sme_contrib

Liam Keegan

Oct 02, 2023

API REFERENCE

A collection of useful modules for use with sme, the python interface to Spatial Model Editor.

CHAPTER

ONE

INSTALLATION

To install from PyPI:

```
pip install sme-contrib
```

CHAPTER

TWO

API REFERENCE

sme_contrib.optimize	Optimization, fitting and analysis
sme_contrib.plot	Plotting

2.1 sme_contrib.optimize

Optimization, fitting and analysis

Functions

abs_diff(x, y)	Absolute difference between two arrays
hessian(f, x0[, rel_eps, processes])	Approximate Hessian of function `f` at point `x0`
minimize(f, lowerbounds, upperbounds[, ...])	Minimize function `f` using particle swarm
rescale(x, new_max_element)	Rescale an array

2.1.1 sme_contrib.optimize.abs_diff

sme_contrib.optimize.**abs_diff**(x, y)

Absolute difference between two arrays

$\frac{1}{2}\sum_i (x_i - y_i)^2$

Parameters

- **x** (*numpy.array*) – The first array
- **y** (*numpy.array*) – The second array

Returns
absolute difference between the two arrays

Return type
float

2.1.2 sme_contrib.optimize.hessian

sme_contrib.optimize.**hessian**(*f*, *x0*, *rel_eps=0.01*, *processes=None*)

Approximate Hessian of function `f` at point `x0`

Uses a finite difference approximation where the step size used for each element `i` of `x0` is `rel_eps * x[i]`.

Requires $N^2 + N + 1$ evalulations of `f`, where N is the number of elements of `x0`

The evaluations of `f` are done in parallel, so `f` must be a thread-safe function that can safely be called from multiple threads at the same time.

Note: This choice of step size allows the different elements of x0 to have vastly different scales without causing numerical instabilities, but it will fail if an element of `x0` is equal to 0.

Parameters

- **f** – The function to evaluate, it should be callable as f(x0) and return a scalar
- **x0** – The point at which to evaluate the function, a flot or list of floats.
- **rel_eps** – The relative step size to use
- **processes** – The number of processes to use (the default `None` means use all available cpu cores)

Returns
The Hessian as a 2d numpy array of floats

Return type
np.array

2.1.3 sme_contrib.optimize.minimize

sme_contrib.optimize.**minimize**(*f*, *lowerbounds*, *upperbounds*, *particles=20*, *iterations=20*, *processes=None*, *ps_options=None*)

Minimize function `f` using particle swarm

The function `f` should take an array or list of parameters `x`, and return a value: parameters will be found using particle swarm that minimize this value.

Each parameter should have a specified lower and upper bound.

The evaluations of `f` are done in parallel, so `f` must be a thread-safe function that can safely be called from multiple threads at the same time. The evaluations are parallelized over the particles for each iteration, so for good performance the number of particles should be larger than the number of processes.

Parameters

- **f** – The function to evaluate, it should be callable as f(x) and return a scalar
- **lowerbounds** – The lower bound for each element of x.
- **upperbounds** – The upper bound for each element of x.
- **particles** – The number of particles to use in the swarm
- **iterations** – The number of iterations to do

- **processes** – The number of processes to use (the default `None` means use all available cpu cores)
- **ps_options** – A map of the particle swarm hyper parameters to use

Returns
The lowest cost ps_res: The parameters that gave this lowest cost optimizer: The PySwarms optimizer object

Return type
ps_cost

2.1.4 sme_contrib.optimize.rescale

`sme_contrib.optimize.`**rescale**(*x*, *new_max_element*)

Rescale an array

Parameters

- **x** (*numpy.array*) – The array to rescale
- **new_max_element** (*float*) – The desired new maximum element value

Returns
The rescaled array

Return type
np.array

Classes

SteadyState(modelfile, imagefile, species, ...)	Steady state parameter fitting

2.1.5 sme_contrib.optimize.SteadyState

class `sme_contrib.optimize.`**SteadyState**(*modelfile*, *imagefile*, *species*, *function_to_apply_params*, *lower_bounds*, *upper_bounds*, *simulation_time=1000*, *steady_state_time=200*, *timeout_seconds=10*)

Steady state parameter fitting

Given a 2d model and an image of the target steady state distribution of a species (or the sum of multiple species), this class tries to find a set of parameters where the simulated model has a steady state solution that is as close as possible to the target image.

Note: This functionality assumes the model is 2d, i.e. it only takes into account the first z-slice of a 3d model

Parameters

- **modelfile** (*str*) – The sbml file containing the model
- **imagefile** (*str*) – The image file containing the target concentration Optionally this can instead be a dict of geometryimagefilename:targetconcentrationfilename, in which case the model is simultaneously fitted to the target concentration image steady state for each geometry image.
- **species** (*List of str*) – The species to compare to the target concentration

- **function_to_apply_params** – A function that sets the parameters in the model. This should be a function with signature `f(model, params)`, and which sets the value of the parameters to be fitted in `model` according to the values in `params`, which will be a list of floats.
- **lower_bounds** (*List of float*) – The lower bound for each parameter to be fitted
- **upper_bounds** (*List of float*) – The upper bound for each parameter to be fitted
- **simulation_time** (*float*) – The length of time to simulate the model
- **steady_state_time** (*float*) – The length of time to multiply the final rate of change of concentration. The cost function that is minimized is the sum of squares over all pixels of the difference between the final concentration and the target concentration, plus the sum of squares over all pixels of the difference between `steady_state_time` * dc/dt and zero. Multiplying the rate of change by a time makes the second term have the same units as the first term, and the relative importance of being close to steady state versus close to the desired concentration in the fit can be adjusted by altering `steady_state_time`. The larger it is, the closer the results will be to a steady state.

params

The best model parameters found

Type
numpy.array

cost_history

The history of the best cost at each iteration

Type
List of float

cost_history_pbest

The history of the mean particle best at each iteration

Type
List of float

Methods

__init__(modelfile, imagefile, species, ...)	
find([particles, iterations, processes])	Find parameters that result in a steady state concentration close to the target image
get_model()	Returns the model with best parameters applied
hessian([rel_eps, processes])	
plot_all([cmap])	Generate all plots
plot_cost_history([ax])	Plot the cost history
plot_cost_history_pbest([ax])	Plot the mean particle best cost history
plot_model_concentration([index, ax, cmap])	Plot the model concentration as a 2d heat map
plot_target_concentration([index, ax, cmap])	Plot the target concentration as a 2d heat map
plot_timeseries(simulation_time, ...[, ax])	Plot a timeseries of the sum of concentrations

sme_contrib.optimize.SteadyState.__init__

SteadyState.**__init__**(*modelfile*, *imagefile*, *species*, *function_to_apply_params*, *lower_bounds*, *upper_bounds*, *simulation_time=1000*, *steady_state_time=200*, *timeout_seconds=10*)

sme_contrib.optimize.SteadyState.find

SteadyState.**find**(*particles=20*, *iterations=20*, *processes=None*)

Find parameters that result in a steady state concentration close to the target image

Uses particle swarm to minimize the difference between the rescaled concentration and the target image, as well as the distance from a steady state solution.

Parameters

- **particles** (*int*) – The number of particles in the particle swarm
- **iterations** (*int*) – The number of particle swarm iterations
- **processes** – The number of processes to use (the default `None` means use all available cpu cores)

Returns
the best parameters found

Return type
List of float

Note: On Windows, calling this function from a jupyter notebook can result in an error message of the form *Can't get attribute 'apply_params' on <module '__main__'*, where `apply_params` is the function you have defined to apply the parameters to the model. This is a known issue with Python multiprocessing, and a workaround is to define the `apply_params` function in a separate *.py* file and import it into the notebook.

---

sme_contrib.optimize.SteadyState.get_model

SteadyState.**get_model**()

Returns the model with best parameters applied

Returns
The model with the best parameters applied

Return type
sme.Model

sme_contrib.optimize.SteadyState.hessian

SteadyState.**hessian**(*rel_eps=0.1*, *processes=None*)

sme_contrib.optimize.SteadyState.plot_all

SteadyState.**plot_all**(*cmap=None*)

Generate all plots

Helper function for interactive use in a jupyter notebook. Generates and shows all plots for user to see at a glance the results of the fit.

Parameters

cmap (*matplotlib.Colormap*) – Optionally specify the colormap to use for heatmap plots

sme_contrib.optimize.SteadyState.plot_cost_history

SteadyState.**plot_cost_history**(*ax=None*)

Plot the cost history

The cost of the best set of parameters at each iteration of particle swarm.

Parameters

ax (*matplotlib.axes._subplots.AxesSubplot*) – Optionally specify the axes to draw the plot on

Returns

The axes the plot was drawn on

Return type

matplotlib.axes._subplots.AxesSubplot

sme_contrib.optimize.SteadyState.plot_cost_history_pbest

SteadyState.**plot_cost_history_pbest**(*ax=None*)

Plot the mean particle best cost history

The mean of the best cost for each particle in the swarm, at each iteration of particle swarm.

Parameters

ax (*matplotlib.axes._subplots.AxesSubplot*) – Optionally specify the axes to draw the plot on

Returns

The axes the plot was drawn on

Return type

matplotlib.axes._subplots.AxesSubplot

sme_contrib.optimize.SteadyState.plot_model_concentration

SteadyState.**plot_model_concentration**(*index=0*, *ax=None*, *cmap=None*)

Plot the model concentration as a 2d heat map

The model concentration is normalized such that the maximum pixel intensity matches the maximum pixel intensity of the target concentration image

Parameters

- **index** (*int*) – Optionally specify index of concentration
- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – Optionally specify the axes to draw the plot on
- **cmap** (*matplotlib.Colormap*) – Optionally specify the colormap to use

Returns
The axes the plot was drawn on

Return type
matplotlib.axes._subplots.AxesSubplot

sme_contrib.optimize.SteadyState.plot_target_concentration

SteadyState.**plot_target_concentration**(*index=0*, *ax=None*, *cmap=None*)

Plot the target concentration as a 2d heat map

Parameters

- **index** (*int*) – Optionally specify index of concentration
- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – Optionally specify the axes to draw the plot on
- **cmap** (*matplotlib.Colormap*) – Optionally specify the colormap to use

Returns
The axes the plot was drawn on

Return type
matplotlib.axes._subplots.AxesSubplot

sme_contrib.optimize.SteadyState.plot_timeseries

SteadyState.**plot_timeseries**(*simulation_time*, *image_interval_time*, *ax=None*)

Plot a timeseries of the sum of concentrations

The sum of all species concentrations summed over all pixels, as a function of the simulation time. This is a convenience plot just to see by eye how close the simulation is to a steady state.

Parameters

- **simulation_time** (*float*) – The simulation time to simulate
- **image_interval_time** (*float*) – The interval in between images
- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – Optionally specify the axes to draw the plot on

Returns
The axes the plot was drawn on

Return type
matplotlib.axes._subplots.AxesSubplot

2.2 sme_contrib.plot

Plotting

Functions

colormap(color[, name])	Create a linear matplotlib colormap
concentration_heatmap(simulation_result, species)	Plot 2d heatmap of species concentration
concentration_heatmap_animation(...[, ...])	Plot 2d animated heatmap of species concentration

2.2.1 sme_contrib.plot.colormap

sme_contrib.plot.**colormap**(*color*, *name='my colormap'*)

Create a linear matplotlib colormap

The minimum value corresponds to the color black, and the maximum value corresponds to the supplied `color`.

This color can be supplied as a triplet of floats in the range from zero to one, or as a hex RGB string `"#rgb` or `"#rrggbb`.

So for example, three equivalent ways to set the `color` to red would be `(1.0, 0.0, 0.0)`, `#f00`, or `"#ff0000"`.

Parameters
color – RBG triplet of floats between 0 and 1, or hex RGB string

Returns
the Colormap

Return type
matplotlib.Colormap

2.2.2 sme_contrib.plot.concentration_heatmap

sme_contrib.plot.**concentration_heatmap**(*simulation_result*, *species*, *z_slice: int = 0*, *title=None*, *ax=None*, *cmap=None*)

Plot 2d heatmap of species concentration

Plots the concentration of species in the list `species` from the supplied `simulation_result` as a 2d heatmap.

Parameters

- **simulation_result** (*sme.SimulationResult*) – A simulation result to plot
- **species** (*List of str*) – The species to plot
- **z_slice** (*int*) – The z-slice to plot
- **title** (*str*) – Optionally specify the title

- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – Optionally specify the axes to draw the plot on
- **cmap** (*matplotlib.Colormap*) – Optionally specify the colormap to use

Returns
The axes the plot was drawn on matplotlib.image.AxesImage: The axes of the image

Return type
matplotlib.axes._subplots.AxesSubplot

2.2.3 sme_contrib.plot.concentration_heatmap_animation

sme_contrib.plot.**concentration_heatmap_animation**(*simulation_results*, *species*, *z_slice: int = 0*, *title=None*, *figsize=None*, *interval=200*)

Plot 2d animated heatmap of species concentration

Plots the concentration of species in the list `species` from the supplied list `simulation_results` as an animated 2d heatmap.

Parameters

- **simulation_results** (*List of sme.SimulationResult*) – A simulation result to plot
- **species** (*List of str*) – The species to plot
- **z_slice** (*int*) – The z-slice to plot
- **title** (*str*) – Optionally specify the title
- **figsize** (*(float, float)*) – Optionally specify the figure size
- **interval** – Optionally specify the interval in ms between images

Returns
the matplotlib animation

Return type
matplotlib.animation.ArtistAnimation

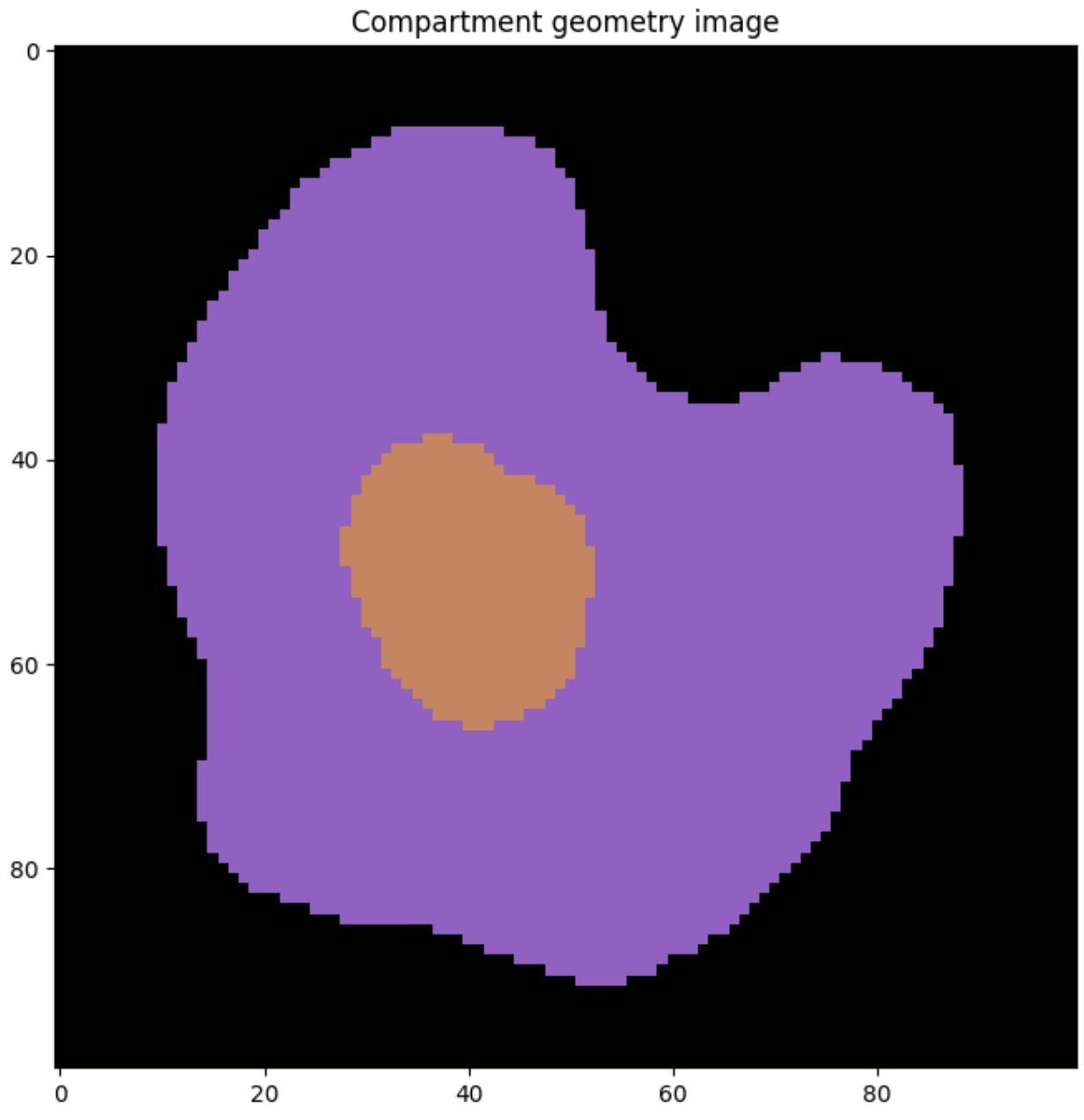
Interactive online version

2.3 sme_contrib.plot

[1]:
```
!pip install -q sme_contrib
import sme
import sme_contrib.plot as smeplot
from matplotlib import pyplot as plt
from IPython.display import HTML
```

2.3.1 Load and simulate example model

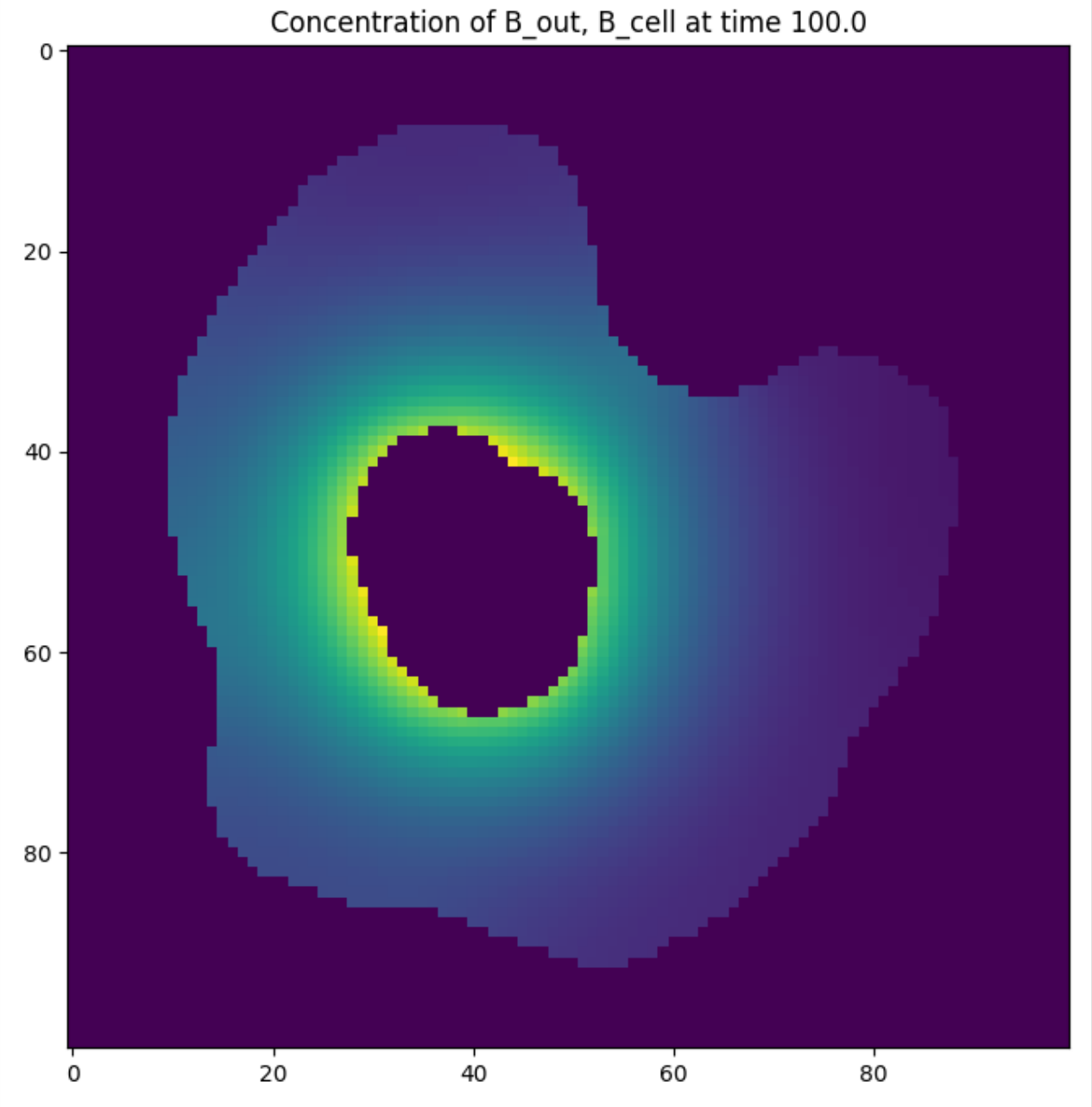
[2]:
```
model = sme.open_example_model()
fig = plt.figure(figsize=(16, 8))
plt.imshow(model.compartment_image[0, :])
plt.title("Compartment geometry image")
plt.show()
results = model.simulate(100, 1)
species = ["B_out", "B_cell"]
```



2.3.2 Plot resulting species concentration

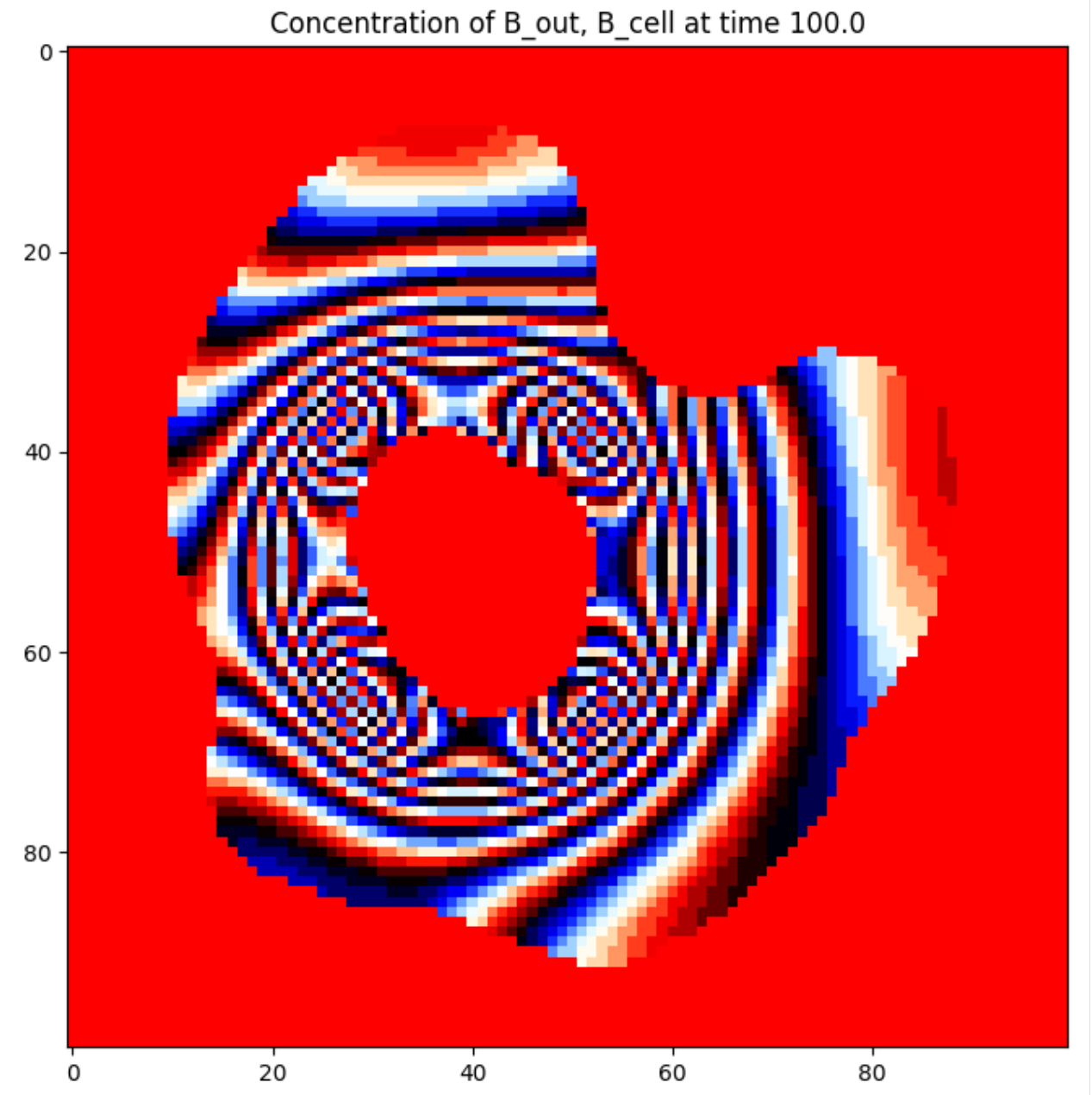
Use default colormap

[3]:
```
fig = plt.figure(figsize=(16, 8))
smeplot.concentration_heatmap(results[-1], species)
plt.show()
```



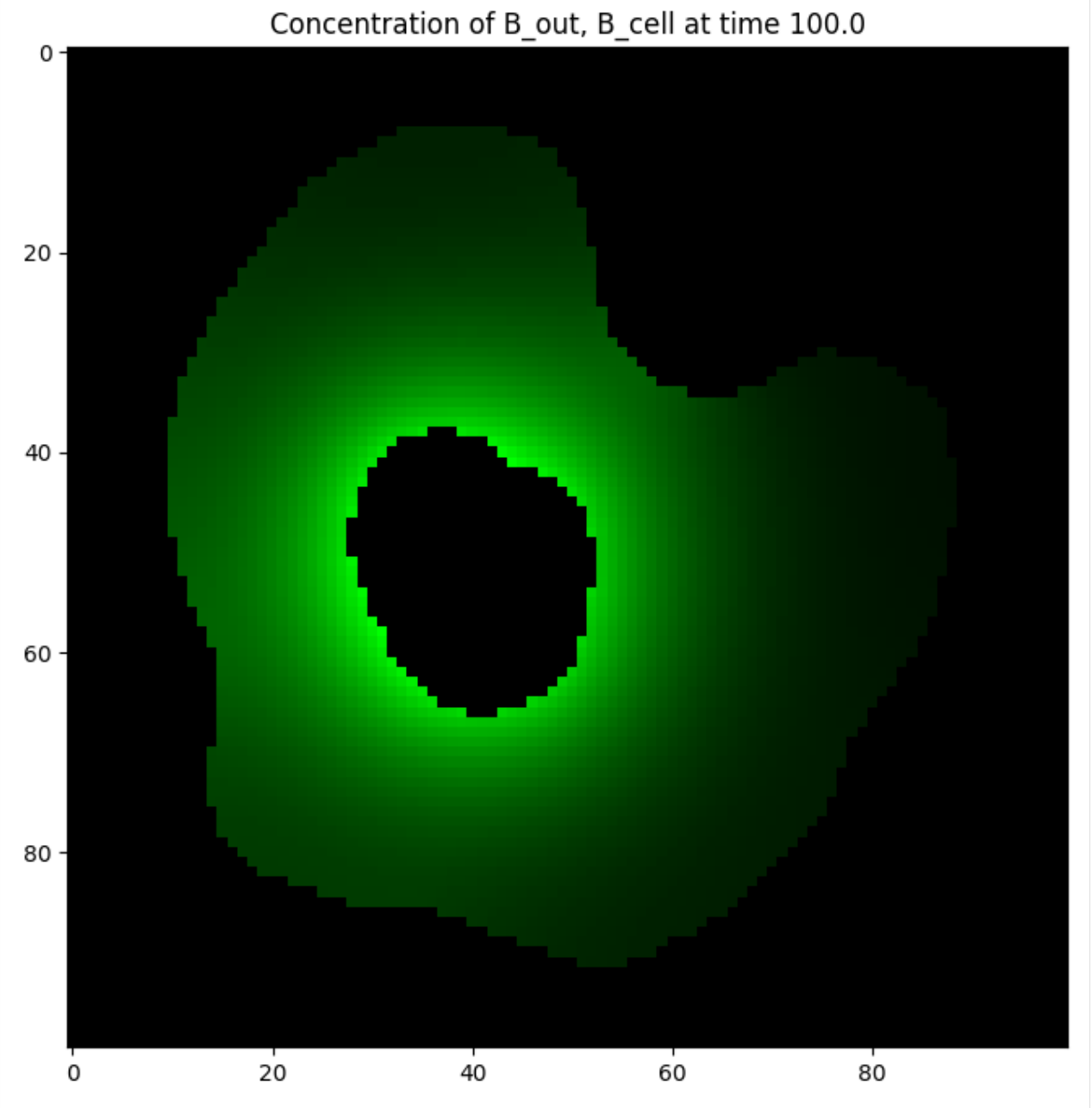
Use a built-in matplotlib colormap

[4]:
```
fig = plt.figure(figsize=(16, 8))
smeplot.concentration_heatmap(results[-1], species, cmap="flag")
plt.show()
```



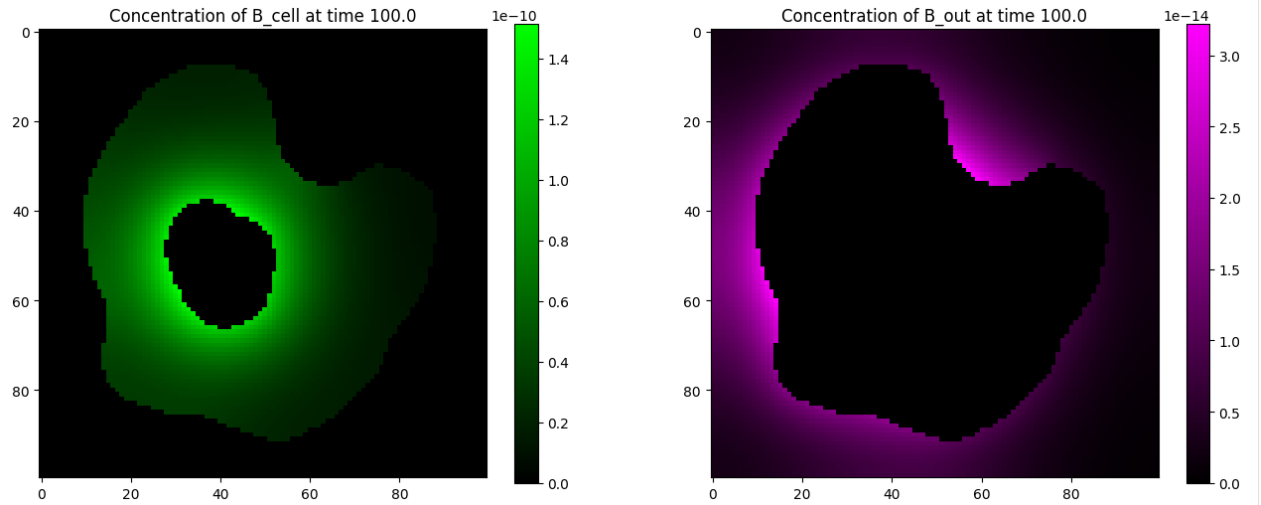
Create your own colormap

[5]:
```
# make a black -> green colormap using sme_contrib.plot.colormap:
cmap = smeplot.colormap("#00ff00")
fig = plt.figure(figsize=(16, 8))
smeplot.concentration_heatmap(results[-1], species, cmap=cmap)
plt.show()
```



Display on existing axes with colorbar

[6]:
```
fig, (ax_l, ax_r) = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))
ax_l, im_l = smeplot.concentration_heatmap(
    results[-1], ["B_cell"], cmap=smeplot.colormap("#00ff00"), ax=ax_l
)
ax_r, im_r = smeplot.concentration_heatmap(
    results[-1], ["B_out"], cmap=smeplot.colormap("#ff00ff"), ax=ax_r
)
fig.colorbar(im_l, ax=ax_l)
fig.colorbar(im_r, ax=ax_r)
plt.show()
```



2.3.3 Plot animation of species concentration

[7]:
```
anim = smeplot.concentration_heatmap_animation(results, ["B_cell"], figsize=(8, 6))
```

Display as html5 video

[8]:
```
HTML(anim.to_html5_video())
```

[8]:
```
<IPython.core.display.HTML object>
```

Display as javascript widget

[9]:
```
HTML(anim.to_jshtml())
```

[9]:
```
<IPython.core.display.HTML object>
```

[ ]:
```

```

Interactive online version

2.4 sme_contrib.optimize

[1]:
```
!pip install -q sme_contrib
import sme
import sme_contrib.optimize as smeopt
import sme_contrib.plot as smeplot
import numpy as np
from matplotlib import pyplot as plt
from IPython.display import HTML
from PIL import Image
```

2.4.1 Gray-Scott model

A simple two-species model with two reaction rate parameters, that forms spatial patterns and eventually reaches a steady state

[2]:
```
def simulated_gray_scott(f, k):
    m = sme.open_example_model("gray-scott")
    m.compartments[0].species[
        "V"
    ].analytic_concentration = "exp(-((x-49.5)^2+(y-49.5)^2))"
    m.parameters["f"].value = f"{f}"
    m.parameters["k"].value = f"{k}"
    m.simulate(5000, 50, return_results=False)
    return m


def gray_scott_anim(f, k):
    gray_scott = simulated_gray_scott(f, k)
    return smeplot.concentration_heatmap_animation(
        gray_scott.simulation_results(), ["V"]
    )
```

[3]:
```
anim = gray_scott_anim(0.04, 0.06)
HTML(anim.to_html5_video())
```

```
2023-10-02 08:10:28,649 - matplotlib.animation - INFO - Animation.save using <class
↪'matplotlib.animation.FFMpegWriter'>
2023-10-02 08:10:28,651 - matplotlib.animation - INFO - MovieWriter._run: running␣
↪command: ffmpeg -f rawvideo -vcodec rawvideo -s 640x480 -pix_fmt rgba -framerate 5.0 -
↪loglevel error -i pipe: -vcodec h264 -pix_fmt yuv420p -y /tmp/tmpy50emvfo/temp.m4v
```

[3]:
```
<IPython.core.display.HTML object>
```

[4]:
```
anim = gray_scott_anim(0.051, 0.061)
HTML(anim.to_html5_video())
```

```
2023-10-02 08:10:35,767 - matplotlib.animation - INFO - Animation.save using <class
↪'matplotlib.animation.FFMpegWriter'>
2023-10-02 08:10:35,768 - matplotlib.animation - INFO - MovieWriter._run: running␣
↪command: ffmpeg -f rawvideo -vcodec rawvideo -s 640x480 -pix_fmt rgba -framerate 5.0 -
↪loglevel error -i pipe: -vcodec h264 -pix_fmt yuv420p -y /tmp/tmpddztyxya/temp.m4v
```

[4]:
```
<IPython.core.display.HTML object>
```

[5]:
```
anim = gray_scott_anim(0.028, 0.062)
HTML(anim.to_html5_video())
```

```
2023-10-02 08:10:43,253 - matplotlib.animation - INFO - Animation.save using <class
↪'matplotlib.animation.FFMpegWriter'>
2023-10-02 08:10:43,254 - matplotlib.animation - INFO - MovieWriter._run: running␣
↪command: ffmpeg -f rawvideo -vcodec rawvideo -s 640x480 -pix_fmt rgba -framerate 5.0 -
↪loglevel error -i pipe: -vcodec h264 -pix_fmt yuv420p -y /tmp/tmpu9su3rzh/temp.m4v
```

[5]:
```
<IPython.core.display.HTML object>
```

2.4.2 Try to fit to the target steady state

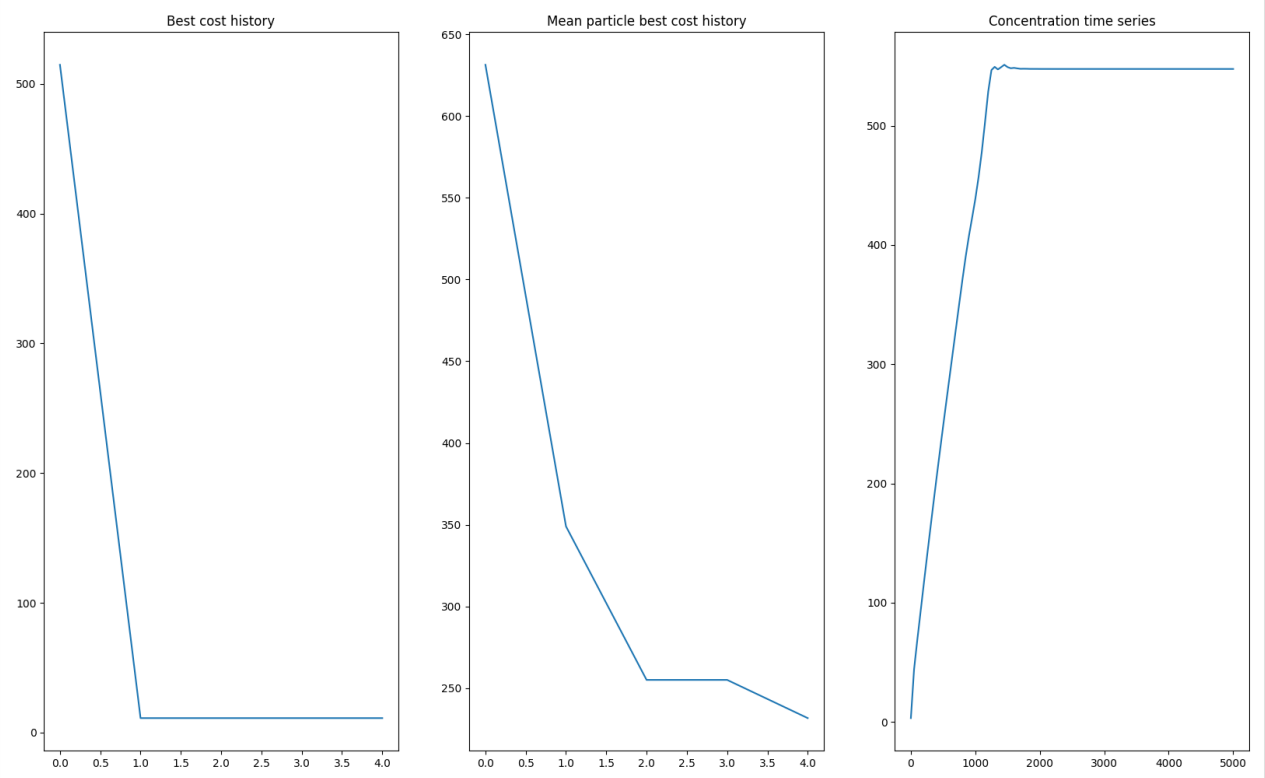
Increasing the number of particles and the number of iterations will improve the fit, but take longer to run.

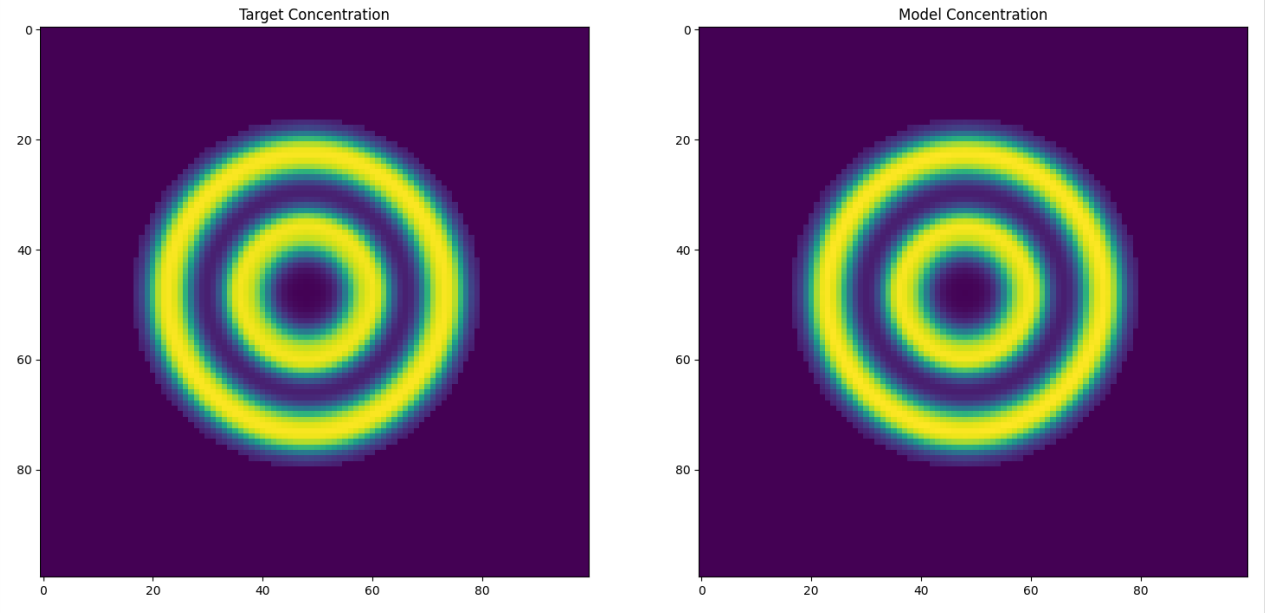
[6]:
```
def create_target_image(f, k):
    gray_scott = simulated_gray_scott(f, k)
    conc = gray_scott.simulation_results()[-1].species_concentration["V"][0, :]
    conc = 255 * conc / np.max(conc)
    Image.fromarray(conc.astype("uint8")).save("tmp.png")
    gray_scott.export_sbml_file("tmp.xml")
```

[7]:
```
def apply_params(model, params):
    model.parameters["f"].value = f"{params[0]}"
    model.parameters["k"].value = f"{params[1]}"
```

[8]:
```
create_target_image(0.04, 0.06)
ss = smeopt.SteadyState(
    "tmp.xml",
    "tmp.png",
    ["V"],
    apply_params,
    [0.01, 0.05],
    [0.06, 0.07],
    5000,
    2000,
    90,
)
ss.find(5, 5)
ss.plot_all()
```

```
2023-10-02 08:10:49,522 - pyswarms.single.global_best - INFO - Optimize for 5 iters with
↪{'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best: 100%||5/5, best_cost=11.2
2023-10-02 08:11:03,766 - pyswarms.single.global_best - INFO - Optimization finished |␣
↪best cost: 11.17878621420758, best pos: [0.03638574 0.05947195]
```





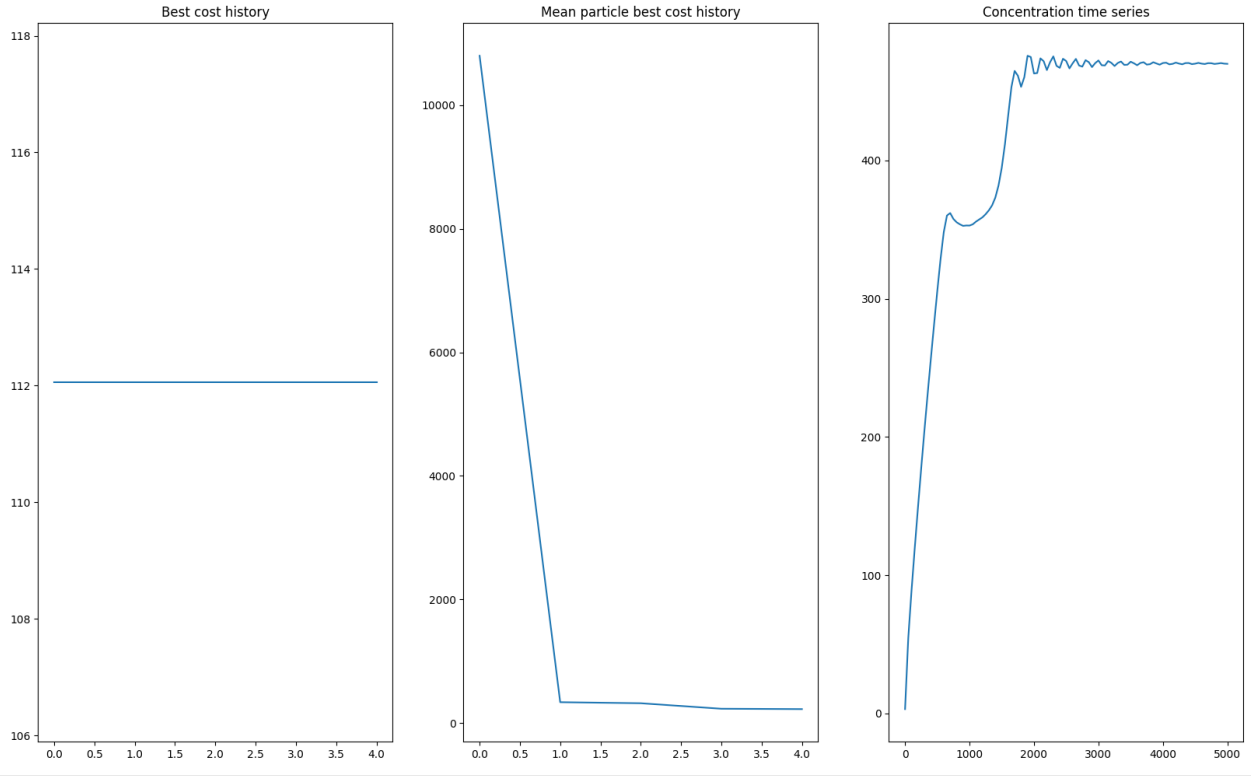
[9]:
```
create_target_image(0.051, 0.061)
ss = smeopt.SteadyState(
    "tmp.xml",
    "tmp.png",
    ["V"],
    apply_params,
    [0.01, 0.05],
    [0.06, 0.07],
```

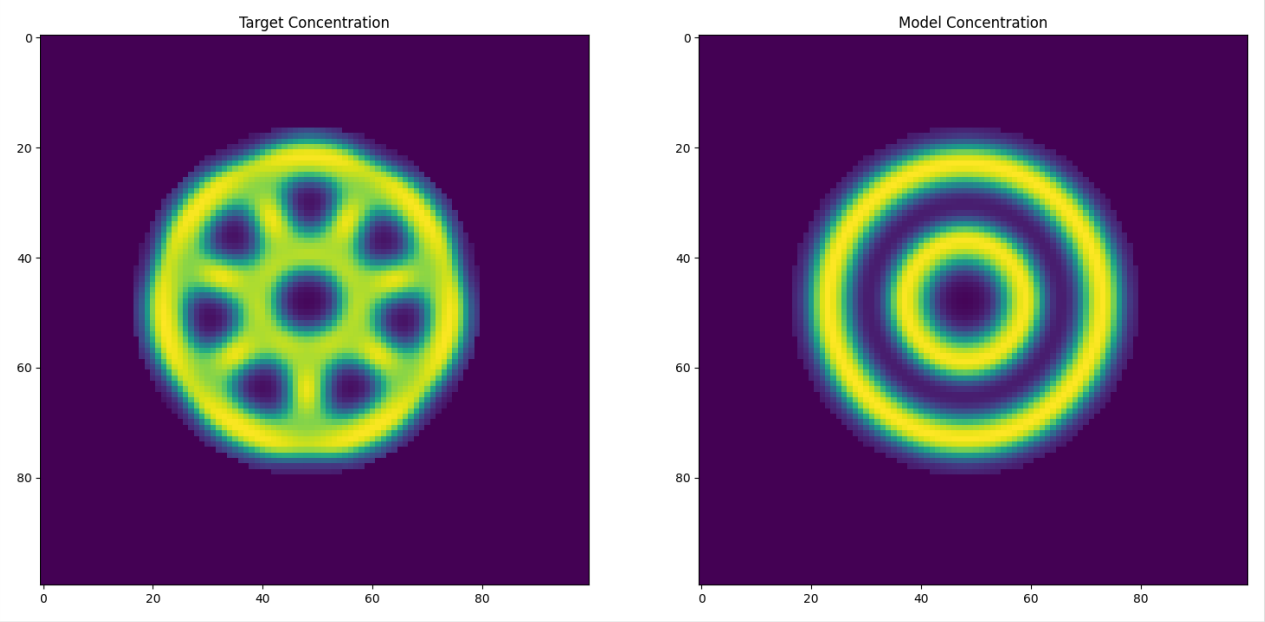
(continues on next page)

(continued from previous page)

```
    5000,
    2000,
    90,
)
ss.find(5, 5)
ss.plot_all()
```

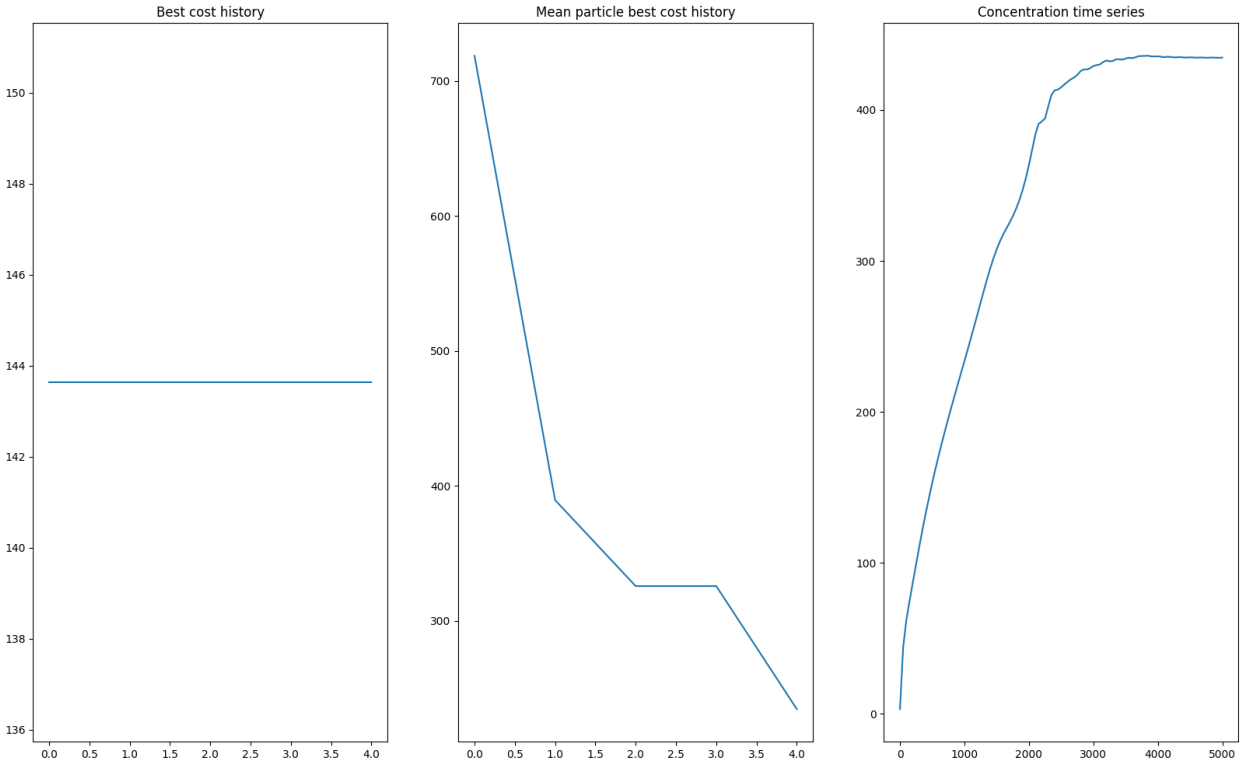
```
2023-10-02 08:11:06,700 - pyswarms.single.global_best - INFO - Optimize for 5 iters with
↪{'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best: 100%||5/5, best_cost=112
2023-10-02 08:11:21,053 - pyswarms.single.global_best - INFO - Optimization finished |␣
↪best cost: 112.05401544882662, best pos: [0.02611539 0.05628643]
```





[10]:
```
create_target_image(0.028, 0.062)
ss = smeopt.SteadyState(
    "tmp.xml",
    "tmp.png",
    ["V"],
    apply_params,
    [0.01, 0.05],
    [0.06, 0.07],
    5000,
    2000,
    90,
)
ss.find(5, 5)
ss.plot_all()
```

```
2023-10-02 08:11:23,926 - pyswarms.single.global_best - INFO - Optimize for 5 iters with
↪{'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best: 100%||5/5, best_cost=144
2023-10-02 08:11:38,024 - pyswarms.single.global_best - INFO - Optimization finished |␣
↪best cost: 143.63176259811402, best pos: [0.03104711 0.06042086]
```


Best cost history
Mean particle best cost history
Concentration time series
Target Concentration
Model Concentration

[ ]:

CHAPTER

THREE

SOURCE CODE

The source code is available from GitHub. and released under a permissive MIT license.

PYTHON MODULE INDEX

S

INDEX