
nufeb_{tools}*Documentation*

Release unknown

Jsakkos

Jul 21, 2022

NOTES

1	nufeb_tools	1
1.1	Description	1
1.2	Getting Started	1
2	License	3
3	Atom Generation	5
3.1	Generate Atoms	5
3.2	Generate Atoms (development version)	5
4	NUFEB Simulation Analysis	9
4.1	Get Simulation Data	9
4.2	Spatial Analysis	11
4.3	Plotting	13
4.3.1	Average Nutrient Concentration	13
4.3.2	Single Cell Growth	14
4.3.3	Single Cell Growth Rate	15
4.3.4	Overall Cell Growth	17
4.3.5	Whole Colony Plotting	18
5	Notebooks	21
	Index	23

NUFEB_TOOLS

1.1 Description

Python-based tools and utilities for NUFEB simulations

1.2 Getting Started

Install via pip:

```
pip install nufeb-tools
```

Generate NUFEB simulations from the CLI:

```
nufeb-seed
```

Remove old runs:

```
nufeb-clean
```

Get data from a simulation for analysis

```
from nufeb_tools import utils
x = utils.get_data(test=True)
```

Plot the overall growth

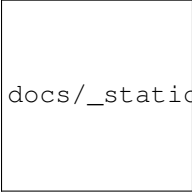
```
from nufeb_tools import plot
import matplotlib.pyplot as plt
f, ax = plt.subplots()
plot.overall_growth(x.biomass, ax=ax)
```



docs/_static/images/total_biomass_vs_time.png

Plot colonies based on initial seed cells

```
from nufeb_tools import utils, plot
import matplotlib.pyplot as plt
x = utils.get_data(directory=r'E:\sucrose\runs\Run_50_50_1.00e+00_1_2022-01-11_48525')
f, ax = plt.subplots()
plot.colony(x, 25900, colors, ax=ax)
plt.show()
```



docs/_static/images/colonies.png

LICENSE

The MIT License (MIT)

Copyright (c) 2021 Jsakkos

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

ATOM GENERATION

3.1 Generate Atoms

`nufeb_tools.generate_atom.clean()`

Remove old NUFEB runs

`nufeb_tools.generate_atom.main(args)`

Wrapper function to generate new NUFEB simulation conditions

Parameters `args` (`List[str]`) – command line parameters as list of strings

`nufeb_tools.generate_atom.parse_args(args)`

Parse command line parameters

Parameters `args` (`List[str]`) – command line parameters as list of strings (for example `["--help"]`).

Returns command line parameters namespace

Return type `argparse.Namespace`

`nufeb_tools.generate_atom.run()`

Calls `main()` passing the CLI arguments extracted from `sys.argv`

This function can be used as entry point to create console scripts with `setuptools`.

`nufeb_tools.generate_atom.setup_logging(loglevel)`

Setup basic logging

Parameters `loglevel` (`int`) – minimum loglevel for emitting messages

3.2 Generate Atoms (development version)

This is a script to seed NUFEB simulations

```
class nufeb_tools.generate_atom_dev.Cell (Species, Group, idx, args, Info={'cyano': {'Decay': 0, 'Density': 370, 'Diameter': 1e-06, 'GrowthRate': 1.67e-05, 'Inertia': {'ixx': 0, 'ixy': 0, 'ixz': 0, 'iyy': 0, 'iyz': 0, 'izz': 9.2e-23}, 'K_s': {'co2': 0.000138, 'light': 0.00035, 'o2': 0.0002, 'suc': 0.01}, 'Maintenance': 0, 'Yield': 0.55, 'max_length': 5e-06, 'min_length': 1e-06}, 'ecw': {'Decay': 2e-05, 'Density': 236, 'Diameter': 7.3e-07, 'GrowthRate': 0.00027, 'Inertia': {'ixx': 0, 'ixy': 0, 'ixz': 0, 'iyy': 0, 'iyz': 0, 'izz': 9.2e-23}, 'K_s': {'co2': 0.05, 'light': 0, 'o2': 0.001, 'suc': 3.6}, 'Maintenance': 9.5e-07, 'Yield': 0.43, 'max_length': 2.72e-06, 'min_length': 1.94e-06}})
```

Bacteria object class

```
Atom ()
```

Function to return atom (cell) positions to render atom.in file

```
Bacillus ()
```

Function to return rod shape (bacillus) parameters to render atom.in file

```
Check ()
```

Return cell position, orientation, and size

```
Report ()
```

Return cell position, orientation, and size

```
rotate (z_dim=False)
```

Randomly generate cell orientation displacements based on input angle

```
class nufeb_tools.generate_atom_dev.Culture (args)
```

Create a collection of cells with defined positions, lengths, and orientations

```
Check ()
```

Check initial positions of each cell and move one of them if there is a collision

```
class nufeb_tools.generate_atom_dev.Nutrient (c, d, mw, state, bc)
```

Nutrient class to define the chemicals present in the simulation volume, their concentrations, and their properties

```
nufeb_tools.generate_atom_dev.main (args)
```

Wrapper function to generate new NUFEB simulation conditions

```
    Parameters args (List[str]) – command line parameters as list of strings
```

```
nufeb_tools.generate_atom_dev.parse_args (args)
```

Parse command line parameters

```
    Parameters args (List[str]) – command line parameters as list of strings (for example ["--help"]).
```

```
    Returns command line parameters namespace
```

```
    Return type argparse.Namespace
```

```
nufeb_tools.generate_atom_dev.run ()
```

Calls `main()` passing the CLI arguments extracted from `sys.argv`

This function can be used as entry point to create console scripts with `setuptools`.

```
nufeb_tools.generate_atom_dev.setup_logging (loglevel)
```

Setup basic logging

Parameters `loglevel` (*int*) – minimum loglevel for emitting messages

NUFEB SIMULATION ANALYSIS

4.1 Get Simulation Data

class `nufeb_tools.utils.get_data` (*directory=None, id=None, test=None, timestep=10*)

Bases: `object`

Collect results for analysis.

NUFEB simulation data class to collect results for analysis

test

Set *test = True* to get example data from the Github repository

Type `bool`

directory

Path to the directory containing NUFEB simulation data.

Type `str`

timestep

Length of simulation timestep in seconds

Type `int`

SucRatio

Relative cyanobacterial sucrose secretion level, 0-100

Type `int`

timepoints

List of timepoints in the simulation

Type `List(str)`

dims

Size of the simulation boundaries in micrometers

Type `List(str)`

numsteps

Number of timepoints

Type `int`

biomass

Pandas Dataframe containing the biomass vs time data from biomass.csv

Type `pandas.DataFrame`

ntypes

Pandas Dataframe containing the cell number vs time data from ntypes.csv

Type `pandas.DataFrame`

avg_con

Pandas Dataframe containing the average nutrient concentrations vs time data from avg_concentration.csv

Type `pandas.DataFrame`

positions

Pandas Dataframe containing the single cell biomass over time of all cell ids present at the timepoint

Type `pandas.DataFrame`

get_local_data()

Collect NUFEB simulation data from a local directory.

convert_units_avg_con()

Convert the object attribute avg_con, which contains the average nutrient concentration, units to hours and mM.

convert_units_biomass()

Convert the object attribute biomass units to hours and femtograms.

calc_biomass()**collect_positions(h5)**

Extract the x, y, z position of each cell during the simulation.

Parameters **timepoint** (*int*) – The simulation timestep to get the position data from.

Returns Dataframe containing Timestep, ID, type, radius, x, y, z columns

Return type `pandas.DataFrame`

get_neighbor_distance(id, timepoint)

Get the nearest neighbor cell distances

Parameters

- **id** (*int*) – The ID of the reference cell
- **timepoint** (*int*) – The timepoint to check the neighbor distances from

Returns Dataframe containing ID, type, Distance

Return type `pandas.DataFrame`

get_neighbors(timestep)

Get the nearest neighbor cell distances

Parameters **timestep** (*int*) – The timepoint to check the neighbor distances from

Returns Pandas dataframe containing pairwise neighbor distances

Return type `pd.DataFrame`

get_mothers__old()

Assign mother cells based on initial cells in the simulation.

Returns Dataframe containing ID, type, position, radius, and mother_cell

Return type `pandas.DataFrame`

get_mothers()

Assign mother cells based on initial cells in the simulation.

Returns Dataframe containing Timestep, ID, type, position, radius, biomass, total biomass, and mother_cell

Return type `pandas.DataFrame`

count_colony_area (*timestep*)

Count the 2d area in pixel dimensions of each colony at a given timestep.

Parameters **timestep** (*int*) – Timestep to count

get_colony_areas ()

Count colony areas for all timesteps

get_nutrient_grid (*h5*)

get_local_con (*timestep, cellID*)

Get the local nutrient concentration of a cell

Parameters

- **timestep** (*int*) – The timestep at which to check the concentration
- **cellID** (*int*) – The cell identification number

Returns The concentration of the specified nutrient within the cell's grid

Return type Nutrient Concentration (*float*)

get_fitness (*timestep, cellID*)

Get the fitness of an individual cell based on the relative Monod growth rate at a given timestep

Parameters

- **timestep** (*int*) – The timestep at which to check the concentration
- **cellID** (*int*) – The cell identification number

Returns The Monod growth rate (1/s)

Return type *float*

collect_fitness ()

4.2 Spatial Analysis

`nufeb_tools.spatial.fitness_metrics` (*obj*)

Function to calculate colony-level fitness metrics.

Mother cell: Cell ID which seeded the colony

Type: Cell type - cyanobacteria are type 1 and E. coli are type 2

Voronoi area: Area of species-specific Voronoi Tessellation at the beginning of the simulation

IPTG: Sucrose induction level

total biomass: Biomass of each colony at the end of the simulation (fg)

Nearest 1: Distance to nearest cyanobacteria colony

Nearest 2: Distance to nearest E. coli colony

Nearest Neighbor: Distance to nearest colony

IC1: Average distance to nearest cyanobacteria colony

IC2: Average distance to nearest E. coli colony

IC: Average distance to nearest colony

Relative Neighbor Dist 1: Distance to nearest cyanobacteria colony divided by IC1

Relative Neighbor Dist 2: Distance to nearest E. coli colony divided by IC2

Relative Neighbor Dist: Distance to nearest colony divided by IC

Z1: Relative neighbor distance 1 divided by $\sqrt{D_{\text{sucrose}}/\mu_{\text{cyano}}}$

Z2: Relative neighbor distance 2 divided by $\sqrt{D_{\text{sucrose}}/\mu_{\text{ecw}}}$

Z1_2: Relative neighbor distance 1 divided by $\sqrt{D_{\text{sucrose}}/\mu_{\text{ecw}}}$

Z2_1: Relative neighbor distance 2 divided by $\sqrt{D_{\text{sucrose}}/\mu_{\text{cyano}}}$

LogNearest 1: $\log(\text{Nearest 1})$

LogNearest 2: $\log(\text{Nearest 2})$

LogNearest: $\log(\text{Nearest Neighbor})$

Inv1: Inverse sum of neighbor distance 1

Inv2: Inverse sum of neighbor distance 2

Log Inv1: Log squared inverse sum of neighbor distance 1

Log Inv2: Log squared inverse sum of neighbor distance 2

Colony Area: 2D area of colony at the end of the simulation

Parameters `obj` (`nufeb_tools.utils.get_data`) – Data object collected with `nufeb_tools.utils.get_data`

Returns Dataframe containing colony number (mother cell ID), cell type, total biomass, colony area, Voronoi area, nearest neighbor, mean neighbor distance, etc.

Return type `pandas.DataFrame`

```
from nufeb_tools import utils, spatial
x = utils.get_data(directory = None, test=True)
metrics = spatial(x)
metrics.head()
```


Table 1: Metrics

			moth	prev	col	mo	to	Nea	Nea	Nea	C	I	C	Rel	Rel	Rel	Z	Z	Z	Log	Log	Log	Inv	Inv	Log	Log	Colony
						Area	Rat	biom	est	est	est	Neigh	bor	a-	a-	a-	1	2	1	ear	ear	ear	1	2	Inv	Inv	Area
						tio	bio	mass	2	bor				Neigh	Neigh	Neigh				1	2	est	est	est			
0	17	1	1.37	10	66	47	40	25	10	89	10	50	10	50	10	50	10	50	10	50	10	50	10	50	10	50	10

4.3 Plotting

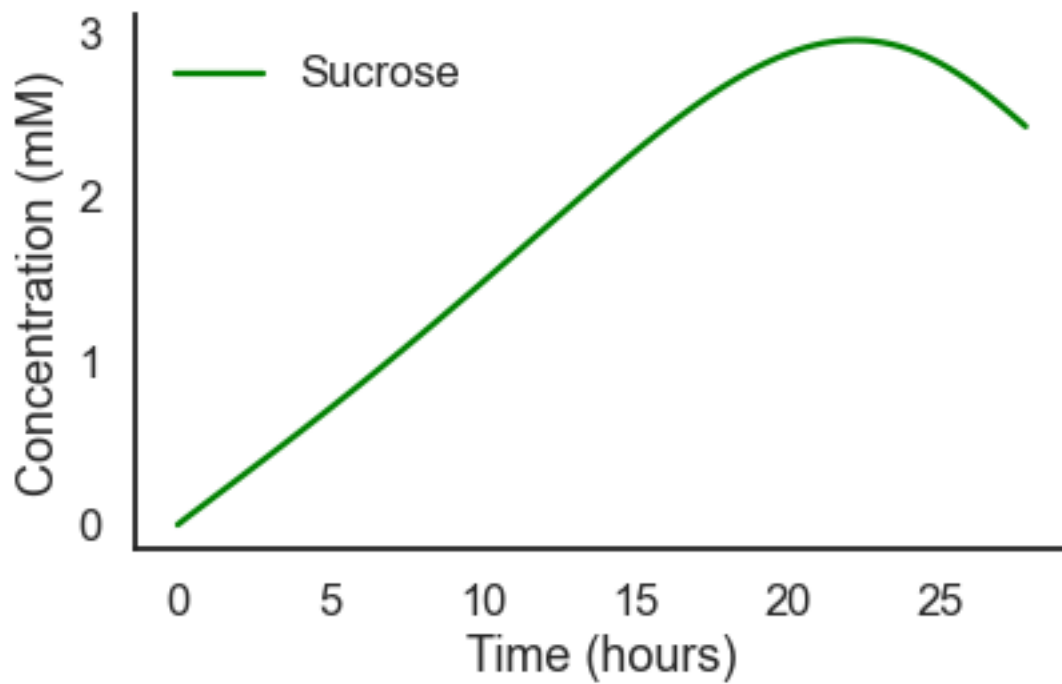
4.3.1 Average Nutrient Concentration

`nufeb_tools.plot.average_nutrients(df, nutrient, ax=None, legend=None, **kwargs)`

This is a function to plot the nutrient concentration over time

Parameters

- **df** (`pandas.DataFrame`) – Pandas Dataframe containing nutrient data
- **nutrient** (`str`) – Name of the nutrient to plot, e.g., 'Sucrose'
- **ax** – Axis on which to make the plot
- **legend** (`bool`) – Include legend in the plot
- ****kwargs** – Additional arguments to pass to `plt.plot`



```
from nufeb_tools import utils, plot
import matplotlib.pyplot as plt
import seaborn as sns
x = utils.get_data(directory = None, test=True)
f, ax = plt.subplots()
sns.set_context('talk')
sns.set_style('white')
plot.average_nutrients(x.avg_con, 'Sucrose', color='Green', legend=True)
```

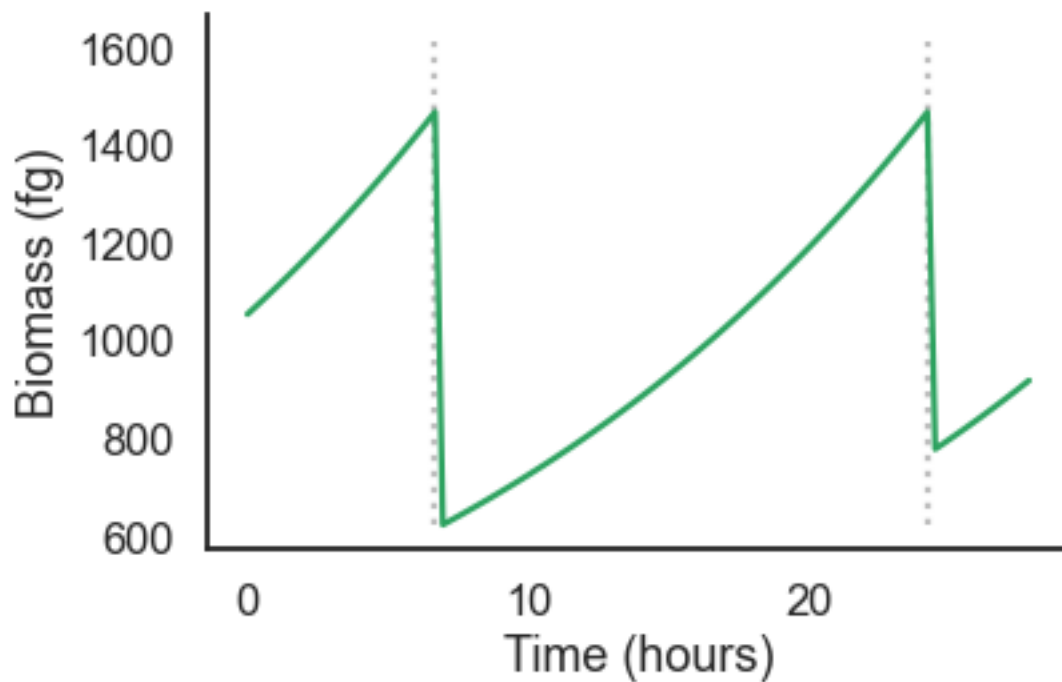
4.3.2 Single Cell Growth

`nufeb_tools.plot.biomass_time(df, id=None, ax=None, legend=None, **kwargs)`

This is a function to plot the cell biomass over time

Parameters

- **df** (`pandas.DataFrame`) – Pandas Dataframe containing biomass data
- **ax** – Axis on which to make the plot
- **legend** (`bool`) – Include legend in the plot
- ****kwargs** – Additional arguments to pass to `plt.plot`



```
from nufeb_tools import utils, plot
import matplotlib.pyplot as plt
import seaborn as sns
f, ax = plt.subplots()
sns.set_context('talk')
sns.set_style('white')
x = utils.get_data(directory = None, test=True)
plot.biomass_time(x.positions)
f.tight_layout()
```

4.3.3 Single Cell Growth Rate

`nufeb_tools.plot.growth_rate_div(df, **kwargs)`

Plot a heatmap of the single cell growth rates relative to each division

Parameters

- **df** (*pandas.DataFrame*) – Pandas Dataframe containing biomass data
- ****kwargs** – Additional arguments to pass to `plt.plot`

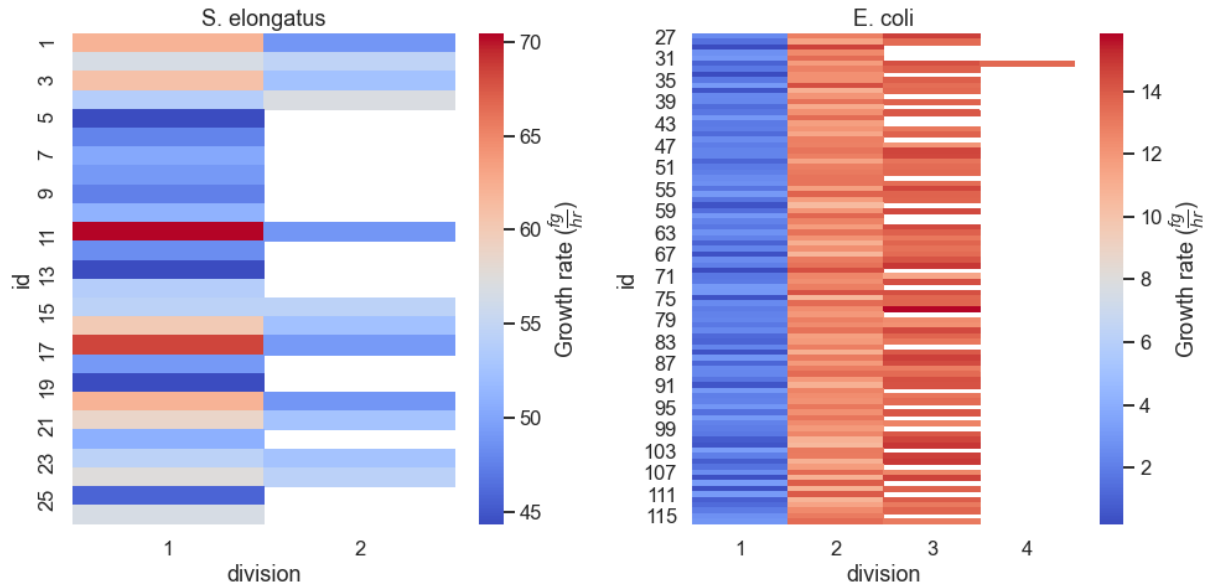
`nufeb_tools.plot.growth_rate_time(df, period=3)`

Plot a heatmap of the single cell growth rates over time

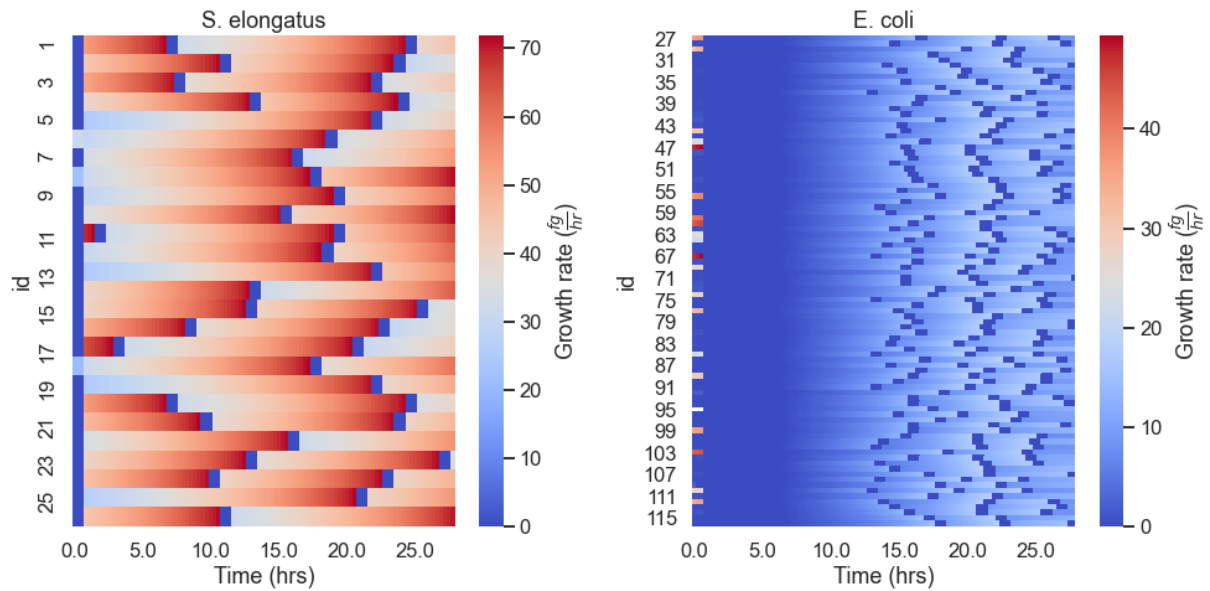
Parameters

- **df** (*pandas.DataFrame*) – Pandas Dataframe containing biomass data
- **period** (*int*) – Number of timesteps to average growth rate calculation over
- ****kwargs** – Additional arguments to pass to `plt.plot`

Returns `matplotlib.figure.Figure`



```
from nufeb_tools import utils, plot
import matplotlib.pyplot as plt
x = utils.get_data(directory = None, test=True)
plot.growth_rate_div(x.positions)
```



```
from nufeb_tools import utils, plot
import matplotlib.pyplot as plt
x = utils.get_data(directory = None, test=True)
plot.growth_rate_time(x.positions)
```

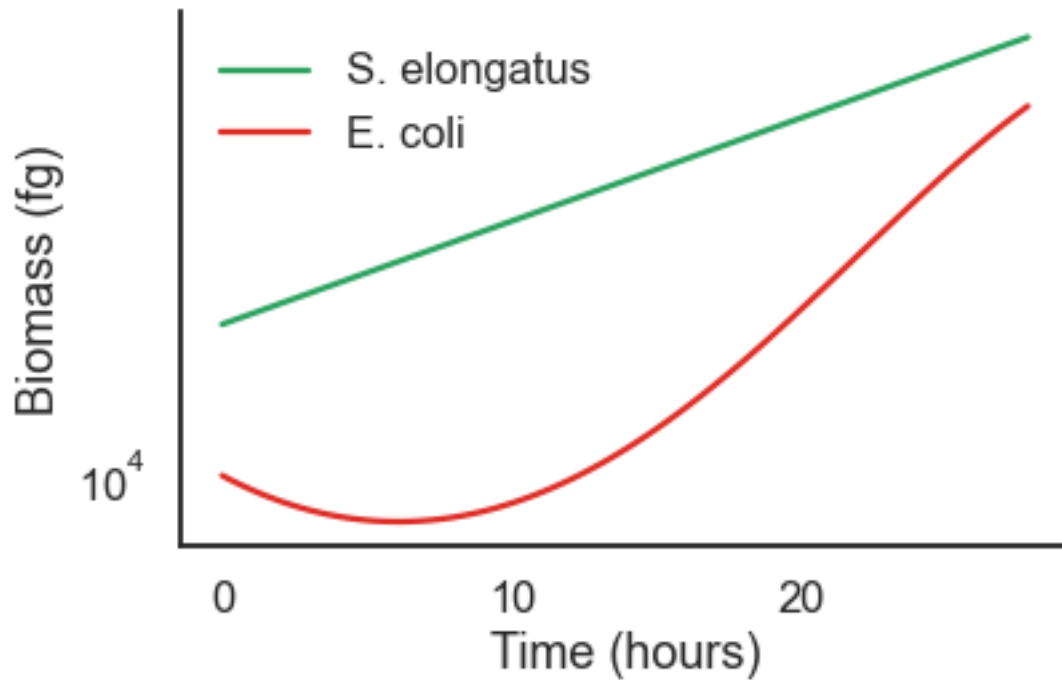
4.3.4 Overall Cell Growth

`nufeb_tools.plot.overall_growth(df, ax=None, **kwargs)`

This is a function to generate growth curve plots

Parameters

- **df** (*pandas.DataFrame*) – Pandas Dataframe containing biomass data over time
- **ax** (*plt.ax*) – Axis to plot data on
- ****kwargs** –



```
from nufeb_tools import utils, plot
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
sns.set_context('talk')
f, ax = plt.subplots()
x = utils.get_data(directory = None, test=True)
plot.overall_growth(x.biomass, ax=ax)
f.tight_layout()
```

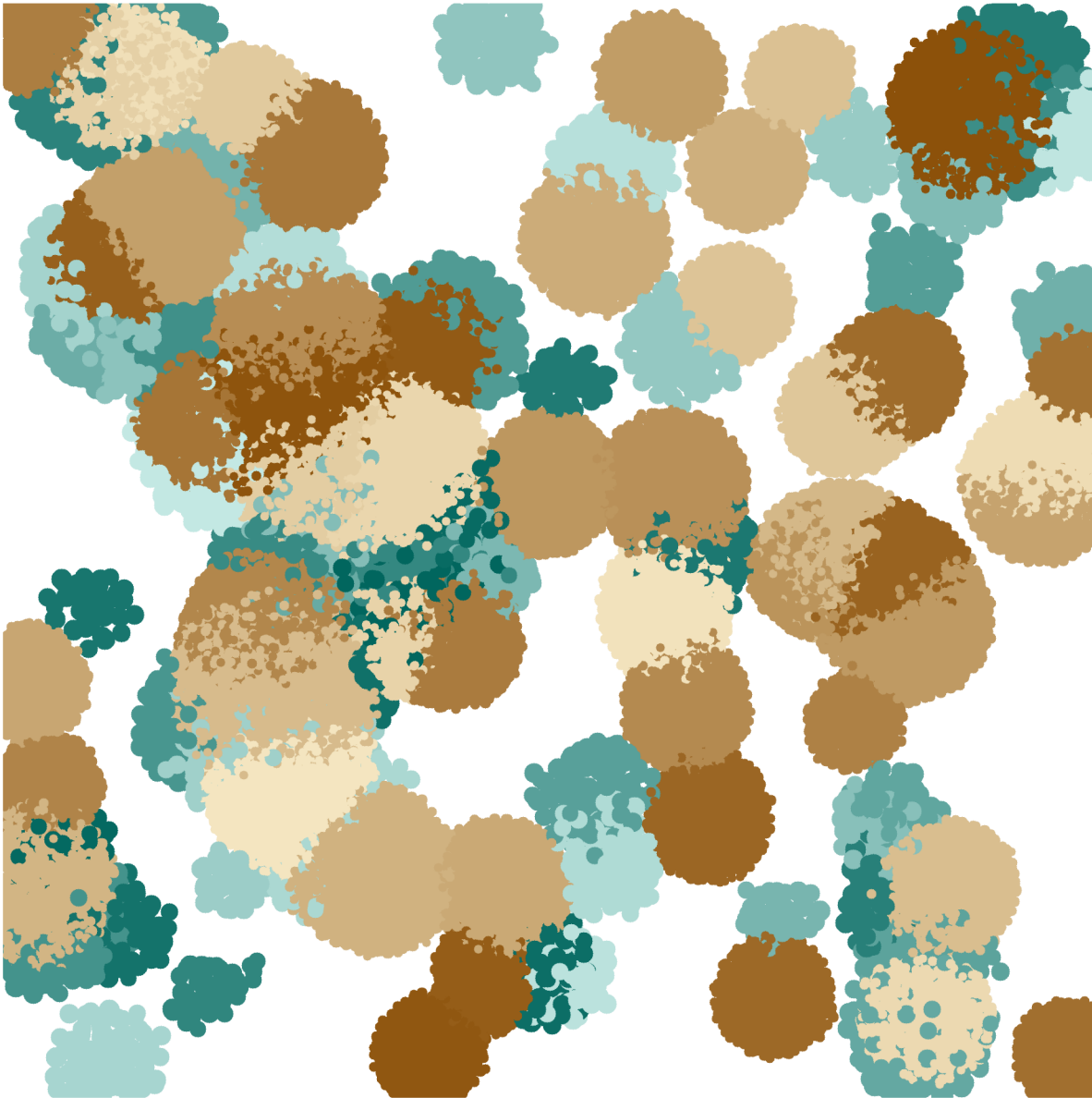
4.3.5 Whole Colony Plotting

```
nufeb_tools.plot.colony(obj, time, colors=None, colony=None, ax=None, by=None, img=array([],  
dtype=float64), fitness=None, overlay=None, **kwargs)
```

Plot bacterial colonies at a specific timepoint

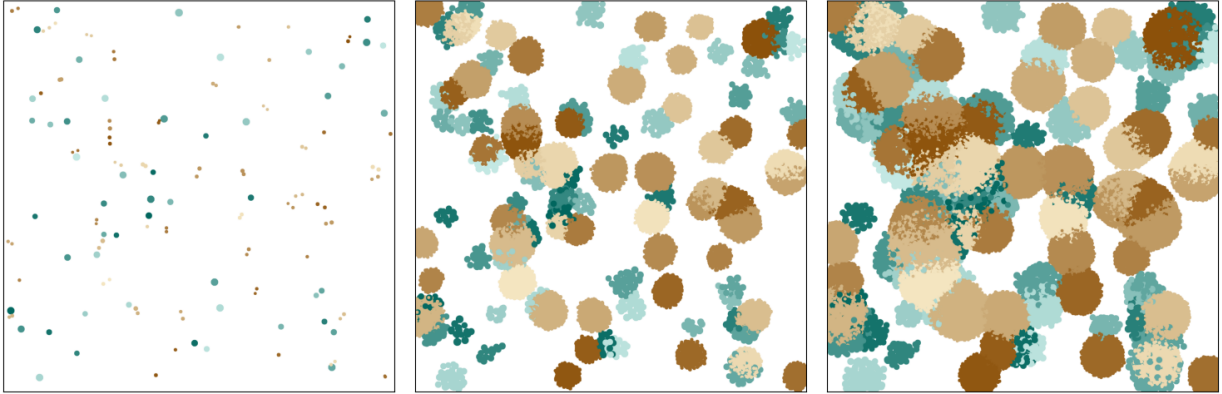
Parameters

- **obj** (*nufeb_tools.utils.get_data*) – Object containing cell locations
- **time** (*int*) – Simulation timestep to plot
- **colors** (*dict, optional*) – Dictionary of colors to plot each colony. Defaults to random.
- **colony** (*int, optional*) – Plot a specific colony. Defaults to None.
- **ax** (*matplotlib.pyplot.axes, optional*) – Axis to plot on. Defaults to None.
- **by** (*str, optional*) – Plot by species. Defaults to None.
- **img** (*np.array, optional*) – Image array to overlay colonies onto.
- **fitness** (*pandas.DataFrame, optional*) – Takes a dataframe containing spatial metrics data as an input or if bool, will calculate the metrics internally. Defaults to None.
- **overlay** (*bool, optional*) – If True, plot a specific colony on top of the others.



```
from nufeb_tools import utils, plot
import matplotlib.pyplot as plt
x = utils.get_data(directory=r'E:\sucrose\runs\Run_50_50_1.00e+00_1_2022-01-11_48525')
f, ax = plt.subplots()
plot.colony(x, 35000, colors, ax=ax)
plt.show()
```

Plot colonies over time by species:



```
from nufeb_tools import utils, plot
import matplotlib.pyplot as plt
x = utils.get_data(directory=r'E:\sucrose\runs\Run_50_50_1.00e+00_1_2022-01-11_48525')
f, axes = plt.subplots(ncols=3, figsize=(15, 5))
for ax, time in zip(axes, [100, 20000, 25900]):
    plot.plot_colony(x, time, by='Species', ax=ax)
plt.show()
```


NOTEBOOKS

Examples go here

A

`Atom()` (*nufeb_tools.generate_atom_dev.Cell method*), 6
`average_nutrients()` (in module *nufeb_tools.plot*), 13
`avg_con` (*nufeb_tools.utils.get_data attribute*), 10

B

`Bacillus()` (*nufeb_tools.generate_atom_dev.Cell method*), 6
`biomass` (*nufeb_tools.utils.get_data attribute*), 9
`biomass_time()` (in module *nufeb_tools.plot*), 14

C

`calc_biomass()` (*nufeb_tools.utils.get_data method*), 10
`Cell` (class in *nufeb_tools.generate_atom_dev*), 5
`Check()` (*nufeb_tools.generate_atom_dev.Cell method*), 6
`Check()` (*nufeb_tools.generate_atom_dev.Culture method*), 6
`clean()` (in module *nufeb_tools.generate_atom*), 5
`collect_fitness()` (*nufeb_tools.utils.get_data method*), 11
`collect_positions()` (*nufeb_tools.utils.get_data method*), 10
`colony()` (in module *nufeb_tools.plot*), 18
`convert_units_avg_con()` (*nufeb_tools.utils.get_data method*), 10
`convert_units_biomass()` (*nufeb_tools.utils.get_data method*), 10
`count_colony_area()` (*nufeb_tools.utils.get_data method*), 11
`Culture` (class in *nufeb_tools.generate_atom_dev*), 6

D

`dims` (*nufeb_tools.utils.get_data attribute*), 9
`directory` (*nufeb_tools.utils.get_data attribute*), 9

F

`fitness_metrics()` (in module *nufeb_tools.spatial*), 11

G

`get_colony_areas()` (*nufeb_tools.utils.get_data method*), 11
`get_data` (class in *nufeb_tools.utils*), 9
`get_fitness()` (*nufeb_tools.utils.get_data method*), 11
`get_local_con()` (*nufeb_tools.utils.get_data method*), 11
`get_local_data()` (*nufeb_tools.utils.get_data method*), 10
`get_mothers()` (*nufeb_tools.utils.get_data method*), 10
`get_mothers__old()` (*nufeb_tools.utils.get_data method*), 10
`get_neighbor_distance()` (*nufeb_tools.utils.get_data method*), 10
`get_neighbors()` (*nufeb_tools.utils.get_data method*), 10
`get_nutrient_grid()` (*nufeb_tools.utils.get_data method*), 11
`growth_rate_div()` (in module *nufeb_tools.plot*), 15
`growth_rate_time()` (in module *nufeb_tools.plot*), 15

M

`main()` (in module *nufeb_tools.generate_atom*), 5
`main()` (in module *nufeb_tools.generate_atom_dev*), 6
module
 nufeb_tools.generate_atom, 5
 nufeb_tools.generate_atom_dev, 5

N

`ntypes` (*nufeb_tools.utils.get_data attribute*), 9
nufeb_tools.generate_atom
 module, 5
nufeb_tools.generate_atom_dev
 module, 5
`numsteps` (*nufeb_tools.utils.get_data attribute*), 9
`Nutrient` (class in *nufeb_tools.generate_atom_dev*), 6

O

`overall_growth()` (in module *nufeb_tools.plot*), 17

P

`parse_args()` (in module *nufeb_tools.generate_atom*), 5

`parse_args()` (in module *nufeb_tools.generate_atom_dev*), 6

`positions` (*nufeb_tools.utils.get_data attribute*), 10

R

`Report()` (*nufeb_tools.generate_atom_dev.Cell method*), 6

`rotate()` (*nufeb_tools.generate_atom_dev.Cell method*), 6

`run()` (in module *nufeb_tools.generate_atom*), 5

`run()` (in module *nufeb_tools.generate_atom_dev*), 6

S

`setup_logging()` (in module *nufeb_tools.generate_atom*), 5

`setup_logging()` (in module *nufeb_tools.generate_atom_dev*), 6

`SucRatio` (*nufeb_tools.utils.get_data attribute*), 9

T

`test` (*nufeb_tools.utils.get_data attribute*), 9

`timepoints` (*nufeb_tools.utils.get_data attribute*), 9

`timestep` (*nufeb_tools.utils.get_data attribute*), 9