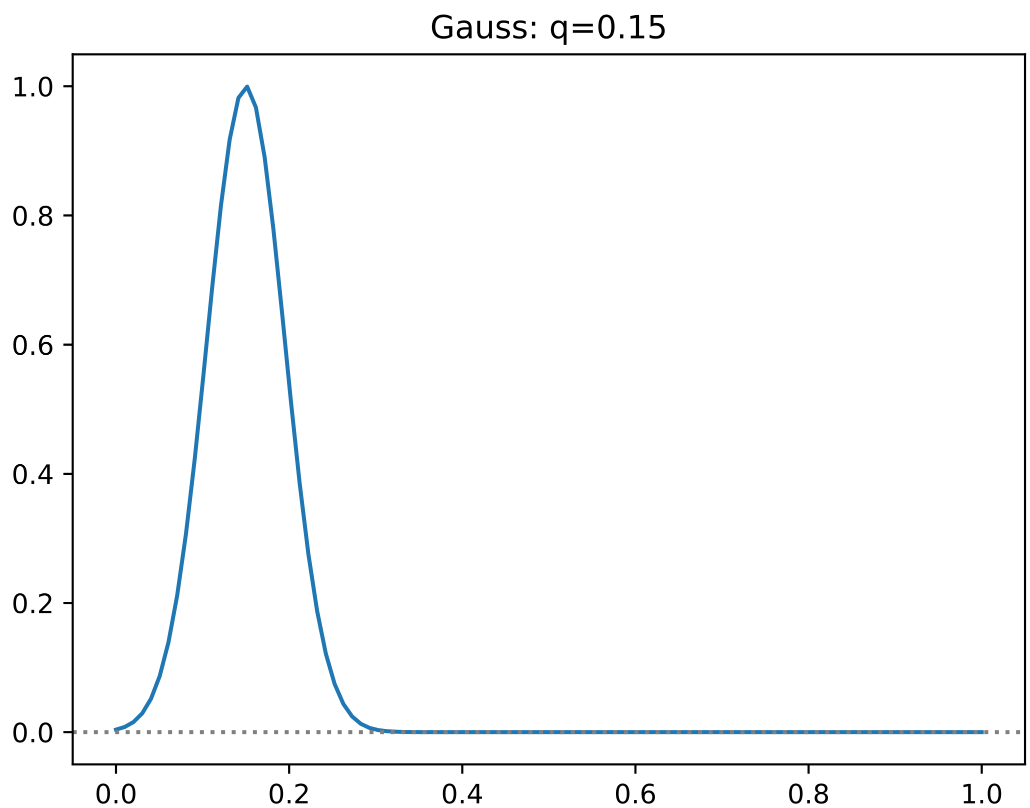
leniax.kernel_functions.**gauss**(*params*, *X*)

Gauss function

$$y = e^{-\frac{1}{2}\left(\frac{X-q}{0.3*q}\right)^2}$$

Parameters

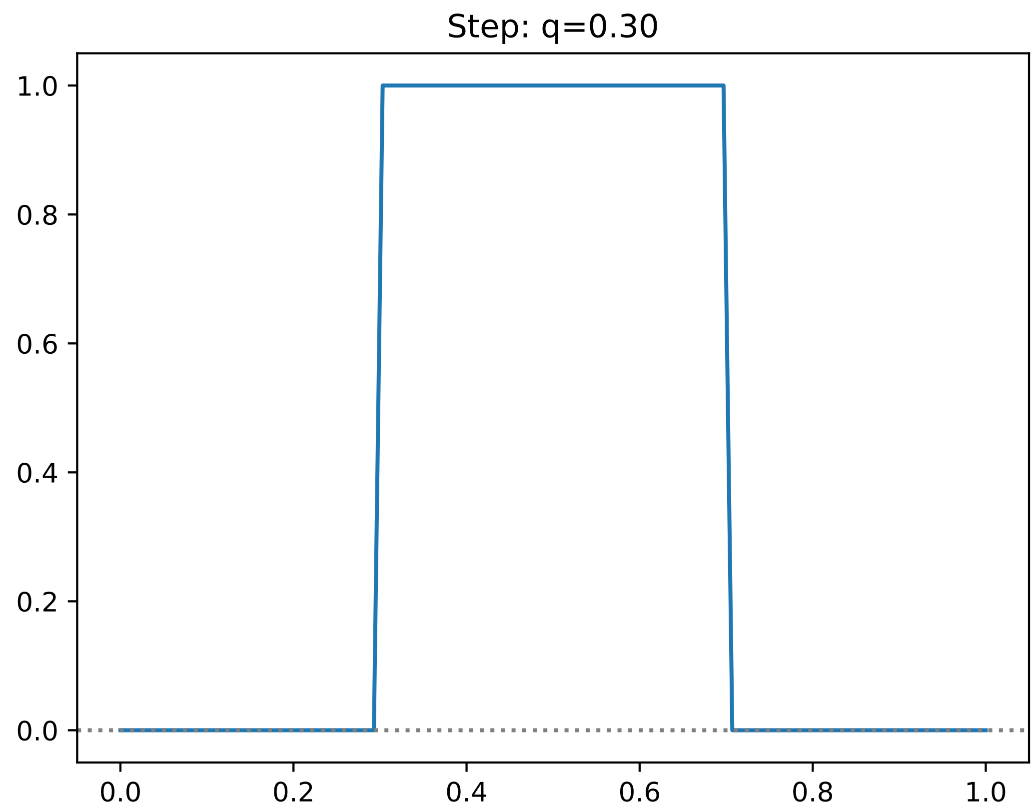
- **params** (*Array*) – Kernel function parameters
- **X** (*Array*) – Raw kernel


Gauss: q=0.15
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

`leniax.kernel_functions.step`(*params*, *X*)

Step function

$$y = \begin{cases} 1, & \text{if } X \geq q \text{ and } X \leq 1 - q \\ 0, & \text{otherwise} \end{cases}$$



Parameters

- **params** (*Array*) – Kernel function parameters
- **X** (*Array*) – Raw kernel

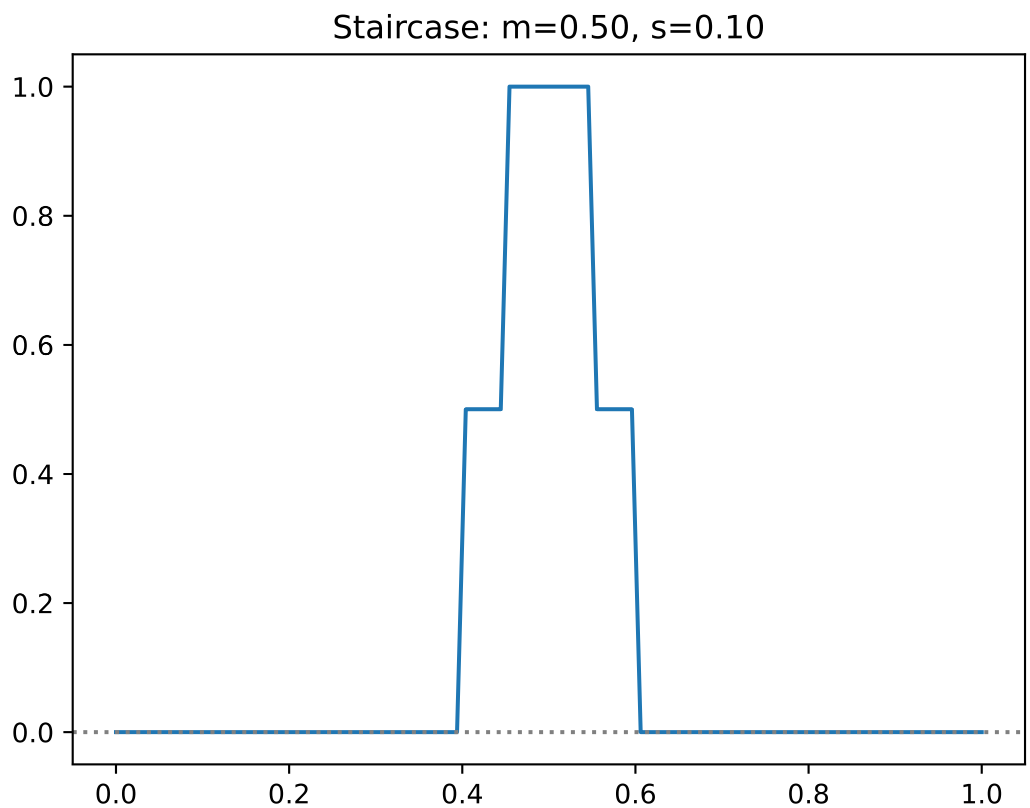
`leniax.kernel_functions.staircase`(*params*, *X*)

Staircase function

$$y = \begin{cases} 0.5, & \text{if } X \geq m - s \text{ and } X < m - \frac{s}{2} \\ 1, & \text{if } X \geq m - \frac{s}{2} \text{ and } X \leq m + \frac{s}{2} \\ 0.5, & \text{if } X > m + \frac{s}{2} \text{ and } X \leq m + s \\ 0, & \text{otherwise} \end{cases}$$

Parameters

- **params** (*Array*) – Kernel function parameters

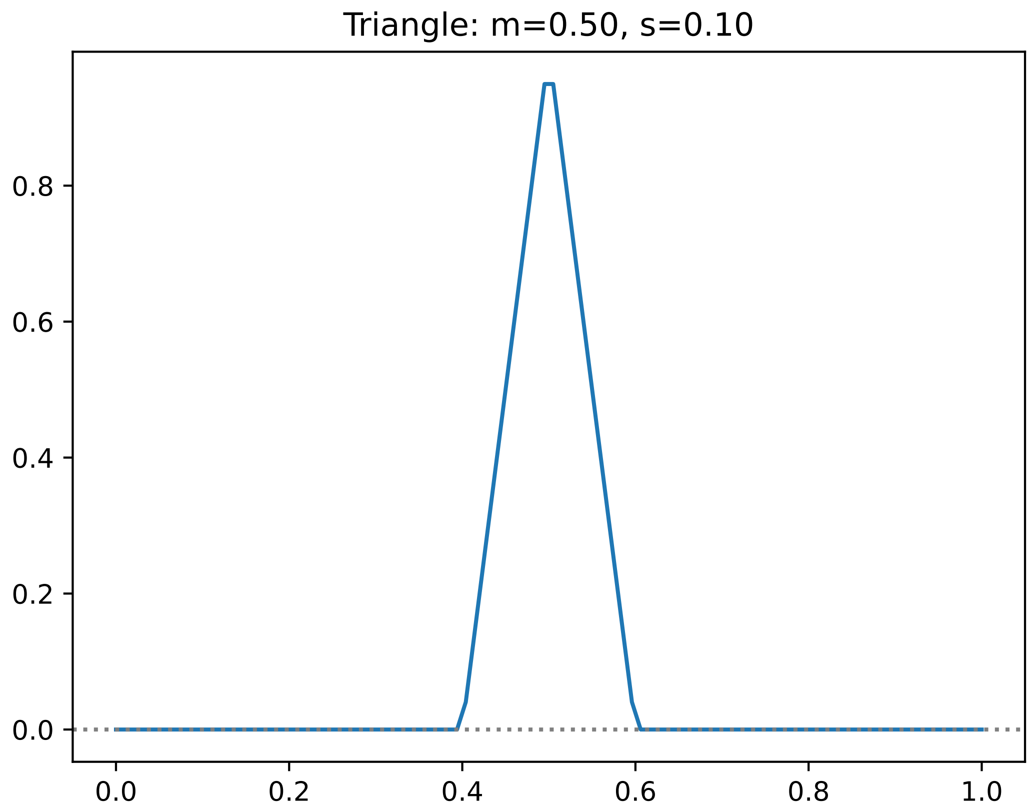

Staircase: m=0.50, s=0.10
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

- **X** (*Array*) – Raw kernel

`leniax.kernel_functions.triangle(params, X)`

Gauss function

$$y = \begin{cases} \frac{X-(m-s)}{m-(m-s)}, & \text{if } X \geq m - s \text{ and } X < m \\ \frac{X-(m+s)}{m-(m+s)}, & \text{if } X \geq m \text{ and } X \leq m + s \\ 0, & \text{otherwise} \end{cases}$$



Parameters

- **params** (*Array*) – Kernel function parameters
- **X** (*Array*) – Raw kernel

CHAPTER TEN

LENIAX.STATISTICS PACKAGE

leniax.statistics.**build_compute_stats_fn**(*world_params*, *render_params*)

Construct the conpute_statistics function

Parameters

- **world_params** (*Dict*) – World parameters dictrionnary.
- **render_params** (*Dict*) – Render parameters dictrionnary.

Returns
The compute statistics function

Return type
Callable

leniax.statistics.**check_heuristics**(*stats*)

Check heuristics on statistic data

Parameters
stats (*Dict[str, Array]*) – Simulation statistics dictionnary

Returns
An array of boolean value indicating if the heuristics are valid for each timsteps

Return type
Array

leniax.statistics.**init_counters**(*N*)

Initialize different counters used in heuristics decisions

Parameters
N (*int*) – Number of simulated timesteps

Returns
Adictionnary of counters

Return type
Dict[str, *Array*]

leniax.statistics.**min_channel_mass_heuristic**(*epsilon*, *channel_mass*)

Check if a total mass per channel is below the threshold

Parameters

- **epsilon** (*float*) – A very small value to avoid division by zero
- **channel_mass** (*Array*) – Total mass per channel of shape `[N, C]`

Returns
A boolean array of shape [N]

Return type
Array

`leniax.statistics.`**`max_channel_mass_heuristic`**(*init_channel_mass*, *channel_mass*)

Check if a total mass per channel is above the threshold

Parameters

- **init_channel_mass** (*Array*) – Initial mass per channel of shape [N, C]
- **channel_mass** (*Array*) – Total mass per channel of shape [N, C]

Returns
A boolean array of shape [N]

Return type
Array

`leniax.statistics.`**`min_mass_heuristic`**(*epsilon*, *mass*)

Check if the total mass of the system is below the threshold

Parameters

- **epsilon** (*float*) – A very small value to avoid division by zero
- **mass** (*Array*) – Total mass of shape [N]

Returns
A boolean array of shape [N]

Return type
Array

`leniax.statistics.`**`max_mass_heuristic`**(*init_mass*, *mass*)

Check if a total mass per channel is above the threshold

Parameters

- **init_mass** (*Array*) – Initial mass per channel of shape [N]
- **mass** (*Array*) – Total mass per channel of shape [N]

Returns
A boolean array of shape [N]

Return type
Array

`leniax.statistics.`**`monotonic_heuristic`**(*sign*, *previous_sign*, *monotone_counter*)

Check if the mass variation is being monotonic for too many timesteps

Parameters

- **sign** (*Array*) – Current sign of mass variation of shape [N]
- **previous_sign** (*Array*) – Previous sign of mass variation of shape [N]
- **monotone_counter** (*Array*) – Counter used to count number of timesteps with monotonic variations of shape [N]

Returns
A tuple representing a boolean array of shape [N] and the counter

Return type
Tuple[*Array*, *Array*]

`leniax.statistics.`**`mass_volume_heuristic`**(*mass_volume*, *mass_volume_counter*)

Check if the mass volume is above the threshold for too manye timesteps

Parameters

- **`mass_volume`** (*`Array`*) – Mass volume of shape `[N]`
- **`mass_volume_counter`** (*`Array`*) – Counter of shape `[N]` used to count number of timesteps with a volume above the threshold

Returns
A tuple representing a boolean array of shape `[N]` and the counter

Return type
Tuple[*Array*, *Array*]

`leniax.statistics.`**`stats_list_to_dict`**(*all_stats*)

Change a list of dictionnary in a dictionnary of array

Parameters
`all_stats` (*`List[Dict]`*) – List of 1-timestep statistics dictionary

Returns
A dictionnary of N-timestep array.

Return type
Dict[str, *Array*]

CHAPTER

ELEVEN

LENIAX.HELPERS PACKAGE

Leniax helper functions

Those functions are provided to ease the use of this library. See them as template gluing core functions together to achieve common usages.

`leniax.helpers.`**`init`**(*config*, *use_init_cells=True*, *fft=True*)

Construct the initial state and metadata to run a simulation.

Parameters

- **`config`** (*`Dict`*) – Leniax confguration
- **`use_init_cells`** (*`bool`*) – Set to `True` to use the `init_cells` configuration property.
- **`fft`** (*`bool`*) – Set to `True` to use FFT optimization

Returns

A 3-tuple representing the initial state, Kernel and mapping data

Return type

Tuple[*Array*, *Array*, KernelMapping]

`leniax.helpers.`**`create_init_cells`**(*world_size*, *nb_channels*, *other_cells=[]*, *offsets=[]*)

Construct the initial state

Parameters

- **`world_size`** (*`List[int]`*) – World size
- **`nb_channels`** (*`int`*) – Number of world channels
- **`other_cells`** (*`Union[Array, List[Array]]`*) – Other initial states to merge
- **`offsets`** (*`List[List[int]]`*) – Offsets used to merge other initial states

Returns

The initial state

Return type

Array

`leniax.helpers.`**`init_and_run`**(*rng_key*, *config*, *use_init_cells=True*, *with_jit=True*, *fft=True*, *stat_trunc=False*)

Initialize and simulate a Lenia configuration

To simulate a configuration with multiple initializations you must set: - `with_jit=True` so the function use the scan implementaton. - `stat_trunc=False` multiple initializations means different simulation length measured by the statistics.

Parameters

- **rng_key** (*PRNGKeyArray*) – JAX PRNG key.
- **config** (*Dict*) – Lenia configuration
- **use_init_cells** (*bool*) – Set to `True` to use the `init_cells` configuration property.
- **with_jit** (*bool*) – Set to `True` to use the jitted scan implementation
- **fft** (*bool*) – Set to `True` to use FFT optimization
- **stat_trunc** (*bool*) – Set to `True` to truncate run based on its statistics

Returns
A tuple of *[nb_iter, nb_init, nb_channels, world_dims…]* shaped cells, fields, potentials and statistics of the simulation.

Return type
Tuple[*Array*, *Array*, *Array*, *Dict*]

`leniax.helpers.`**`search_for_mutation`**(*rng_key*, *config*, *nb_scale_for_stability=1*, *use_init_cells=True*, *fft=True*, *mutation_rate=1e-05*)

Search for a stable mutation

Parameters

- **rng_key** (*PRNGKeyArray*) – JAX PRNG key.
- **config** (*Dict*) – Lenia configuration.
- **use_init_cells** (*bool*) – Set to `True` to use the `init_cells` configuration property.
- **fft** (*bool*) – Set to `True` to use FFT optimization.
- **nb_scale_for_stability** (*int*) – Number of time the configuration will be scaled and tested.
- **mutation_rate** (*float*) – Mutation rate.

Returns
A 2-tuple of a dictionnary with the best run data and the number of runs made to find it

Return type
Tuple[*Dict*, int]

`leniax.helpers.`**`search_for_init`**(*rng_key*, *config*, *fft=True*)

Search for a stable initial state

Parameters

- **rng_key** (*PRNGKeyArray*) – JAX PRNG key.
- **config** (*Dict*) – Lenia configuration.
- **fft** (*bool*) – Set to `True` to use FFT optimization.

Returns
A 2-tuple of a dictionnary with the best run data and the number of runs made to find it

Return type
Tuple[*Dict*, int]

`leniax.helpers.`**`build_update_fn`**(*kernel_shape*, *mapping*, *get_state_fn_slug='v1'*, *average_weight=True*, *fft=True*)

Construct an Leniax update function

An update function allows one to update a simulation state.

Parameters

- **kernel_shape** (*Tuple[int, ...]*) – Kernel shape.
- **mapping** (KernelMapping) – Mapping data.
- **get_state_fn_slug** (*str*) – Which version of Lenia should be run
- **fft** (*bool*) – Set to True to use FFT optimization
- **average_weight** (*bool*) –

Returns
A Leniax update function

Return type
Callable

leniax.helpers.**build_get_potential_fn**(*kernel_shape*, *true_channels=None*, *fft=True*, *channel_first=True*)

Construct an Leniax potential function

A potential function allows one to compute the potential from a Lenia state.

Parameters

- **kernel_shape** (*Tuple[int, ...]*) – Kernel shape.
- **true_channels** (*Optional[List[bool]]*) – Boolean array indicating the true potential channels
- **fft** (*bool*) – Set to True to use FFT optimization
- **channel_first** (*bool*) –

Returns
A Leniax potential function

Return type
Callable

leniax.helpers.**build_get_field_fn**(*cin_gfs*, *average=True*)

Construct an Leniax field function

A field function allows one to compute the field from a Lenia potential.

Parameters

- **cin_gfs** (*List[List[str]]*) – List of growth functions per channel.
- **average** (*bool*) – Set to True to average instead of summing input channels

Returns
A Leniax field function

Return type
Callable

leniax.helpers.**dump_assets**(*save_dir*, *config*, *all_cells*, *stats_dict*, *colormaps=[]*, *transparent_bg=False*)

Dump a set of interesting assets.

Those assets include:

- Simulation statistics (plots and data)
- Kernels and growth functions plots
- Last frame

- Video and Gif of the simulation

Parameters

- **save_dir** (*str*) – directory used to save assets.
- **config** (*Dict*) – Leniax configuration.
- **all_cells** (*Array*) – Simulation data of shape `[nb_iter, C, world_dims...]`.
- **stats_dict** (*Dict*) – Leniax statistics dictionnary.
- **colormaps** (*List*) – A List of matplotlib compatible colormap.
- **transparent_bg** (*bool*) – Set to `True` to make the background transparent.

`leniax.helpers.`**`dump_last_frame`**(*save_dir*, *all_cells*, *center_and_crop=True*, *colormap=None*)

Dump the last frame of the simulation

The dumped last frame is called **last_frame.png**.

Parameters

- **save_dir** (*str*) – directory used to save assets.
- **all_cells** (*Array*) – Simulation data of shape `[nb_iter, C, world_dims...]`.
- **center_and_crop** (*bool*) – Set to `True` to center the stable pattern and crop the margin.
- **colormap** – A matplotlib compatible colormap.

`leniax.helpers.`**`dump_frame`**(*save_dir*, *filename*, *cells*, *center_and_crop=True*, *colormap=None*)

Dump a Lenia state as a image

Parameters

- **save_dir** (*str*) – directory used to save assets.
- **filename** (*str*) – File name.
- **cells** (*Array*) – A Lenia state of shape `[C, world_dims...]`.
- **center_and_crop** (*bool*) – Set to `True` to center the stable pattern and crop the margin.
- **colormap** – A matplotlib compatible colormap.

`leniax.helpers.`**`dump_viz_data`**(*save_dir*, *config*, *stats_dict*)

Dump vizualization data as JSON

Parameters

- **save_dir** (*str*) – directory used to save assets.
- **config** (*Dict*) – Leniax configuration.
- **stats_dict** (*Dict*) – Leniax statistics dictionnary.

`leniax.helpers.`**`plot_kernels`**(*save_dir*, *config*)

Plots kernels and growth functions

Parameters

- **save_dir** (*str*) – directory used to save assets.
- **config** (*Dict*) – Leniax configuration.

CHAPTER

TWELVE

LENIAX.UTILS PACKAGE

leniax.utils.**get_container**(*omegaConf*, *main_path*)

Retrieve the populated container from an omegaConf

This functions is used to: - Populate missing mandatory value in Leniax configuration - Handle versioning update if necessary

```
# Use with Hydra as follow
@hydra.main(config_path=config_path, config_name=config_name)
def run(omegaConf: DictConfig) -> None:
    config = leniax_utils.get_container(omegaConf, config_path)
```

Parameters

- **omegaConf** (*DictConfig*) – Hydra's Omega configuration
- **main_path** (*str*) – Absolute path of the configuration directory

Returns

A populated python dictionnary

Return type

Dict

leniax.utils.**update_config_v1_v2**(*config*)

Update a configuration from version 1 to version 2

Parameters

config (*Dict*) – A v1 configuration

Returns

A v2 configuration

Return type

Dict

leniax.utils.**get_param**(*dic*, *key_string*)

Retrieve a parameter in a dictionnary using a string

Parameters

- **dic** (*Dict*) – Dictionnary
- **key_string** (*str*) – String representing the parameter path in the dicitonnary

Returns

The parameter

Return type
Any

`leniax.utils.`**`set_param`**(*dic*, *key_string*, *value*)

Set a parameter in a dictionnary using a string

Parameters

- **`dic`** (*`Dict`*) – Dictionnary
- **`key_string`** (*`str`*) – String representing the parameter path in the dicitonnary
- **`value`** (*`Any`*) – Value to be set

`leniax.utils.`**`st2fracs2float`**(*st*)

Convert a string of fraction into an list of floats

String example: `st = '1,2/3,6.7'`

Parameters
`st` (*`str`*) – String of fractions.

Returns
An list of float

Return type
List[float]

`leniax.utils.`**`merge_cells`**(*cells*, *other_cells*, *offset=None*)

Merge cells together using addition

Parameters

- **`cells`** (*`Array`*) – Base cells
- **`other_cells`** (*`Array`*) – Cells to be merged into the base cells
- **`offset`** (*`Optional[List[int]]`*) – Offsets in each dimensions

Returns
Resulting cells

Return type
Array

`leniax.utils.`**`center_world`**(*cells*, *field*, *potential*, *shift_idx*, *axes*)

Vectorized version of center_world. Takes similar arguments as center_world but with additional array axes over which center_world is mapped.

Original documentation:

Rool cells, field and potential

Args:
cells: Lenia state of shape `[bs, other_axes...]` field: Lenia field `[bs, other_axes... ]` potential: Lenia potential `[bs, other_axes...]` shift_idx: Amount to roll of shape `[bs, nb_axes]`` axes: Axes to roll

Returns:
Updated cells, field and potential

Parameters

- **`cells`** (*`Array`*) –

- **field** (*Array*) –
- **potential** (*Array*) –
- **shift_idx** (*Array*) –
- **axes** (*Tuple[int, ...]*) –

Return type
Tuple[*Array*, *Array*, *Array*]

leniax.utils.**crop_zero**(*kernels*)
Crop zero values out for 3 and 4 dimensions array

Parameters
kernels (*Array*) – A 3 or 4 dimension kernels

Returns
The croppped kernels

Return type
Array

leniax.utils.**auto_center**(*cells*)
Automatically center cells on its total mass centroid

Parameters
cells (*Array*) – Lenia state

Returns
The mass centered Lenia state

Return type
Array

leniax.utils.**center_and_crop**(*cells*)
Automatically center cells on its total mass centroid and crop zeros values

Parameters
cells (*Array*) – Lenia state

Returns
The mass centered cropped Lenia state

Return type
Array

leniax.utils.**get_image**(*cells_buffer*, *pixel_size*, *colormap*)
Convert a numpy array into a PIL image

Parameters

- **cells_buffer** (*ndarray*) – A Lenia state of shape [C, world_dims...]
- **pixel_size** (*int*) – Size of each state pixels in the image
- **colormap** – A matplotlib compatible colormap

Returns
A PIL image

Return type
<module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/leniax/envs/latest/lib/python3.7/site-packages/PIL/Image.py'>

`leniax.utils.`**`plot_stats`**(*save_dir*, *stats_dict*)

Plot Leniax statistics

Parameters

- **`save_dir`** (*`str`*) – directory used to save assets.
- **`stats_dict`** (*`Dict`*) – Statistics dictionnary

`leniax.utils.`**`generate_beta_faces`**(*nb_betas*, *denominator*)

Generate a grid of all the valid beta values given a maximum number
of beta values.

This function makes sense only if we normalize our kernels.

Parameters

- **`nb_betas`** (*`int`*) – Maximum number of betas values
- **`denominator`** (*`int`*) – Denominator to construct the betas grid

Returns
The list of possible list of beta values

Return type
List[*List*[*List*[float]]]

`leniax.utils.`**`check_dir`**(*dir*)

Ensure a directory exist and is not a file

Parameters
`dir` (*`str`*) – Checked directory

`leniax.utils.`**`save_config`**(*save_dir*, *config*)

Save a configuration file

Parameters

- **`save_dir`** (*`str`*) – directory used to save assets.
- **`config`** (*`Dict`*) – Leniax configuration

`leniax.utils.`**`print_config`**(*config*)

Pretty print a Leniax configuration

Parameters
`config` (*`Dict`*) – Leniax configuration

`leniax.utils.`**`load_img`**(*fullpath*, *resize*)

Load an image as a np.array

Parameters

- **`fullpath`** (*`str`*) – Absolute image path
- **`resize`** (*`Tuple[int, int]`*) – Resize factors for the image dimensions

Returns
The image as a np.array

Return type
Array

`leniax.utils.set_log_level(config)`

Set the python logging root level

Parameters
`config` (`Dict`) – Leniax configuration

`leniax.utils.seed_everything(seed)`

Seed all the depedencies used by Leniax

Parameters
`seed` (`int`) – A seed integer

Returns
A JAX PRNG key

Return type
PRNGKeyArray

`leniax.utils.get_needed_memory(config, nb_sols=1)`

Compute an approximate of the needed memory by different kind of simulations

Parameters

- `config` (`Dict`) – Leniax configuration
- `nb_sols` (`int`) – How many solutions will be simulated at the same time

Returns
A dictionnary witth different memory requirements

CHAPTER

THIRTEEN

LENIAX.LOADER PACKAGE

leniax.loader.**make_array_compressible**(*cells*)

Round values so the array can be encoded using a subset of utf-8 characters

Parameters
`cells` (*Array*) – Cells state.

Returns
The rounded cells state

Return type
Array

leniax.loader.**compress_array**(*cells*)

Compress a cells state into a base64 utf-8 string.

Note: The cells state in float32 is first encoded as int32.

That state is then flattened and converted into raw bytes of length 4 in little endian.

Finally we prepend the total number of bytes of the state and append the shape as bytes.

Finally, we compress the array using the gzip algorithm and the resulting bytes are encoded as base64 in the utf-8 encoding.

Parameters
`cells` (*Array*) – Cells state

Returns
Cells state encoded as a string.

Return type
str

leniax.loader.**decompress_array**(*string_cells*, *nb_dims=0*)

Best effort helpers which tries all existing decompress function built so far

Parameters

- **string_cells** (*str*) – A string encoded cells state.
- **nb_dims** (*int*) – the number of dimensions in the cells state.

Returns
The decoded cells state array.

Return type
Array

`leniax.loader.`**`decompress_array_gzip`**`(`*string_cells*`)`

Decompress string encoded cells state using the gzip algorithm

Parameters
string_cells (*str*) – A base64 string encoded cells state.

Returns
The decoded cells state array.

Return type
Array

`leniax.loader.`**`decompress_array_base64`**`(`*string_cells*`)`

Decompress string encoded cells state using only the base64 algorithm

Parameters
string_cells (*str*) – A base64 string encoded cells state.

Returns
The decoded cells state array.

Return type
Array

`leniax.loader.`**`ch2val`**`(`*c*`)`

Map characters to integers

Parameters
c (*str*) – A character.

Returns
An integer.

Return type
int

`leniax.loader.`**`val2ch`**`(`*v*`)`

Map integers to characters

Parameters
v (*int*) – An integer

Returns
A character.

Return type
str

`leniax.loader.`**`load_raw_cells`**`(`*config*, *use_init_cells=True*`)`

Load and decompress cells state contained in a Leniax configuration.

Parameters

- **config** (*Dict*) – Leniax configuration
- **use_init_cells** (*bool*) – Set to `True` to use the `init_cells` configuration property.

Returns
A Leniax cells state.

Return type

Array

CHAPTER

FOURTEEN

LENIAX.QD PACKAGE

leniax.qd.**build_eval_lenia_config_mem_optimized_fn**(*qd_config*, *fitness_coef=1.0*, *fft=True*)

Construct the evaluation function for the mem_optimized runner function

Parameters

- **qd_config** (*Dict*) – QD configuration
- **fitness_coef** (*float*) – Multiply all returned fitness by this coefficient before ranking (mainly used to negate raw fitness values)
- **fft** (*bool*) – Set to `True` to use FFT optimization

Returns

The evaluation function.

Return type

Callable

leniax.qd.**get_dynamic_args**(*qd_config*, *leniax_sols*, *fft=True*)

Prepare dynamic arguments to be used in parrallel simulations

Parameters

- **qd_config** (*Dict*) – Leniax QD configuration
- **leniax_sols** (*List[LeniaIndividual]*) – Candidate Leniax solutions
- **fft** (*bool*) – Set to `True` to use FFT optimization

Returns

A 2-tuple representing a JAX PRNG key and a 5-tuple of the batch of simulation parameters of shape `[N_sols, N_init, nb_channels, world_dims...]`

Return type

Tuple[*PRNGKeyArray*, *Tuple*[*Array*, *Array*, *Array*, *Array*, *Array*]]

leniax.qd.**update_individuals**(*inds*, *stats*, *fitness_coef=1.0*)

Update Lenia individuals

> **Warning:** In the statistics dictionnary, the `N` statistic is of shape `[N_sols, N_init]`.

Parameters

- **inds** (*List[LeniaIndividual]*) – Evaluated Lenia individuals
- **stats** (*Dict[str, Array]*) – `Dict[str, [N_sols, nb_iter, N_init]]`

- **fitness_coef** – Multiply all returned fitness by this coefficient before ranking (mainly used to negate raw fitness values)

Returns
Lpdate Lenia individuals

Return type
List[LeniaIndividual]

leniax.qd.**run_qd_search**(*rng_key*, *qd_config*, *optimizer*, *fitness_domain*, *eval_fn*, *log_freq=1*, *n_workers=-1*)

Run a Quality-diveristy search

> **Warning:**
> - n_workers == -1 means that your evaluation functions handles parallelism
> - n_workers == 0 means that you want to use a sinple python loop function
> - n_workers > 0 means that you want to use python spawn mechanism

Parameters

- **rng_key** (*PRNGKeyArray*) – jax PRNGKey
- **qd_config** (*Dict*) – QD configuration
- **optimizer** (*Optimizer*) – pyribs Optimizer
- **fitness_domain** (*Tuple[int, int]*) – a 2-tuple of ints representing the fitness bounds
- **eval_fn** (*Callable*) – The evaluation function
- **log_freq** (*int*) – Logging frequency
- **n_workers** (*int*) – Number of workers used to eval a set of candidate solutions

Returns
Qd metrics

Return type
Dict[str, *Dict*[str, list]]

leniax.qd.**load_qd_grid_and_config**(*grid_fullpath*)

Helper function to load the QD grid and configuration

Parameters
grid_fullpath (*str*) – The absolute path the pickled grid.

Returns
A 2-tuple representing the QD grid and configuration.

Return type
Tuple[*ArchiveBase*, *Dict*]

leniax.qd.**render_best**(*grid*, *fitness_threshold*)

Helper function to render configurations above the threshold

Parameters

- **grid** (*ArchiveBase*) – QD grid.
- **fitness_threshold** (*float*) – Threshold definie what is among the best fitness values

`leniax.qd.`**`render_found_lenia`**`(`*enum_lenia*`)`

Render one Lenia

Parameters

`enum_lenia` (*`Tuple[int, LeniaIndividual]`*) – A 2-tuple representing and index and a Lenia individual.

`leniax.qd.`**`save_ccdf`**`(`*archive*, *fullpath*`)`

Saves a CCDF showing the distribution of the archive's objective values.

Note: CCDF = Complementary Cumulative Distribution Function

The CCDF plotted here is not normalized to the range `(0,1)`.

This may help when comparing CCDF's among archives with different amounts of coverage (i.e. when one archive has more cells filled).

Parameters

- **`archive`** (*`ArchiveBase`*) – Archive containing the experiment results.
- **`fullpath`** (*`str`*) – Absolute path to an image file.

`leniax.qd.`**`save_metrics`**`(`*metrics*, *save_dir*`)`

Plot and save QD metrics.

Parameters

- **`metrics`** (*`Dict[str, Dict[str, list]]`*) – Dictionnary of metrics.
- **`save_dir`** (*`str`*) – Absolute path of the saving directory.

`leniax.qd.`**`save_heatmap`**`(`*archive*, *fitness_domain*, *fullpath*`)`

Save QD heatmap

Parameters

- **`archive`** (*`ArchiveBase`*) – Archive containing the experiment results.
- **`fitness_domain`** (*`Tuple`*) – Bounds of fitness values.
- **`fullpath`** (*`str`*) – Absolute path of the file.

`leniax.qd.`**`save_parallel_axes_plot`**`(`*archive*, *fitness_domain*, *fullpath*`)`

Save parallel axes plot.

Parameters

- **`archive`** (*`ArchiveBase`*) – Archive containing the experiment results.
- **`fitness_domain`** (*`Tuple`*) – Bounds of fitness values.
- **`fullpath`** (*`str`*) – Absolute path of the file.

`leniax.qd.`**`save_emitter_samples`**`(`*archive*, *fitness_domain*, *sols*, *fits*, *bcs*, *fullpath*, *title*`)`

Save emitter sampling points.

Parameters

- **`archive`** (*`ArchiveBase`*) – Archive containing the experiment results.
- **`fitness_domain`** (*`Tuple`*) – Bounds of fitness values.

- **sols** (*List*) – Solutions parameters.
- **fits** (*List*) – Fitness measurements.
- **bcs** (*List*) – Behaviours measurements.
- **fullpath** (*str*) – Absolute path of the file.
- **title** (*str*) – Title of the image.

leniax.qd.**save_all**(*current_iter*, *optimizer*, *fitness_domain*, *sols*, *fits*, *bcs*)

Helper function to all kind of vizualisation for a QD iteration.

Parameters

- **current_iter** (*int*) – Current QD iteration.
- **optimizer** – Pyribs compatible optimizer.
- **fitness_domain** (*Tuple*) – Bounds of fitness values.
- **sols** (*List*) – Solutions parameters.
- **fits** (*List*) – Fitness measurements.
- **bcs** (*List*) – Behaviours measurements.

CHAPTER

FIFTEEN

LENIAX.LENIA PACKAGE

class leniax.lenia.**LeniaIndividual**(*config*, *rng_key*, *params=[]*)

A Lenia individual used by QD algorithms

Note: The philosophy of the lib is to have parameters sampled from the same domain And then scaled by custom functions before being used in the evaluation function To sum up:

- All parameters are generated in the sampling_domain
- the dimension parameter is the number of parameter
- **in the eval function:**
  1. You scale those parameters
  2. You create the configuration from those parameters
  3. You evaluate the configuration
  4. you set fitness and features

Parameters

- **config** (*Dict*) –
- **rng_key** (*PRNGKeyArray*) –
- **params** (*List*) –

fitness

QD fitness value

Type
float

features

List of QD behaviour values

Type
List[float]

qd_config

QD configuration

Type
Dict

rng_key

JAX PRNG key

Type
jax._src.prng.PRNGKeyArray

params

A list of parameters to be updated by QD

Type
List

leniax.lenia.**get_update_config**(*genotype*, *raw_values*)

Update the QD configuration using raw_values

Parameters

- **genotype** (*Dict*) – A dictionnary of genotype value to be updated
- **raw_values** (*List*) – Raw values for the update

Returns
A dictionnary mapping keys with updated values

Return type
Dict

leniax.lenia.**linear_scale**(*raw_value*, *domain*)

Scale linearly raw_value in domain

Parameters

- **raw_value** (*float*) – a value in `[0, 1]`
- **domain** (*Tuple[float, float]*) – Domain bounding the final value

Returns
The scaled value

Return type
float

leniax.lenia.**log_scale**(*raw_value*, *domain*)

Scale logarithmically raw_value in domain

Parameters

- **raw_value** (*float*) – a value in `[0, 1]`
- **domain** (*Tuple[float, float]*) – Domain bounding the final value

Returns
The scaled value

Return type
float

LENIAX.VIDEO PACKAGE

leniax.video.**render_video**(*save_dir*, *all_cells*, *render_params*, *colormaps*, *prefix=''*, *transparent_bg=False*)

Render a Leniax video

```
ffmpeg
    -format='rawvideo',
    -pix_fmt='rgba',
    -s=f"{width}x{height}",
    -framerate=30,
    -i pipe:
    -c:v libx264
    -profile:v high
    -preset slow
    -movflags faststart
    -pix_fmt yuv420p
    out.mp4
```

Parameters

- **save_dir** (*str*) – directory used to save assets.
- **all_cells** (*Array*) – Simulation data of shape [nb_iter, C, H, W] .
- **render_params** (*Dict*) – Rendering configuration.
- **colormaps** (*Union[List, Any]*) – A List of matplotlib compatible colormaps
- **prefix** (*str*) – Video name prefix
- **transparent_bg** (*bool*) – Set to True to make the background transparent.

leniax.video.**render_gif**(*video_fullpath*)

Render a video as a GIF

```
ffmpeg
    -i $video_fullpath
    -vf "fps=30,scale=width:-1:flags=lanczos,split[s0][s1];[s0]palettegen[p];
↪[s1][p]paletteuse"
    -loop 0
   \$video_fullpath.gif
```

Parameters

video_fullpath – Fullpath of a video.

leniax.video.**render_qd_search**(*output_fullpath*, *framerate=10*)

Render a video from QD vizualisation

```
ffmpeg
    -framerate $framerate -i '%4d-emitter_0.png'
    -framerate $framerate -i '%4d-emitter_1.png'
    -framerate $framerate -i '%4d-emitter_2.png'
    -framerate $framerate -i '%4d-emitter_3.png'
    -framerate $framerate -i '%4d-archive_ccdf.png'
    -framerate $framerate -i '%4d-archive_heatmap.png'
    -filter_complex "[0:v][1:v]hstack[h1];
        [2:v][3:v]hstack[h2];
        [4:v][5:v]hstack[h3];
        [h1][h2]vstack[v1];
        [v1][h3]vstack[o]"
    -map "[o]"
    \$output_fullpath.mp4
```

Parameters

- **output_fullpath** – Fullpath of the video file.
- **framerate** – Frame rate of the video.

CHAPTER SEVENTEEN

LENIAX.COLORMAPS PACKAGE

Leniax colormaps

class leniax.colormaps.**PerceptualGradientColormap**(*name*, *hex_bg_color*, *hex_colors*)

Perceptual gradient colormap

Parameters

- **name** (*str*) –
- **hex_bg_color** (*str*) –
- **hex_colors** (*List[str]*) –

name

Colormap name

Type
str

hex_bg_color

Background color (in hexadecimal)

Type
str

hex_colors

List of colors used to create the perceptual gradient

Type
List[str]

cmap

Matplotlib `ListedColormap`

Type
matplotlib.colors.ListedColormap

PYTHON MODULE INDEX

l

INDEX

M

N

P

Q

R

S