BioMASS

Release 0.13.0

Hiroaki Imoto

Apr 12, 2024

CONTENTS:

Source code: https://github.com/biomass-dev/biomass

Arakane, K., Imoto, H., Ormersbach, F. & Okada, M. Extending BioMASS to construct mathematical models from external knowledge. *Bioinformatics Advances* **4**, vbae042 (2024). https://doi.org/10.1093/bioadv/vbae042

CHAPTER

ONE

ABOUT

BioMASS is a computational framework for modeling and analysis of biological signaling systems in Python. It provides useful tools for model construction, numerical simulation, parameter estimation, network analysis, and result visualization.

1.1 Example

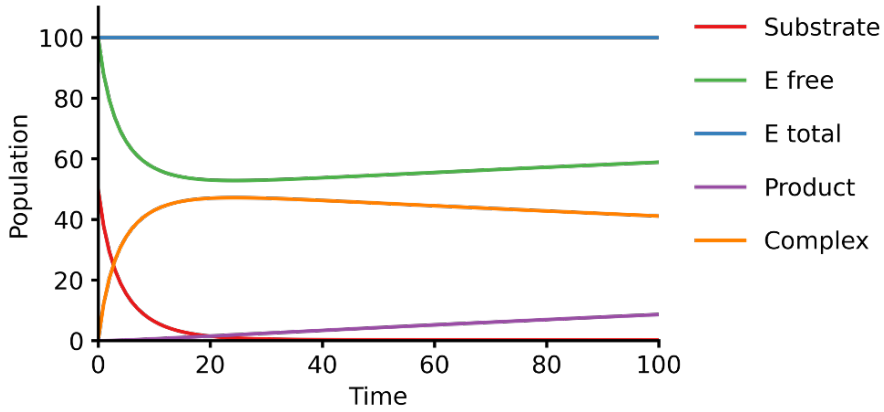
Text file (`michaelis_menten.txt`):

```
E + S  ES | kf=0.003, kr=0.001 | E=100, S=50
ES → E + P | kf=0.002

@obs Substrate: u[S]
@obs E_free: u[E]
@obs E_total: u[E] + u[ES]
@obs Product: u[P]
@obs Complex: u[ES]

@sim tspan: [0, 100]
```

Text-to-model conversion:

```
>>> from biomass import Text2Model, create_model, run_simulation
>>> description = Text2Model("michaelis_menten.txt")
>>> description.convert()
Model information
-----------------
2 reactions
4 species
4 parameters

>>> model = create_model("michaelis_menten")
>>> run_simulation(model)
```

Output:



For an advanced model, see a mechanistic model of the c-Fos expression network dynamics.

1.2 License

The software is released under the Apache License 2.0. For details, see the LICENSE file in the biomass repository.

1.3 Author

Hiroaki Imoto

1.4 Contact

Please contact me with any questions or comments via Issues | Discussions on GitHub. You can also always send me an email.

Any contributions to BioMASS are more than welcome!

CHAPTER

TWO

INSTALLATION

BioMASS supports Python 3.8 or newer.

2.1 PyPI

Install BioMASS from PyPI using:

```
pip install biomass
```

Note: If you wish to use graph visualization functions, install biomass via `pip install biomass[graph]`. In that case you will need to manually install Graphviz (version 2.42 or later).

2.2 Development version

If you want the latest development version, install from GitHub using:

```
pip install git+https://github.com/biomass-dev/biomass
```

CHAPTER

THREE

TUTORIAL

3.1 Tutorial 1: parameter estimation and sensitivity analysis

This tutorial shows you how to build computational models, estimate parameter values from experimental data, and identify sensitive components in complex biochemical systems. We will use a mechanistic model of the c-Fos expression network dynamics [Nakakuki *et al.*, 2010]. For a detailed description of the model, please refer to the following paper:

- Nakakuki, T. *et al*. Ligand-specific c-Fos expression emerges from the spatiotemporal control of ErbB network dynamics. *Cell* **141**, 884–896 (2010). https://doi.org/10.1016/j.cell.2010.03.054

3.1.1 Requirements

- `biomass>=0.8.0` for simulation, parameterization, and analysis of the model
- tqdm for visualizing progress bars

To check the software versions, run the following code:

```
import biomass
print('biomass version:', biomass.__version__)
```

3.1.2 Model preparation

A brief description of each file/folder is below:

Name	Content
`name2idx/`	Names of model parameters and species
`reaction_network.py`	Flux vector and reaction indices grouped according to biological processes
`ode.py`	Differential equation, parameters and initial condition
`observable.py`	Observables, simulations and experimental data
`search_param.py`	Lower and upper bounds of model parameters to be estimated
`problem.py`	An objective function to be minimized, i.e., the distance between model simulation and experimental data
`viz.py`	Plotting parameters for customizing figure properties

Note: Text2Model allows you to build a BioMASS model from text [Imoto *et al.*, 2022]. You simply describe biochemical reactions and the molecular mechanisms extracted from text are converted into an executable model.

Prepare a text file describing the biochemical reactions

By comparing the reaction scheme (Fig.1E) and the description below, you can learn how to build computational models via Text2Model.

```
@rxn ERKc --> pERKc: p[V1] * p[a] * u[ppMEKc] * u[ERKc] /  ( p[K1] * (1 + u[pERKc] /␣
↪p[K2]) + u[ERKc] ) || ERKc=9.60e02
@rxn pERKc --> ppERKc: p[V2] * p[a] * u[ppMEKc] * u[pERKc] /  ( p[K2] * (1 + u[ERKc] /␣
↪p[K1]) + u[pERKc] ) | const V2=2.20e-01, const K2=3.50e02
@rxn pERKc --> ERKc: p[V3] * u[pERKc] /  ( p[K3] * (1 + u[ppERKc] / p[K4]) + u[pERKc] )␣
↪| const V3=7.20e-01, const K3=1.60e02
@rxn ppERKc --> pERKc: p[V4] * u[ppERKc] /  ( p[K4]* (1 + u[pERKc] / p[K3]) + u[ppERKc]␣
↪) | const V4=6.48e-01, const K4=6.00e01
@rxn pERKn --> ERKn: p[V5] * u[pERKn] /  ( p[K5] * (1 + u[ppERKn] / p[K6]) + u[pERKn] )
@rxn ppERKn --> pERKn: p[V6] * u[ppERKn] /  ( p[K6] * (1 + u[pERKn] / p[K5]) + u[ppERKn]␣
↪) |5|
ERKc translocates to nucleus (0.94, 0.22) <--> ERKn | const kf=1.20e-02, const kr=1.80e-
↪02
pERKc translocates to nucleus (0.94, 0.22) <--> pERKn | const kf=1.20e-02, const kr=1.
↪80e-02
ppERKc translocates to nucleus (0.94, 0.22) <--> ppERKn | const kf=1.10e-02, const kr=1.
↪30e-02
ppERKn transcribes PreduspmRNAn
PreduspmRNAn translocates to cytoplasm --> duspmRNAc
duspmRNAc is degraded
duspmRNAc is translated into DUSPc
ppERKc phosphorylates DUSPc --> pDUSPc
pDUSPc is dephosphorylated --> DUSPc
DUSPc is degraded | const kf=2.57e-04
pDUSPc is degraded | const kf=9.63e-05
DUSPc translocates to nucleus (0.94, 0.22) <--> DUSPn  # assuming cytoplasmic and␣
↪nuclear volume to 0.94 pl and 0.22 pl
pDUSPc translocates to nucleus (0.94, 0.22) <--> pDUSPn |18|
ppERKn phosphorylates DUSPn --> pDUSPn
pDUSPn is dephosphorylated --> DUSPn
DUSPn is degraded | const kf=2.57e-04
pDUSPn is degraded | const kf=9.63e-05
ppERKc phosphorylates RSKc --> pRSKc || RSKc=3.53e02
pRSKc is dephosphorylated --> RSKc
pRSKc translocates to nucleus (0.94, 0.22) <--> pRSKn
pRSKn phosphorylates CREBn --> pCREBn || CREBn=1.00e03
pCREBn is dephosphorylated --> CREBn
ppERKn phosphorylates Elk1n --> pElk1n || Elk1n=1.51e03
pElk1n is dephosphorylated --> Elk1n
pCREBn & pElk1n transcribes PrecfosmRNAn, repressed by Fn
PrecfosmRNAn translocates to cytoplasm --> cfosmRNAc
cfosmRNAc is degraded
cfosmRNAc is translated into cFOSc
```

(continues on next page)

(continued from previous page)

```
ppERKc phosphorylates cFOSc --> pcFOSc
pRSKc phosphorylates cFOSc --> pcFOSc
pcFOSc is dephosphorylated --> cFOSc
cFOSc is degraded | const kf=2.57e-04
pcFOSc is degraded | const kf=9.63e-05
cFOSc translocates to nucleus (0.94, 0.22) <--> cFOSn
pcFOSc translocates to nucleus (0.94, 0.22) <--> pcFOSn |40|
ppERKn phosphorylates cFOSn --> pcFOSn
pRSKn phosphorylates cFOSn --> pcFOSn
pcFOSn is dephosphorylated --> cFOSn
cFOSn is degraded | const kf=2.57e-04
pcFOSn is degraded | const kf=9.63e-05
DUSPn + ppERKn <--> DUSPn_ppERKn
DUSPn_ppERKn --> DUSPn + pERKn
DUSPn + pERKn <--> DUSPn_pERKn
DUSPn_pERKn --> DUSPn + ERKn
DUSPn + ERKn <--> DUSPn_ERKn
pDUSPn + ppERKn <--> pDUSPn_ppERKn |47|
pDUSPn_ppERKn --> pDUSPn + pERKn |48|
pDUSPn + pERKn <--> pDUSPn_pERKn |49|
pDUSPn_pERKn --> pDUSPn + ERKn |50|
pDUSPn + ERKn <--> pDUSPn_ERKn |51|
pcFOSn transcribes PreFmRNAn
PreFmRNAn translocates to cytoplasm --> FmRNAc
FmRNAc is degraded
FmRNAc is translated into Fc
Fc is degraded
Fc translocates to nucleus (0.94, 0.22) <--> Fn
Fn is degraded

@add species ppMEKc
@add param Ligand

@obs Phosphorylated_MEKc: u[ppMEKc]
@obs Phosphorylated_ERKc: u[pERKc] + u[ppERKc]
@obs Phosphorylated_RSKw: u[pRSKc] + u[pRSKn] * (0.22 / 0.94)
@obs Phosphorylated_CREBw: u[pCREBn] * (0.22 / 0.94)
@obs dusp_mRNA: u[duspmRNAc]
@obs cfos_mRNA: u[cfosmRNAc]
@obs cFos_Protein: (u[pcFOSn] + u[cFOSn]) * (0.22 / 0.94) + u[cFOSc] + u[pcFOSc]
@obs Phosphorylated_cFos: u[pcFOSn] * (0.22 / 0.94) + u[pcFOSc]

@sim tspan: [0, 5400]
@sim unperturbed: p[Ligand] = 0
@sim condition EGF: p[Ligand] = 1
@sim condition HRG: p[Ligand] = 2
```

- You can download this text file from here.
- For more details about available reaction rules, please see ReactionRules.

Text-to-model conversion:

```
>>> from biomass import Text2Model
>>> description = Text2Model("cfos_model")
>>> description.convert()  # generate cfos_model/ in your working directory.
Model information
-----------------
63 reactions
36 species
110 parameters
```

You can also export model reactions as markdown files by running the following code:

```
>>> description.to_markdown(num_reactions=63)  # generate markdown/ in your working␣
↪directory.
```

Set the input of the model

The input for the mechanistic c-Fos model is given by an interpolation function of the ppMEK experimental data.

Open `ode.py`.

```
class DifferentialEquation(ReactionNetwork):

    def __init__(self, perturbation):
        super(DifferentialEquation, self).__init__()
        self.perturbation = perturbation

    @staticmethod
    def _get_ppMEK_slope(t, ligand) -> float:
        assert ligand in ['EGF', 'HRG']
        timepoints = [0, 300, 600, 900, 1800, 2700, 3600, 5400]
        ppMEK_data = {
            'EGF': [0.000, 0.773, 0.439, 0.252, 0.130, 0.087, 0.080, 0.066],
            'HRG': [0.000, 0.865, 1.000, 0.837, 0.884, 0.920, 0.875, 0.789],
        }
        assert len(timepoints) == len(ppMEK_data[ligand])
        slope = [
            (ppMEK_data[ligand][i + 1] - activity) / (timepoints[i + 1] - timepoint)
            for i, (timepoint, activity) in enumerate(zip(timepoints, ppMEK_
↪data[ligand]))
            if i + 1 < len(timepoints)
        ]
        for i, timepoint in enumerate(timepoints):
            if timepoint <= t <= timepoints[i + 1]:
                return slope[i]
        assert False

    # Refined Model
    def diffeq(self, t, y, *x):

        v = self.flux(t, y, x)

        if self.perturbation:
```

(continues on next page)

(continued from previous page)

```
            for i, dv in self.perturbation.items():
                v[i] = v[i] * dv

        dydt = [0] * V.NUM

        if x[C.Ligand] == 1:  # EGF=10nM
            dydt[V.ppMEKc] = self._get_ppMEK_slope(t, 'EGF')
        elif x[C.Ligand] == 2:  # HRG=10nM
            dydt[V.ppMEKc] = self._get_ppMEK_slope(t, 'HRG')
        else:  # Default: No ligand input
            dydt[V.ppMEKc] = 0.0

    ...
```

Normalize simulation results

Experimental data were normalized by dividing them by the maximum value of the responses. To correlate model simulation results with experimental measurements, we will need to normalize simulation results.

Open `observable.py`.

```
class Observable(DifferentialEquation):

    ...

    self.normalization: dict = {}
    for observable in self.obs_names:
        self.normalization[observable] = {"timepoint": None, "condition": []}
```

Here, you can define how you would like to normalize simulation results for each observable. The `normalization[observable]` dictionary accepts two keys, *'timepoint'* and *'condition'*.

- **'timepoint'**
  [*Optional[int]*] The time point at which simulated values are normalized. If `None`, the maximum value will be used for normalization.
- **'condition'**
  [*list of strings*] The experimental conditions to use for normalization. If empty, all conditions defined in `self.conditions` will be used.

Choose an ODE solver to use

Most systems biology models are non-linear and closed form solutions are not available. Accordingly, numerical integration methods have to be employed to study them [Maiwald and Timmer, 2008].

Open `observable.py` and choose integration method in `get_steady_state()` and `solve_ode()`.

```
class Observable(DifferentialEquation):

    ...

    def simulate(self, x, y0, _perturbation=None):
```

(continues on next page)

(continued from previous page)

```
        ...

        x[C.Ligand] = 0
        y0 = get_steady_state(self.diffeq, y0, tuple(x), integrator='vode')
        if not y0:
            return False

        for i, condition in enumerate(self.conditions):
            if condition == "EGF":
                x[C.Ligand] = 1
            elif condition == "HRG":
                x[C.Ligand] = 2

            sol = solve_ode(self.diffeq, y0, self.t, tuple(x), method="BDF")

            ...
```

- `get_steady_state` runs a model simulation till steady state for that parameter set. First, we simulate the model with no ligand until the system reaches steady state, take the final state of the equilibration simulation, and use it as the initial state of the new simulation.
- By default, LSODA is used in both integrators.

Set experimental data for parameterization of the model

- **self.experiments**
  [*list of dict*] Time-series experimetal measurements.
- **self.error_bars**
  [*list of dict*] Error bars to show in figures (e.g., SD or SE).

Open `observable.py`.

```
class Observable(DifferentialEquation):

    ...

    def set_data(self):

        self.experiments[self.obs_names.index("Phosphorylated_MEKc")] = {
            "EGF": [0.000, 0.773, 0.439, 0.252, 0.130, 0.087, 0.080, 0.066],
            "HRG": [0.000, 0.865, 1.000, 0.837, 0.884, 0.920, 0.875, 0.789],
        }
        self.error_bars[self.obs_names.index("Phosphorylated_MEKc")] = {
            "EGF": [
                sd / np.sqrt(3) for sd in [0.000, 0.030, 0.048, 0.009, 0.009, 0.017, 0.
↪012, 0.008]
            ],
            "HRG": [
                sd / np.sqrt(3) for sd in [0.000, 0.041, 0.000, 0.051, 0.058, 0.097, 0.
↪157, 0.136]
            ],
```

(continues on next page)

(continued from previous page)

```
        }

        self.experiments[self.obs_names.index("Phosphorylated_ERKc")] = {
            "EGF": [0.000, 0.867, 0.799, 0.494, 0.313, 0.266, 0.200, 0.194],
            "HRG": [0.000, 0.848, 1.000, 0.971, 0.950, 0.812, 0.747, 0.595],
        }
        self.error_bars[self.obs_names.index("Phosphorylated_ERKc")] = {
            "EGF": [
                sd / np.sqrt(3) for sd in [0.000, 0.137, 0.188, 0.126, 0.096, 0.087, 0.
↪056, 0.012]
            ],
            "HRG": [
                sd / np.sqrt(3) for sd in [0.000, 0.120, 0.000, 0.037, 0.088, 0.019, 0.
↪093, 0.075]
            ],
        }

        self.experiments[self.obs_names.index("Phosphorylated_RSKw")] = {
            "EGF": [0, 0.814, 0.812, 0.450, 0.151, 0.059, 0.038, 0.030],
            "HRG": [0, 0.953, 1.000, 0.844, 0.935, 0.868, 0.779, 0.558],
        }
        self.error_bars[self.obs_names.index("Phosphorylated_RSKw")] = {
            "EGF": [
                sd / np.sqrt(3) for sd in [0, 0.064, 0.194, 0.030, 0.027, 0.031, 0.043,␣
↪0.051]
            ],
            "HRG": [
                sd / np.sqrt(3) for sd in [0, 0.230, 0.118, 0.058, 0.041, 0.076, 0.090,␣
↪0.077]
            ],
        }

        self.experiments[self.obs_names.index("Phosphorylated_cFos")] = {
            "EGF": [0, 0.060, 0.109, 0.083, 0.068, 0.049, 0.027, 0.017],
            "HRG": [0, 0.145, 0.177, 0.158, 0.598, 1.000, 0.852, 0.431],
        }
        self.error_bars[self.obs_names.index("Phosphorylated_cFos")] = {
            "EGF": [
                sd / np.sqrt(3) for sd in [0, 0.003, 0.021, 0.013, 0.016, 0.007, 0.003,␣
↪0.002]
            ],
            "HRG": [
                sd / np.sqrt(3) for sd in [0, 0.010, 0.013, 0.001, 0.014, 0.000, 0.077,␣
↪0.047]
            ],
        }

        # ------------------------------------------------------------------------

        self.experiments[self.obs_names.index("Phosphorylated_CREBw")] = {
            "EGF": [0, 0.446, 0.030, 0.000, 0.000],
            "HRG": [0, 1.000, 0.668, 0.460, 0.340],
```

(continues on next page)

(continued from previous page)

```
        }
        self.error_bars[self.obs_names.index("Phosphorylated_CREBw")] = {
            "EGF": [sd / np.sqrt(3) for sd in [0, 0.0, 0.0, 0.0, 0.0]],
            "HRG": [sd / np.sqrt(3) for sd in [0, 0.0, 0.0, 0.0, 0.0]],
        }
        # ------------------------------------------------------------------

        self.experiments[self.obs_names.index("cfos_mRNA")] = {
            "EGF": [0, 0.181, 0.476, 0.518, 0.174, 0.026, 0.000],
            "HRG": [0, 0.353, 0.861, 1.000, 0.637, 0.300, 0.059],
        }
        self.error_bars[self.obs_names.index("cfos_mRNA")] = {
            "EGF": [sd / np.sqrt(3) for sd in [0.017, 0.004, 0.044, 0.004, 0.023, 0.007,␣
↪0.008]],
            "HRG": [sd / np.sqrt(3) for sd in [0.017, 0.006, 0.065, 0.044, 0.087, 0.023,␣
↪0.001]],
        }
        # ------------------------------------------------------------------

        self.experiments[self.obs_names.index("cFos_Protein")] = {
            "EGF": [0, 0.078, 0.216, 0.240, 0.320, 0.235],
            "HRG": [0, 0.089, 0.552, 0.861, 1.000, 0.698],
        }
        self.error_bars[self.obs_names.index("cFos_Protein")] = {
            "EGF": [sd / np.sqrt(3) for sd in [0, 0.036, 0.028, 0.056, 0.071, 0.048]],
            "HRG": [sd / np.sqrt(3) for sd in [0, 0.021, 0.042, 0.063, 0.000, 0.047]],
        }

        self.experiments[self.obs_names.index("dusp_mRNA")] = {
            "EGF": [0.000, 0.177, 0.331, 0.214, 0.177, 0.231],
            "HRG": [0.000, 0.221, 0.750, 1.000, 0.960, 0.934],
        }
        self.error_bars[self.obs_names.index("dusp_mRNA")] = {
            "EGF": [sd / np.sqrt(3) for sd in [0.033, 0.060, 0.061, 0.032, 0.068, 0.
↪050]],
            "HRG": [sd / np.sqrt(3) for sd in [0.027, 0.059, 0.094, 0.124, 0.113, 0.
↪108]],
        }

    @staticmethod
    def get_timepoint(obs_name) -> List[int]:
        """
        Time points at which experimental data was taken.
        """
        if obs_name in [
            "Phosphorylated_MEKc",
            "Phosphorylated_ERKc",
            "Phosphorylated_RSKw",
            "Phosphorylated_cFos",
        ]:
            return [0, 300, 600, 900, 1800, 2700, 3600, 5400]  # (Unit: sec.)
        elif obs_name == "Phosphorylated_CREBw":
```

(continues on next page)

(continued from previous page)

```
            return [0, 600, 1800, 3600, 5400]
        elif obs_name == "cfos_mRNA":
            return [0, 600, 1200, 1800, 2700, 3600, 5400]
        elif obs_name in ["cFos_Protein", "dusp_mRNA"]:
            return [0, 900, 1800, 2700, 3600, 5400]
        assert False
```

You can visualize experimental data defined here by running the following code:

```
from biomass import run_simulation

run_simulation(model, viz_type="experiment")
```

Set lower/upper bounds of parameters to be estimated

Open `search_param.py`.

```
class SearchParam(object):

    ...

    def get_region(self):

        ...

        search_rgn = np.zeros((2, len(x) + len(y0)))

        search_rgn[:, C.V1] = [7.33e-2, 6.60e-01]
        search_rgn[:, C.K1] = [1.83e2, 8.50e2]
        search_rgn[:, C.V5] = [6.48e-3, 7.20e1]
        search_rgn[:, C.K5] = [6.00e-1, 1.60e04]
        search_rgn[:, C.V10] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K10] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.n10] = [1.00, 4.00]
        search_rgn[:, C.kf11] = [8.30e-13, 1.44e-2]
        search_rgn[:, C.kf12] = [8.00e-8, 5.17e-2]
        search_rgn[:, C.kf13] = [1.38e-7, 4.84e-1]
        search_rgn[:, C.V14] = [4.77e-3, 4.77e1]
        search_rgn[:, C.K14] = [2.00e2, 2.00e6]
        search_rgn[:, C.V15] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K15] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.kf18] = [2.20e-4, 5.50e-1]
        search_rgn[:, C.kr18] = [2.60e-4, 6.50e-1]
        search_rgn[:, C.V20] = [4.77e-3, 4.77e1]
        search_rgn[:, C.K20] = [2.00e2, 2.00e6]
        search_rgn[:, C.V21] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K21] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.V24] = [4.77e-2, 4.77e0]
        search_rgn[:, C.K24] = [2.00e3, 2.00e5]
        search_rgn[:, C.V25] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K25] = [np.exp(-10), np.exp(10)]
```

(continues on next page)

(continued from previous page)

```
        search_rgn[:, C.kf26] = [2.20e-4, 5.50e-1]
        search_rgn[:, C.kr26] = [2.60e-4, 6.50e-1]
        search_rgn[:, C.V27] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K27] = [1.00e2, 1.00e4]
        search_rgn[:, C.V28] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K28] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.V29] = [4.77e-2, 4.77e0]
        search_rgn[:, C.K29] = [2.93e3, 2.93e5]
        search_rgn[:, C.V30] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K30] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.V31] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K31] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.n31] = [1.00, 4.00]
        search_rgn[:, C.kf32] = [8.30e-13, 1.44e-2]
        search_rgn[:, C.kf33] = [8.00e-8, 5.17e-2]
        search_rgn[:, C.kf34] = [1.38e-7, 4.84e-1]
        search_rgn[:, C.V35] = [4.77e-3, 4.77e1]
        search_rgn[:, C.K35] = [2.00e2, 2.00e6]
        search_rgn[:, C.V36] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K36] = [1.00e2, 1.00e4]
        search_rgn[:, C.V37] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K37] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.kf40] = [2.20e-4, 5.50e-1]
        search_rgn[:, C.kr40] = [2.60e-4, 6.50e-1]
        search_rgn[:, C.V42] = [4.77e-3, 4.77e1]
        search_rgn[:, C.K42] = [2.00e2, 2.00e6]
        search_rgn[:, C.V43] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K43] = [1.00e2, 1.00e4]
        search_rgn[:, C.V44] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K44] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.kf47] = [1.45e-4, 1.45e0]
        search_rgn[:, C.kr47] = [6.00e-3, 6.00e1]
        search_rgn[:, C.kf48] = [2.70e-3, 2.70e1]
        search_rgn[:, C.kf49] = [5.00e-5, 5.00e-1]
        search_rgn[:, C.kr49] = [5.00e-3, 5.00e1]
        search_rgn[:, C.kf50] = [3.00e-3, 3.00e1]
        search_rgn[:, C.kf51] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.kr51] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.V57] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.K57] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.n57] = [1.00, 4.00]
        search_rgn[:, C.kf58] = [8.30e-13, 1.44e-2]
        search_rgn[:, C.kf59] = [8.00e-8, 5.17e-2]
        search_rgn[:, C.kf60] = [1.38e-7, 4.84e-1]
        search_rgn[:, C.kf61] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.kf62] = [2.20e-4, 5.50e-1]
        search_rgn[:, C.kr62] = [2.60e-4, 6.50e-1]
        search_rgn[:, C.kf63] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.KF31] = [np.exp(-10), np.exp(10)]
        search_rgn[:, C.nF31] = [1.00, 4.00]
        search_rgn[:, C.a] = [1.00e2, 5.00e2]
```

- Lower bound must be smaller than upper bound.

- Lower/upper buonds must be positive.

Create a new model

BioMASS core functions require `ModelObject` in the first argument.

```
>>> from biomass import create_model
>>> model = create_model('cfos_model')  # Create a new BioMASS model object.
```

In the following examples, you will use the BioMASS model object: `model` created here for parameter estimation, visualization of simulation results, and sensitivity analysis.

Need help?

If you get an error or need help, please head over to GitHub Issues.

3.1.3 Parameter estimation

Using `optimize()` function

An important step in the development of a mathematical model for a biological system is to identify model parameters. Parameters are adjusted to minimize the distance between model simulation and experimental data.

- Set simulation conditions and the corresponding experimental data in `observable.py`
- Define an objective function to be minimized (`objective()`) in `problem.py`
- Set lower/upper bounds of parameters to be estimated in `search_param.py`

```
from tqdm import tqdm
from biomass import optimize

# Get 30 parameter sets, it will take more than a few hours
for x_id in tqdm(range(1, 31)):
    optimize(model, x_id=x_id, disp_here=False, optimizer_options={"workers": -1})
```

Note: `"workers"` specifies the number of processes to use (default: 1). Set to a larger number (e.g. the number of CPU cores available) for parallel execution of optimizations. For detailed information about `optimizer_options`, please refer to scipy docs.

The progress list will be saved in `out/{x_id}/`:

```
differential_evolution step 1: f(x)= 4.96181
differential_evolution step 2: f(x)= 3.555
differential_evolution step 3: f(x)= 2.50626
differential_evolution step 4: f(x)= 2.00657
differential_evolution step 5: f(x)= 1.83556
differential_evolution step 6: f(x)= 1.28031
differential_evolution step 7: f(x)= 0.973207
differential_evolution step 8: f(x)= 0.741667
differential_evolution step 9: f(x)= 0.741667
```

(continues on next page)

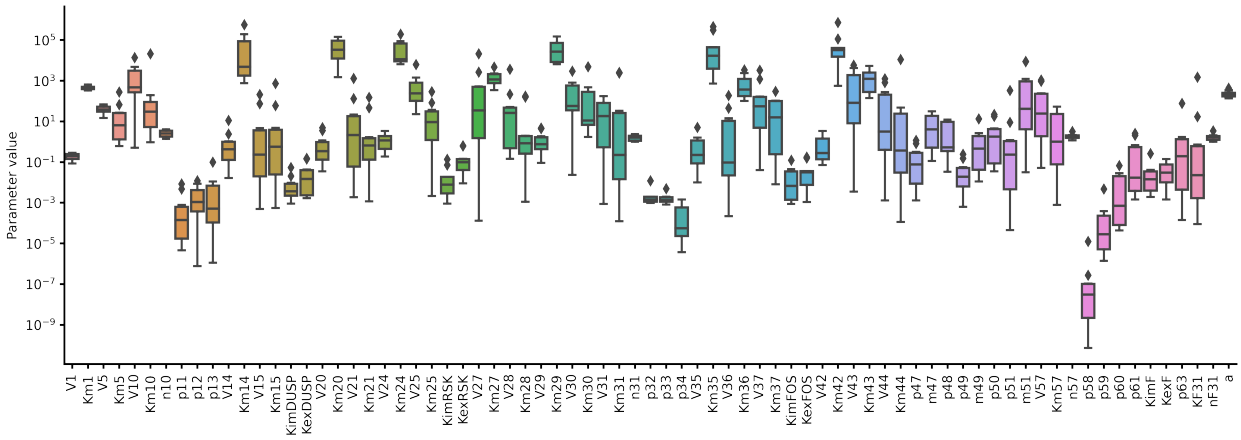
(continued from previous page)

```
differential_evolution step 10: f(x)= 0.735682
differential_evolution step 11: f(x)= 0.717266
differential_evolution step 12: f(x)= 0.603178
differential_evolution step 13: f(x)= 0.56934
differential_evolution step 14: f(x)= 0.56934
differential_evolution step 15: f(x)= 0.549331
differential_evolution step 16: f(x)= 0.459069
differential_evolution step 17: f(x)= 0.447772
differential_evolution step 18: f(x)= 0.430385
differential_evolution step 19: f(x)= 0.37085
differential_evolution step 20: f(x)= 0.37085
```

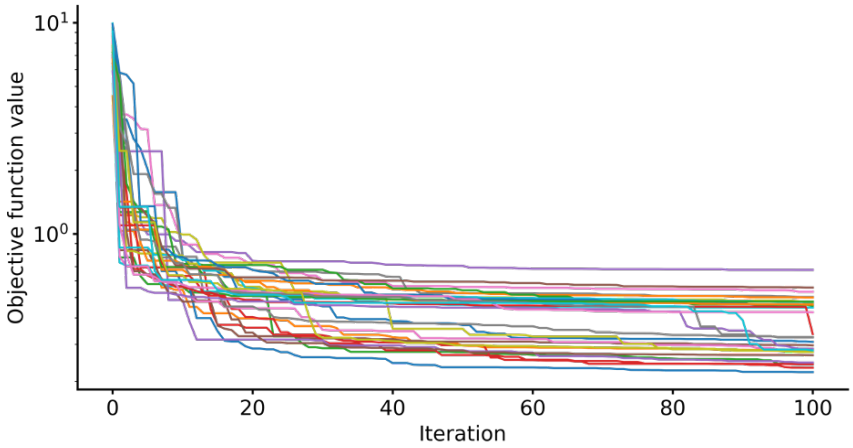
To print the evaluated *func* at every iteration, set `disp_here` to `True`.

Data export and visualization

```
from biomass.result import OptimizationResults

res = OptimizationResults(model)
# Export estimated parameters in CSV format
res.to_csv()
# Visualize estimated parameter sets
res.savefig(figsize=(16,5), boxplot_kws={"orient": "v"})
```

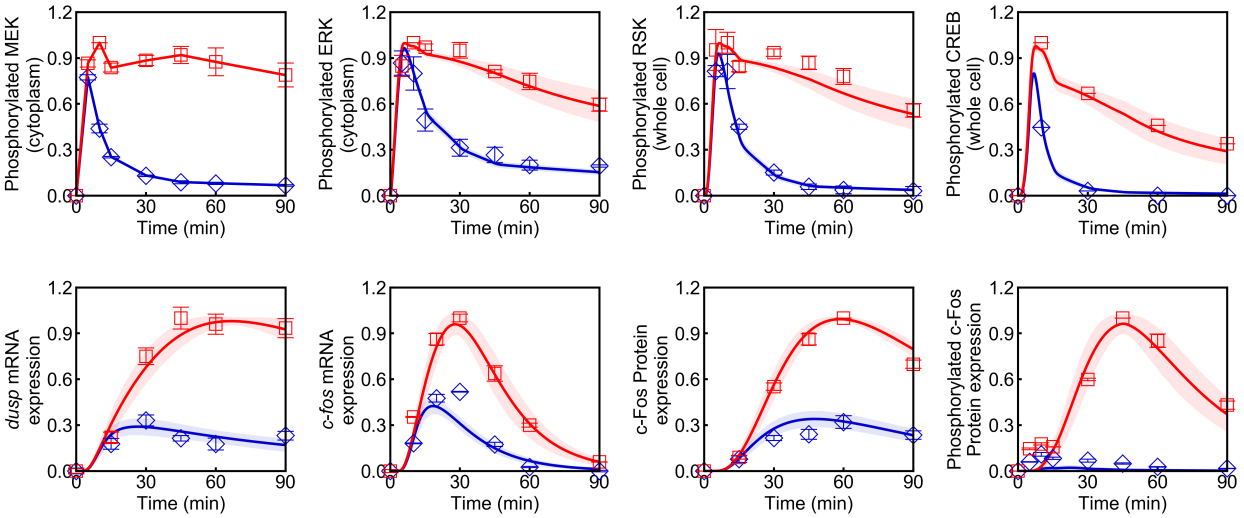



```
# Visualize objective function traces for different optimization runs.
res.trace_obj()
```



3.1.4 Visualization of simulation results

```
from biomass import run_simulation

run_simulation(model, viz_type='average', show_all=False, stdev=True)
```



Points (blue diamonds, EGF; red squares, HRG) denote experimental data, solid lines denote simulations.

3.1.5 Sensitivity analysis

Sensitivity analysis examines how perturbations to the processes in the model affect the quantity of interest, e.g., the integral of the pc-Fos concentration.

To perform sensitivity analysis on reaction rates (`target='reaction'`), you will need to modify `reaction_network.py` in the model folder as follows:

```
class ReactionNetwork(object):

    def __init__(self) -> None:
```

(continues on next page)

(continued from previous page)

```
        """
        Reaction indices grouped according to biological processes.
        This is used for sensitivity analysis (target='reaction').
        """
        super(ReactionNetwork, self).__init__()

        self.reactions: Dict[str, List[int]] = {
            "ERK_activation": [i for i in range(1, 7)],
            "ERK_dephosphorylation_by_DUSP": [i for i in range(47, 57)],
            "ERK_transport": [i for i in range(7, 10)],
            "RSK_activation": [24, 25],
            "RSK_transport": [26],
            "Elk1_activation": [29, 30],
            "CREB_activation": [27, 28],
            "dusp_production_etc": [i for i in range(10, 14)],
            "DUSP_transport": [18, 19],
            "DUSP_stabilization": [14, 15, 20, 21],
            "DUSP_degradation": [16, 17, 22, 23],
            "cfos_production_etc": [i for i in range(31, 35)],
            "cFos_transport": [40, 41],
            "cFos_stabilization": [35, 36, 37, 42, 43, 44],
            "cFos_degradation": [38, 39, 45, 46],
            "Feedback_from_F": [i for i in range(57, 64)],
        }

        ...
```

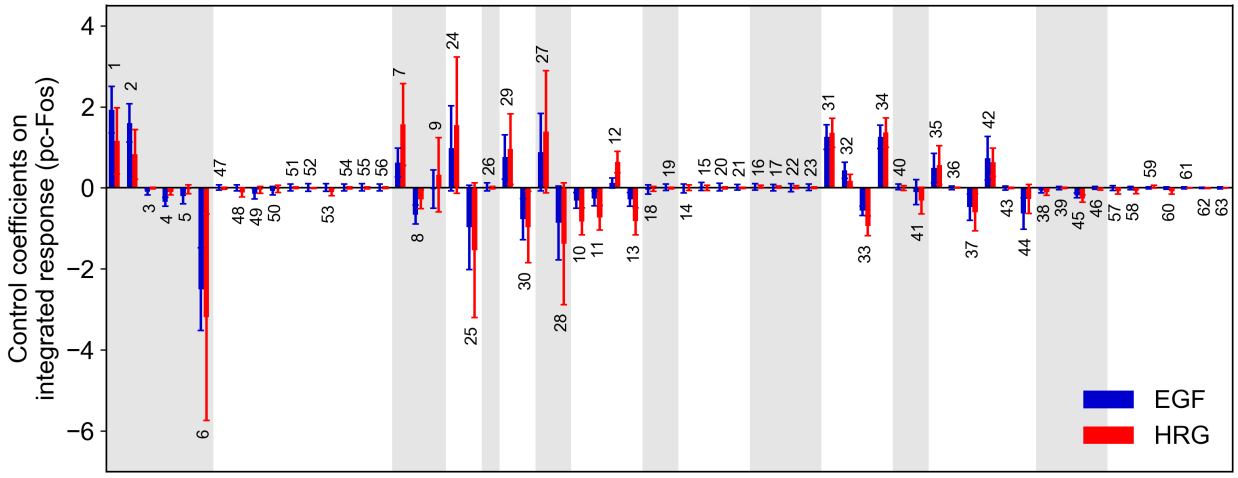
Then, run the following code:

```
from biomass import run_analysis

run_analysis(model, target='reaction', metric='integral', style='barplot', options={
↪'overwrite': True})
```

The single parameter sensitivity of each reaction is defined by

$$C_i^M = d\ln M / d\ln v_i$$

where v_i is the i^{th} reaction rate, v is reaction vector v = (v_1, v_2, ...) and M is a signaling metric, e.g., time-integrated response, duration. Sensitivity coefficients are calculated using finite difference approximations with 1% changes in the reaction rates [Kholodenko *et al.*, 1997].



Control coefficients for integrated pc-Fos are shown by bars (blue, EGF; red, HRG). Numbers above bars indicate the reaction indices, and error bars correspond to simulation standard deviation.

Note: If you want to reuse a result from the previous computation and don't want to calculate sensitivity coefficients again, set `options['overwrite']` to `False`.

3.2 Tutorial 2: graph visualization

This tutorial will show you how to turn your Pasmopy Text model into a graph and generate both static and dynamic images from it. For demonstration purposes we will use a text representation of the nfkb_pathway model included in biomass. For a detailed description of the model, please refer to the following paper:

- Oppelt, A. *et al*. Model-based identification of TNF-induced IKK-mediated and IB-mediated regulation of NFB signal transduction as a tool to quantify the impact of drug-induced liver injury compounds. *npj Syst. Biol. Appl.* **4**, 23 (2018). https://doi.org/10.1038/s41540-018-0058-z

3.2.1 Requirements

- `biomass>=0.9.0`
- `pygraphviz>=1.9`
- `pyvis>=0.2.1`
- `graphviz>=2.42` Installation instructions can be found here

3.2.2 Prepare a text file describing the model

```
TNF synthesizes TNFR| kf=1 | TNF=1
TNFR is degraded | kf=0.001
TNFR + Ikk --> pIkk + TNFR | kf=0.0714 | Ikk=1
pIkk is phosphorylated --> ppIkk | kf=0.0648
ppIkk --> iIkk | kf=0.166
iIkk --> Ikk | kf=0.0041
pIkk + NfkIkb --> NfkpIkb + pIkk | kf=0.398 | NfkIkb=1
pNfkIkb is phosphorylated --> pNfkpIkb | kf=1.3897
pIkk + pNfkIkb --> pNfkpIkb + pIkk | kf=0.389
pIkk + NfkIkb --> pNfkIkb + pIkk | kf=0.6438
pIkk + NfkpIkb --> pNfkpIkb + pIkk | kf=0.2816
NfkpIkb -->  Nfk + pIkb | kf=0.0811
pNfkpIkb --> pNfk + pIkb | kf=1
Nfk + Ikb --> NfkIkb | kf=2.839
nNfk synthesizes mIkb | kf=0.0047
mIkb is degraded | kf=0.0313
mIkb synthesizes Ikb | kf=1
pIkb is degraded | kf=0.6308
Ikb translocates from cytoplasm to nucleus (1, 1) --> nIkb | kf=0.1226
pNfk translocates from cytoplasm to nucleus (1, 1) --> pnNfk | kf=0.179585
Nfk translocates from cytoplasm to nucleus (1, 1) --> nNfk | kf=0.01
pnNfk --> nNfk | kf=1000
nIkb binds nNfk --> nNfkIkb | kf=1000
nNfkIkb translocates from nucleus to cytoplasm (1, 1) --> NfkIkb | kf=1000
nNfk synthesizes RnaA20_1 | kf=1
RnaA20_1 --> RnaA20 | kf=0.0311
RnaA20 is degraded | kf=0.0089
RnaA20 synthesizes A20 | kf=0.0006
A20 is degraded | kf=0.0116

@obs nuclear_IkBa: u[nIkb]
@obs nuclear_NFkB: u[nNfk]

@sim tspan: [0, 200]
```

Note: For further details about setting up a biomass model from text please refer to the corresponding biomass and pasmopy tutorials.

3.2.3 Import the model

```
from biomass import Text2Model

model = Text2Model('name_of_your_txt_file.txt')

model.convert()
```

3.2.4 Graph generation and static image

The graph is constructed from the kinetic information gained during model construction. Connections always go from reactants/modifiers to products. There is no destinction made between modifiers and reactants, as well as activating and inhibiting modifiers.

```
model.graph
```

The graph property contains an instance of the AGraph class implemented by pygraphviz. For available methods please refer to their documentation. You can for example manually add/remove nodes or save the graph into a .dot file and import it into another 3rd party software.

A static image of the graph is drawn using

```
model.static_plot(save_dir='example_dir', file_name='nfkb_static.png')
model.static_plot(save_dir='example_dir', file_name='nfkb_static_cust.png',
                  gviz_args='-Nshape=parallelogram -Nstyle=bold -Estyle=dashed')
```

The desired file format is inferred from the ending of file_name. Graphviz provides a variety of different engines that automatically generate a layout for the graph. By default the 'dot' engine is used, since it uses a hierarchical approach that is natural for biological data. Feel free to play around with the available engines, but be aware that biological networks can quickly become messy due to the prevalance of feedback interactions. Additionally graphviz provides a large variety of customization options, that have to be passed in the command line format. For a comprehensive list see the graphviz manual.

3.2.5 Dynamic image

Thanks to the package pyvis we can also provide an interactive graph. The generation is just as simple as for the static image:

```
model.dynamic_plot(save_dir='example_dir', file_name='nfkb_dynamic.html' show_
↪controls=True, which_controls=['physics', 'layout'])
```

By default the plot will be immediately displayed in your browser. Set `show` to `False` if you don't want that. `pyvis` provides a variety of customization options as well. They can be directly accessed in the html file by setting `show_controls` to `True`. You can also specify which controls you want.

CHAPTER

FOUR

EXAMPLE MODELS

4.1 Circadian clock

- **Code:** circadian_clock
- **Paper:** [Leloup and Goldbeter, 2003]

4.2 G1/S transition

- **Code:** g1s_transition
- **Paper:** [Barr *et al.*, 2016]

4.3 Immediate-early gene response

- **Code:** Nakakuki_Cell_2010
- **Paper:** [Nakakuki *et al.*, 2010]

4.4 Insulin signaling

- **Code:** insulin_signaling
- **Paper:** [Kubota *et al.*, 2012]

4.5 MAPK cascade

- **Code:** mapk_cascade
- **Paper:** [Kholodenko, 2000]

4.6 NF-B pathway

- **Code:** nfkb_pathway
- **Paper:** [Oppelt *et al.*, 2018]

4.7 pan-RTK

- **Code:** pan_rtk
- **Paper:** [Hass *et al.*, 2017]

4.8 Proliferation-quiescence decision

- **Code:** prolif_quies
- **Paper:** [Heldt *et al.*, 2018]

4.9 TGF-/SMAD pathway

- **Code:** tgfb_smad
- **Paper:** [Lucarelli *et al.*, 2018]

CHAPTER

FIVE

API

5.1 BioMASS model object (biomass.model_object)

class biomass.model_object.**ModelObject**(*path*, *biomass_model*)

The BioMASS model object.

Examples

```
>>> from biomass import create_model
>>> from biomass.models import mapk_cascade
>>> model = create_model(mapk_cascade.__package__)
>>> type(model)
<class 'biomass.model_object.ModelObject'>
>>> print('Parameters:', len(model.parameters))
Parameters: 22
>>> print('Species:', len(model.species))
Species: 8
>>> print('Observables:', len(model.observables))
Observables: 2
```

gene2val(*indiv_gene*)

Convert gene to actual value. Mainly used in parameter estimation.

Parameters
indiv_gene (numpy.ndarray) – Individual gene.

Returns
indiv_values – Corresponding values.

Return type
numpy.ndarray

get_executable()

Get executable parameter sets from optimization results.

Return type
List[int]

get_individual(*paramset_id*)

Get estimated parameter values from optimization results.

Parameters
paramset_id (*int*) – Index of parameter set.

Returns
best_individual – Estimated parameter values.

Return type
numpy.ndarray

`get_obj_val`(*indiv_gene*)

An objective function to minimize in parameter estimation.

Parameters
indiv_gene (numpy.ndarray) – Genes, not parameter values.

Returns
obj_val – Objective function value.

Return type
float

`load_param`(*paramset*)

Load a parameter set from optimization results.

Parameters
paramset (*int*) – Index of parameter set.

Returns
optimized_values – Optimized parameter/initial values.

Return type
biomass.model_object.OptimizedValues

5.2 Example models (biomass.models)

biomass.models contains sample models and other scripts if you would prefer to learn how to use biomass by example. For details, please refer to https://biomass-core.readthedocs.io/en/latest/models.html.

biomass.models._copy.`copy_to_current`(*model_name*)

Copy an example model to the current working directory.

To execute example models in your machine, please follow these steps:

```
>>> from biomass import create_model
>>> from biomass.models import copy_to_current
>>> copy_to_current('XXX')
>>> model = create_model('XXX')
```

Parameters
model_name (*str*) – Name of the example model.

Return type
None

5.3 Construction of executable models from text (biomass.construction)

The Text2Model class converts texts and sentences on biochemical systems into an executable mathematical model. It was originally developed in the pasmopy project[1][2].

References

5.3.1 Thermodynamic restrictions (biomass.construction.thermodynamic_restrictions)

class biomass.construction.thermodynamic_restrictions.**ThermodynamicRestrictions**

Thermodynamic restrictions along cyclic pathways in a kinetic scheme.

Notes

If a kinetic scheme includes "true" cycles, in which the initial and final states are identical, the equilibrium constants of the reactions along any cycle satisfy so-called "detailed balance" relationships. These detailed balance relations require the product of the equilibrium constants along a cycle to be equal to 1, since at equilibrium the net flux through any cycle vanishes.

find_cyclic_reaction_routes()

Find cyclic pathways in a reaction network.

Return type
None

[1] Imoto, H., Yamashiro, S. & Okada, M. A text-based computational framework for patient -specific modeling for classification of cancers. iScience 25, 103944 (2022).

[2] Imoto, H., Yamashiro, S., Murakami, K. & Okada, M. Protocol for stratification of triple-negative breast cancer patients using in silico signaling dynamics. STAR Protocols 3, 101619 (2022).

5.3.2 Available reaction rules (biomass.construction.reaction_rules)

class biomass.construction.reaction_rules.**ReactionRules**(*input_txt*, *similarity_threshold*)

Create an executable biochemical model from text.

Table 1: Available reaction rules

Rule	Example sentence	Parameters (optional)
`dimerize()`	*A* dimerizes <–> *AA*	kf, kr
`bind()`	*A* binds *B* <–> *AB*	kf, kr
`dissociate()`	*AB* dissociates to *A* and *B*	kf, kr
`is_phosphorylated()`	*uA* is phosphorylated <–> *pA*	kf, kr
`is_dephosphorylated(`	*pA* is dephosphorylated –> *uA*	V, K
`phosphorylate()`	*B* phosphorylates *uA* –> *pA*	V, K
`dephosphorylate()`	*B* dephosphorylates *pA* –> *uA*	V, K
`transcribe()`	*B* transcribes *a*	$V, K, n, (KF, nF)$
`synthesize()`	*B* synthesizes *A*	kf
`is_synthesized()`	*A* is synthesized	kf
`degrade()`	*B* degrades *A*	kf
`is_degraded()`	*A* is degraded	kf

From v0.2.2, you can specify directionality in binding-dissociation reaction via different arrows:

```
E + S | ES | kf=0.003, kr=0.001 | E=100, S=50  # bi-directional
ES ⇢ E + P | kf=0.002  # unidirectional
```

input_txt

Model description file (*.txt), e.g., Kholodenko1999.txt.

Type
str

similarity_threshold

If all match_scores are below this value, expected_word will not be returned.

Type
float

parameters

x : model parameters.

Type
list of strings

species

y : model species.

Type
list of strings

reactions

v : flux vector.

Type
list of strings

differential_equations

dydt : right-hand side of the differential equation.

Type
list of strings

obs_desc

Description of observables.

Type
list of List[str]

param_info

Information about parameter values.

Type
list of strings

init_info

Information about initial values.

Type
list of strings

`param_constraints`

Information about parameter constraints.

Type
list of strings

`param_excluded`

List of parameters excluded from search params because of parameter constraints.

Type
list of strings

`fixed_species`

List of species which should be held fixed (never consumed) during simulation.

Type
list of strings

`sim_tspan`

Interval of integration.

Type
list of strings ['t0', 'tf']

`sim_conditions`

Simulation conditions with stimulation.

Type
list of List[str]

`sim_unperturbed`

Untreated conditions to get steady state.

Type
str

`rule_words`

Words to identify reaction rules.

Type
dict

`nothing`

Available symbol for degradation/creation to/from nothing.

Type
List[str]

`fwd_arrows`

Available arrows for unidirectional reactions.

Type
List[str]

`double_arrows`

Available arrows for bi-directional reactions.

Type
List[str]

_bind_and_dissociate(*line_num*, *line*)

Return type
None

Examples

```
>>> 'A + B --> AB'  # bind, unidirectional
>>> 'AB --> A + B'  # dissociate, unidirectional
>>> 'A + B <--> AB' # bind and dissociate, bidirectional
```

dimerize(*line_num*, *line*)

Return type
None

Examples

```
>>> 'A dimerizes <--> AA'
>>> 'A homodimerizes <--> AA'
>>> 'A forms a dimer <--> AA'
>>> 'A forms dimers <--> AA'
```

Notes

- **Parameters**

$$kf, kr$$

- **Rate equation**

$$v = kf * [A] * [A] - kr * [AA]$$

- **Differential equation**

$$d[A]]/dt = -2 * v$$
$$d[AA]/dt = +v$$

bind(*line_num*, *line*)

Return type
None

Examples

```
>>> 'A binds B <--> AB'
>>> 'A forms complexes with B <--> AB'
```

Notes

- **Parameters**

$$kf, kr$$

- **Rate equation**

$$v = kf * [A] * [B] - kr * [AB]$$

- **Differential equation**

$$d[A]/dt = -v$$
$$d[B]/dt = -v$$
$$d[AB]/dt = +v$$

`is_phosphorylated`(*line_num*, *line*)

Return type
None

Examples

```
>>> 'uA is phosphorylated <--> pA'
```

Notes

- **Parameters**

$$kf, kr$$

- **Rate equation**

$$v = kf * [uA] - kr * [pA]$$

- **Differential equation**

$$d[uA]/dt = -v$$
$$d[pA]/dt = +v$$

`is_dephosphorylated`(*line_num*, *line*)

Return type
None

Examples

```
>>> 'pA is dephosphorylated --> uA'
```

Notes

- **Parameters**

$$V, K$$

- **Rate equation**

$$v = V * [pA]/(K + [pA])$$

- **Differential equation**

$$d[uA]/dt = +v$$
$$d[pA]/dt = -v$$

phosphorylate(*line_num*, *line*)

Return type
None

Examples

```
>>> 'B phosphorylates uA --> pA'
```

Notes

- **Parameters**

$$V, K$$

- **Rate equation**

$$v = V * [B] * [uA]/(K + [uA])$$

- **Differential equation**

$$d[uA]/dt = -v$$
$$d[pA]/dt = +v$$

dephosphorylate(*line_num*, *line*)

Return type
None

Examples

```
>>> 'B dephosphorylates pA --> uA'
```

Notes

- **Parameters**

$$V, K$$

- **Rate equation**

$$v = V * [B] * [pA] / (K + [pA])$$

- **Differential equation**

$$d[uA]/dt = +v$$
$$d[pA]/dt = -v$$

transcribe(*line_num*, *line*)

Return type
None

Examples

```
>>> 'B transcribes a'
>>> 'B1 & B2 transcribe a'  # (AND-gate)
>>> 'B transcribes a, repressed by C'  # (Negative regulation)
```

Notes

- **Parameters**

$$V, K, n, (KF, nF)$$

- **Rate equation**

$$v = V * [B]^n / (K^n + [B]^n)$$
$$v = V * ([B1] * [B2])^n / (K^n + ([B1] * [B2])^n)$$
$$v = V * [B]^n / (K^n + [B]^n + ([C]/KF)^{nF})$$

- **Differential equation**

$$d[a]/dt = +v$$

synthesize(*line_num*, *line*)

Return type
None

Examples

```
>>> 'B synthesizes A'
```

Notes

- **Parameters**

$$kf$$

- **Rate equation**

$$v = kf * [B]$$

- **Differential equation**

$$d[A]/dt = +v$$

is_synthesized(*line_num*, *line*)

Return type
None

Examples

```
>>> 'A is synthesized'
```

Notes

- **Parameters**

$$kf$$

- **Rate equation**

$$v = kf$$

- **Differential equation**

$$d[A]/dt = +v$$

degrade(*line_num*, *line*)

Return type
None

Examples

```
>>> 'B degrades A'
```

Notes

- **Parameters**

$$kf$$

- **Rate equation**

$$v = kf * [B] * [A]$$

- **Differential equation**

$$d[A]/dt = -v$$

is_degraded(*line_num*, *line*)

Return type
None

Examples

```
>>> 'A is degraded'
```

Notes

- **Parameters**

$$kf$$

- **Rate equation**

$$v = kf * [A]$$

- **Differential equation**

$$d[A]/dt = -v$$

translocate(*line_num*, *line*)

Return type
None

Examples

```
>>> 'A_at_cyt translocates from cytoplasm to nucleus (V_cyt, V_nuc) <--> A_at_
↪nuc'
>>> 'A_at_cyt is translocated from cytoplasm to nucleus (V_cyt, V_nuc) <--> A_
↪at_nuc'
```

Notes

- **Parameters**

$$kf, kr, (V_{pre}, V_{post})$$

- **Rate equation**

$$v = kf * [A\_at\_pre] - kr * (V_{post}/V_{pre}) * [A\_at\_post]$$

- **Differential equation**

$$d[A\_at\_pre]/dt = -v$$
$$d[A\_at\_post]/dt = +v * (V_{pre}/V_{post})$$

state_transition(*line_num*, *line*)

This rule is applied only when any rule words are not detected.

Return type
None

Examples

'Reactant –> Product' 'Reactant <–> Product' 'E + S –> E + P'

Notes

- **Parameters**

$$kf(, kr)$$

- **Rate equation**

$$v = kf * [Reactant](-kr * [Product])$$

- **Differential equation**

$$d[Reactant]/dt = -v$$

$$d[Product]/dt = +v$$

user_defined(*line_num*, *line*)

Return type
None

Examples

```
>>> '@rxn Reactant --> Product: define rate equation here'
```

Notes

- Use p[xxx] and u[xxx] for describing parameters and species, respectively.
- Use '0' or '' for degradation/creation to/from nothing.
- **Differential equation**

$$d[Reactant]/dt = -v$$
$$d[Product]/dt = +v$$

5.3.3 Text-to-model conversion (`biomass.construction.text2model`)

class biomass.construction.text2model.**Text2Model**(*input_txt*, *similarity_threshold=0.7*, *lang='python'*)

Build a BioMASS-formatted model based on template.

reaction | parameters | initial conditions

input_txt

Model description file (*.txt), e.g., 'Kholodenko1999.txt'

Type
str

similarity_threshold

Similarity threshold used in text-to-model conversion. Must lie within (0, 1).

Type
float (default: 0.7)

lang

Either 'python' or 'julia'.

- 'python': biomass (https://github.com/biomass-dev/biomass)
- 'julia': BioMASS.jl (https://github.com/biomass-dev/BioMASS.jl)

Type
Literal["python", "julia"] (default: 'python')

convert(*, *show_restrictions=False*, *overwrite=False*)

Convert text to a biomass-formatted model.

Parameters

- **show_restrictions** (bool (defauld: `False`)) – Whether to display reaction indices in which thermodynamic restrictions should be imposed. These detailed balance constraints require the product of the equilibrium constants along a cycle to be equal to 1.
- **overwrite** (bool (defauld: `False`)) – If `True`, the model folder will be overwritten.

Return type
`None`

Examples

```
>>> from pasmopy import Text2Model
>>> Text2Model("Kholodenko1999.txt").convert()
```

dynamic_plot(*save_dir='.'*, *file_name='network.html'*, *show=True*, *annotate_nodes=True*, *show_controls=False*, *which_controls=None*)

Saves a dynamic and interactive image of the network graph. Graph is read by `pyvis`. Using `pyvis` a dynamic and interactive representation of the biological network is created in html format.

Parameters

- **show** (*bool, default=True*) – If `True` the plot will immediately be displayed in the webbrowser.
- **annotate_nodes** (*bool, default=True*) – If `True` nodes will be scaled according to number of edges and hovering over a node will show interaction partners.
- **show_controls** (*bool, default=False*) – If `True` control buttons will be displayed.
- **which_controls** (*List(str), optional, default=None*) – Used to specify which control buttons should be displayed. If empty all buttons will be displayed.

Return type
None

Examples

```
>>> model.dynamic_plot("path/to/", "graph.html")
Creates graph and shows interactive graph with default options.
>>> model.dynamic_plot("path/to/", "graph.html", show=False, show_controls=True,
↪ which_controls=["physics", "manipulation", "interaction"])
Creates interactive graph. Controls for physics, manipulation and interaction␣
↪will be available.
```

register_word(*terminology=None*)

Register user-defined rule word.

Parameters
terminology (*Dict[str, List[str]], optional*) – Pair of reaction rule and user-defined rule words.

Return type
None

Examples

```
>>> from pasmopy import Text2Model
>>> mm_kinetics = Text2Model("michaelis_menten.txt")
>>> mm_kinetics.register_word({"dissociate": ["releases"]})
>>> mm_kinetics.convert()
```

static_plot(*save_dir=''*, *file_name='model_graph.png'*, *gviz_args=''*, *gviz_prog='dot'*)

Saves a static image of the network.

Static image is created using `pygraphviz`.

Parameters

- **save_dir** (*string*) – Name of the directory in which the image will be stored.
- **file_name** (*string*) – Name as which the image of the graph will be stored.
- **gviz_args** (*string, optional, default=""*) – Used to specify command line options for gviz, see https://graphviz.org/pdf/dot.1.pdf for available options.
- **gviz_prog** (*{"neato", "dot", "twopi", "circo", "fdp", "nop"}, default="dot"*) – Layout engine with which the graph will be arranged. For details see https://graphviz.org/docs/layouts/ .

Raises

ValueError – If something is passed as the gviz_prog that is not a viable layout program.

Return type

None

Examples

```
>>> model.static_plot("path/to/", "graph.png")
Creates graph with dot layout and default options.
>>> model.static_plot("path/to/", "graph.pdf", gviz_prog="-Nshape=box -
↪Nstyle=filled -Nfillcolor="#ffe4c4" -Edir=none")
Creates graph with dot layout in pdf file format. Nodes will be rectangular and␣
↪colored bisque, edges will have no arrows indicating direction.
```

to_markdown(*num_reactions*, *savedir='markdown'*)

Create markdown table describing differential equations.

Parameters

- **num_reactions** (*int*) – The number of rate equations in the model.
- **savedir** (*str (default: "markdown")*) – The directory name to save the output.

Return type

None

Examples

```
>>> from pasmopy import Text2Model
>>> Text2Model("Kholodenko1999.txt").to_markdown(25)
```

5.3.4 Template for BioMASS model construction (biomass.construction.template)

class biomass.construction.template.ode.**DifferentialEquation**(*perturbation*)

diffeq(*t*, *y*, **x*)

Kinetic equations

biomass.construction.template.ode.**initial_values**()

Values of the initial condition

biomass.construction.template.ode.**param_values**()

Parameter values

class biomass.construction.template.observable.**Observable**

Correlating model simulations and experimental measurements.

obs_names

Names of model observables.

Type
list of strings

t

Simulation time span.

Type
range

conditions

Experimental conditions.

Type
list of strings

simulations

The numpy array to store simulation results.

Type
numpy.ndarray

normalization

- **'timepoint'**
  [Optional[int]] The time point at which simulated values are normalized. If `None`, the maximum value will be used for normalization.
- **'condition'**
  [list of strings] The experimental conditions to use for normalization. If empty, all conditions defined in `sim.conditions` will be used.

Type
nested dict

experiments

Time series data.

Type
list of dict

error_bars

Error bars to show in figures.

Type
list of dict

class biomass.construction.template.search_param.**SearchParam**

Specify model parameters and/or initial values to optimize.

class biomass.construction.template.problem.**OptimizationProblem**

property bounds

Lower and upper bounds on independent variables.

objective(*indiv*, **args*)

Define an objective function to be minimized.

class biomass.construction.template.viz.**Visualization**

Plotting parameters for customizing figure properties.

cm

Choosing colormaps for `cmap`.

Type
matplotlib.colors.ListedColormap (default: `matplotlib.colormaps['tab10']`)

single_observable_options

Visualization options for time-course simulation (single-observable).

Type
list of *SingleObservable*

multiple_observables_options

Visualization options for time-course simulation (multi-observables).

Type
MultipleObservables

sensitivity_options

Visualization options for sensitivity analysis results.

Type
SensitivityOptions

static convert_species_name(*name*)

figure/sensitivity/initial_condition - Sensitivity for species with nonzero initial conditions.

Return type
`str`

static set_sensitivity_rcParams()

figure/sensitivity

Return type
`None`

static set_timecourse_rcParams()

figure/simulation

Return type
`None`

class biomass.construction.template.reaction_network.**ReactionNetwork**

Reaction indices grouped according to biological processes. This is used for sensitivity analysis (``target``='reaction').

static flux(*t*, *y*, *x*)

Flux vector.

5.4 BioMASS core functions (biomass.core)

BioMASS core functions

class biomass.core.**Model**(*pkg_name*)

Class for BioMASS model construction.

pkg_name

Path (dot-sepalated) to a biomass model directory. Use `__package__`.

Type
str

create(*show_info=False*)

Build a biomass model.

Parameters
show_info (bool (default: `False`)) – Set to `True` to print the information related to model size.

Returns
model – The BioMASS model object.

Return type
biomass.model_object.ModelObject

Examples

```
>>> from biomass import Model
>>> import your_model
>>> model = Model(your_model.__package__).create()
```

biomass.core.**create_model**(*pkg_name*, *show_info=False*)

Create a BioMASS model.

Parameters

- **pkg_name** (*str*) – Path (dot-sepalated) to a biomass model directory.
- **show_info** (bool (default: `False`)) – Set to `True` to print the information related to model size.

Returns
model – The BioMASS model object.

Return type
biomass.model_object.ModelObject

Examples

```
>>> from biomass import create_model
>>> import your_model
>>> model = create_model(your_model.__package__)
```

biomass.core.**optimize**(*model*, *x_id*, *, *disp_here=False*, *overwrite=False*, *optimizer_options=None*)

Estimate model parameters from experimental data.

Parameters

- **model** (ModelObject) – Model for parameter estimation.
- **x_id** (*int*) – Index of parameter set to estimate.
- **disp_here** (bool (default: False)) – Whether to show the evaluated *objective* at every iteration.
- **overwrite** (bool (default: False)) – If True, the directory (x_id/) will be overwritten.
- **optimizer_options** (*dict, optional*) – Keyword arguments to pass to scipy.optimize.differential_evolution. For details, please refer to https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html.

Return type

None

Examples

```
>>> from biomass import create_model, optimize
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
>>> optimize(model, x_id=1)
```

Notes

- Set simulation conditions and the corresponding experimental data in observable.py
- Define an objective function to be minimized (objective()) in problem.py
- Set lower/upper bounds of parameters to be estimated in search_param.py

biomass.core.**run_analysis**(*model*, *, *target*, *metric='integral'*, *create_metrics=None*, *style='barplot'*, *show_progress=True*, *clustermap_kws=None*, *cbar_ax_tick_params=None*, *options=None*)

Employ sensitivity analysis to identify critical parameters, species or reactions in the complex biological network.

The sensitivity S(y,x) was calculated according to the following equation: S(y,x) = d ln(yi) / d ln (xj), where yi is the signaling metric and xj is each nonzero species, parameter value or reaction rate.

Parameters

- **model** (ModelObject) – Model for sensitivity analysis.
- **target** (*Literal["reaction", "parameter", "initial_condition"]*) – Where to add a small perturbation to calculate sensitivity coefficients.

- **metric** (*str (default: 'integral')*) – A word to specify the signaling metric.
- **create_metrics** (*Dict[str, Callable[[np.ndarray], Union[int, float]]], optional*) – Create user-defined signaling metrics.
- **style** (*Literal["barplot", "heatmap"] (default: 'barplot')*) –
  - 'barplot'
  - 'heatmap'
- **show_progress** (bool (default: `True`)) – Set to `True` to show the progress indicator while calculating sensitivity coefficients.
- **clustermap_kws** (*dict, optional*) – Keyword arguments to pass to `seaborn.clustermap()` when style is 'heatmap'.
- **cbar_ax_tick_params** (*dict, optional*) – Keyword arguments to pass to `ax.tick_params()` of the color bar object when style is 'heatmap'.
- **options** (*dict, optional*) –
  - **show_indices**
    [bool (default: `True`)] (`target` == 'reaction') Set to `True` to put reaction index on each bar.
  - **excluded_params**
    [list of strings] (`target` == 'parameter') List of parameters which are not used for analysis.
  - **excluded_initials**
    [list of strings] (`target` == 'initial_condition') List of species which are not used for analysis.
  - **overwrite**
    [bool (default: `True`)] If `True`, the sensitivity_coefficients/{target}/{metric}.npy file will be overwritten.

Return type
`None`

Examples

```
>>> from biomass import create_model, run_analysis
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
```

Parameters

```
>>> run_analysis(
...     model,
...     target='parameter',
...     options = {
...         'excluded_params': [
...             'a', 'Vn', 'Vc', 'Ligand', 'EGF', 'HRG', 'no_ligand'
...         ]
...     }
... )
```

Initial condition

```
>>> run_analysis(model, target='initial_condition')
```

Reaction

```
>>> run_analysis(model, target='reaction')
```

biomass.core.**run_simulation**(*model*, *, *viz_type='original'*, *show_all=False*, *stdev=False*)

Simulate ODE model with estimated parameter values.

Parameters

- **model** (ModelObject) – Model for simulation.
- **viz_type** (*str*) –
  - **'average':**
    The average of simulation results with parameter sets in out/.
  - **'best':**
    The best simulation result in out/, simulation with *best_fit_param*.
  - **'original':**
    Simulation with the default parameters and initial values defined in set_model.py.
  - **'n(=1,2,...)':**
    Use the parameter set in out/n/.
  - **'experiment'**
    Draw the experimental data written in observable.py without simulation results.
- **show_all** (bool (default: False)) – Whether to show all simulation results.
- **stdev** (bool (default: False)) – If True, the standard deviation of simulated values will be shown (only available for 'average' visualization type).

Return type

None

Examples

```
>>> from biomass import create_model, run_simulation
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
>>> run_simulation(
...     model,
...     viz_type='average',
...     show_all=False,
...     stdev=True,
... )
```

5.5 Optimization results (biomass.results)

class biomass.result.**OptimizationResults**(*model*)

__post_init__()

Create optimization_results/ in the model folder.

Return type
None

dynamic_assessment(*include_original=False*)

Compute objective values using estimated parameters.

Parameters
include_original (bool (default: False)) – If True, an objective value simulated with original parameters will also be shown.

Return type
None

Examples

```
>>> from biomass import create_model, OptimizationResults
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
>>> res = OptimizationResults(model)
>>> res.dynamic_assessment()
```

Notes

Output:

- optimization_results/fitness_assessment.csv

savefig(*, *figsize=None*, *config=None*, *boxplot_kws=None*)

Visualize estimated parameter sets using seaborn.boxplot.

Parameters

- **figsize** (*Tuple[float, float], optional*) – Width, height in inches.
- **config** (*dict, optional*) – A dictionary object for setting matplotlib.rcParams.
- **boxplot_kws** (*dict, optional*) – Keyword arguments to pass to seaborn.boxplot.

Return type
None

Examples

```
>>> from biomass import create_model, OptimizationResults
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
>>> res = OptimizationResults(model)
>>> res.savefig(figsize=(16,5), boxplot_kws={"orient": "v"})
```

Notes

Output:

- optimization_results/estimated_parameter_sets.pdf

to_csv()

Save optimized parameters as CSV file format.

Return type
None

Examples

```
>>> from biomass import create_model, OptimizationResults
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
>>> res = OptimizationResults(model)
>>> res.to_csv()
```

Notes

Output:

- optimization_results/optimized_params.csv
- optimization_results/optimized_initials.csv

trace_obj(*, *config=None*, *xlabel='Iteration'*, *ylabel='Objective function value'*, *xticks=None*, *yticks=None*, *message_head='differential_evolution step'*, *sep=':'*, *prefix='='*)

Visualize objective function traces for different optimization runs.

Parameters

- **config** (*dict, optional*) – A dictionary object for setting *matplotlib.rcParams*.
- **xlabel** (*str (default: "Iteration")*) – The label for the x-axis.
- **ylabel** (*str (default: "Objective function value")*) – The label for the x-axis.
- **xticks** (*list, optional*) – The list of xtick locations.
- **yticks** (*list, optional*) – The list of ytick locations.

- **message_head** (*str (default: "differential_evolution step")*) – Beginning of the progress status message.
- **sep** (*str (default: ":")*) – Suffix for iteration number.
- **prefix** (*str (default: "=")*) – Prefix for objective function value.

Return type
None

Examples

```
>>> from biomass import create_model, OptimizationResults
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
>>> res = OptimizationResults(model)
>>> res.trace_obj()
```

Notes

Output:

- optimization_results/obj_func_traces.pdf

5.6 Parameter estimation (biomass.estimation.optimizer)

class biomass.estimation.**Optimizer**(*model*, *optimize*, *x_id*, *disp_here=False*, *overwrite=False*)

Use an external optimization method for parameterization of a mechanistic model.

model

The BioMASS model object.

Type
ModelObject

optimize

The optimizer, e.g., `scipy.optimize.differential_evolution()`.

Type
Callable

x_id

Index of parameter set to estimate.

Type
int

disp_here

Whether to show the evaluated *objective* at every iteration.

Type
bool (default: `False`)

overwrite

If `True`, the directory (`x_id/`) will be overwritten.

Type
bool (default: `False`)

Examples

```
>>> from scipy.optimize import differential_evolution
>>> from biomass import create_model
>>> from biomass.estimation import Optimizer
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
>>> param_idx = 1
>>> optimizer = Optimizer(model, differential_evolution, param_idx)
>>> def obj_fun(x):
...     "Objective function to be minimized."
...     return model.get_obj_val(x)
>>> res = optimizer.minimize(
...     obj_fun,
...     [(0, 1) for _ in range(len(model.problem.bounds))],
...     strategy="best1bin",
...     maxiter=50,
...     tol=1e-4,
...     mutation=0.1,
...     recombination=0.5,
...     disp=True,
...     polish=False,
...     workers=-1,
... )
```

differential_evolution step 1: f(x)= 5.19392

differential_evolution step 2: f(x)= 2.32477

differential_evolution step 3: f(x)= 1.93583

...

differential_evolution step 50: f(x)= 0.519774

```
>>> from biomass import run_simulation
>>> param_values = model.gene2val(res.x)
>>> optimizer.import_solution(param_values)
>>> run_simulation(model, viz_type=str(param_idx))
```

import_solution(*x*, *cleanup=True*)

Import the solution of the optimization to the model. The solution vector *x* will be saved to *path_to_model*/out/*x_id*/. Use `biomass.run_simulation` to visualize the optimization result.

Parameters

- **x** (*Union[np.ndarray, List[float]]*) – The solution of the optimization.
- **cleanup** (*bool (default: True)*) – If True (default), delete the temporary folder after the optimization is finished.

Return type
None

minimize(**args*, ***kwargs*)

Execute the external optimizer.

class biomass.estimation.**InitialPopulation**(*model*, *popsize=3*, *threshold=1000000000000.0*)

model

BioMASS model object.

Type
ModelObject

popsize

A multiplier for setting the total population size.

Type
int (default: 3)

threshold

Allowable error for generating initial population. Default value is 1e12 (numerically solvable).

Type
float (default: 1e12)

generate(*n_proc=1*, *progress=False*)

Return initial population for `optimizer_options['init']` in `biomass.optimize` function.

Parameters

- **n_proc** (*int (default: 1)*) – Number of processes to use (default: 1). Set to a larger number (e.g. the number of CPU cores available) for parallel execution of simulations.
- **progress** (bool (default: `False`)) – Whether the progress bar is animating or not.

Returns
population – Array specifying the initial population. The array should have shape (M, len(x)), where M is the total population size and len(x) is the number of parameters.

Return type
`numpy.ndarray`

Examples

```
>>> from biomass import create_model, optimize
>>> from biomass.estimation import InitialPopulation
>>> from biomass.models import copy_to_current
>>> copy_to_current("Nakakuki_Cell_2010")
>>> model = create_model("Nakakuki_Cell_2010")
>>> initpop = InitialPopulation(model).generate(n_proc=2, progress=True)
>>> optimize(model, x_id=1, optimizer_options={"init": initpop})
```

5.7 Visualization options (biomass.plotting)

Figure properties for visualizing time-course simulations and sensitivity analysis results.

Example code: Nakakuki_Cell_2010

class biomass.plotting.**MultipleObservables**(*cm*)

Visualization options for time-course simulation (multi-observables).

fname

Output file name.

Type
str

figsize

Figure size.

Type
tuple (default: (4, 3))

observables

Specify which observables to plot.

Type
list of strings

condition

Specify which simulation condition to be plotted.

Type
str, optional

xlim

Set the x limits of the current axes.

Type
tuple

xticks

Set the current tick locations of the x-axis.

Type
list (default: `None`)

xlabel

Set the label for the x-axis.

Type
str (default: 'Time')

ylim

Set the y limits of the current axes.

Type
tuple

yticks

Set the current tick locations of the y-axis.

Type
list (default: `None`)

`ylabel`

Set the label for the y-axis.

Type
str (default: ‘’)

`legend_kws`

Keyword arguments to pass to `matplotlib.pyplot.legend()`.

Type
dict, optional

`cmap`

Set colormap.

Type
list or tuple

`shape`

Set markers.

Type
list or tuple of strings (default: `matplotlib.lines.Line2D.filled_markers`)

`class biomass.plotting.SensitivityOptions(cm)`

Bar plot visualization options for sensitivity analysis results. When setting *style* to ‘heatmap’, change plotting options via `clustermap_kws`.

`figsize`

Figure size.

Type
tuple

`width`

The width(s) of the bars when choosing *style* == ‘barplot’.

Type
float

`legend_kws`

Keyword arguments to pass to `matplotlib.pyplot.legend()`.

Type
dict, optional

`cmap`

Set colormap.

Type
list or tuple

`class biomass.plotting.SingleObservable(cm, obs_name)`

Visualization options for time-course simulation (single-observable).

divided_by

Convert time unit. (e.g. sec -> min).

Type
int or float (default: 1)

figsize

Width, height in inches.

Type
tuple (default: (4, 3))

xlim

Set the x limits of the current axes.

Type
tuple

xticks

Set the current tick locations of the x-axis.

Type
list (default: `None`)

xlabel

Set the label for the x-axis.

Type
str (default: 'Time')

ylim

Set the y limits of the current axes.

Type
tuple

yticks

Set the current tick locations of the y-axis.

Type
list (default: `None`)

ylabel

Set the label for the y-axis.

Type
str (default: obs_name.replace('_', ' '))

exp_data

if False, experimental data will not be shown.

Type
bool (default: True)

legend_kws

Keyword arguments to pass to `matplotlib.pyplot.legend()`.

Type
dict, optional

`cmap`

Set colormap.

Type
list or tuple

`shape`

Set markers.

Type
list or tuple of strings (default: `matplotlib.lines.Line2D.filled_markers`)

`dont_show`

Set conditions you don't want to plot.

Type
list of strings

CHAPTER

SIX

REFERENCES

CHAPTER

SEVEN

CITING BIOMASS

If you use BioMASS in your research, please cite the following paper:

- Arakane, K., Imoto, H., Ormersbach, F. & Okada, M. Extending BioMASS to construct mathematical models from external knowledge. *Bioinformatics Advances* **4**, vbae042 (2024). https://doi.org/10.1093/bioadv/vbae042

In BibTeX format:

```
@article{10.1093/bioadv/vbae042,
  author  = {Arakane, Kiwamu and Imoto, Hiroaki and Ormersbach, Fabian and Okada, Mariko}
↪,
  title   = "{Extending BioMASS to construct mathematical models from external knowledge}
↪",
  journal = {Bioinformatics Advances},
  volume  = {4},
  number  = {1},
  pages   = {vbae042},
  year    = {2024},
  month   = {04},
  issn    = {2635-0041},
  doi     = {10.1093/bioadv/vbae042},
  url     = {https://doi.org/10.1093/bioadv/vbae042},
  eprint  = {https://academic.oup.com/bioinformaticsadvances/advance-article-pdf/doi/10.
↪1093/bioadv/vbae042/57163215/vbae042.pdf},
}
```

If you construct a mathematical model using *biomass.construction.text2model.Text2Model*, please also cite the Pasmopy paper:

- Imoto, H., Yamashiro, S. & Okada, M. A text-based computational framework for patient -specific modeling for classification of cancers. *iScience* **25**, 103944 (2022). https://doi.org/10.1016/j.isci.2022.103944

In BibTeX format:

```
@article{imoto2022text,
  title     = {A text-based computational framework for patient-specific modeling for␣
↪classification of cancers},
  author    = {Imoto, Hiroaki and Yamashiro, Sawa and Okada, Mariko},
  journal   = {iScience},
  volume    = {25},
  number    = {3},
  pages     = {103944},
  year      = {2022},
  publisher = {Elsevier},
```

(continues on next page)

(continued from previous page)

```
  doi       = {10.1016/ j.isci.2022.103944},
  url       = {https://www.cell.com/iscience/fulltext/S2589-0042(22)00214-0}
}
```

When presenting work that uses BioMASS, feel free to use the logo.

BIBLIOGRAPHY

[BHZ+16] Alexis R Barr, Frank S Heldt, Tongli Zhang, Chris Bakal, and Béla Novák. A dynamical framework for the all-or-none g1/s transition. *Cell systems*, 2(1):27–37, 2016. URL: https://www.sciencedirect.com/science/article/pii/S2405471216000028, doi:https://doi.org/10.1016/j.cels.2016.01.001.

[HMW+17] Helge Hass, Kristina Masson, Sibylle Wohlgemuth, Violette Paragas, John E Allen, Mark Sevecka, Emily Pace, Jens Timmer, Joerg Stelling, Gavin MacBeath, and others. Predicting ligand-dependent tumors from multi-dimensional signaling features. *NPJ systems biology and applications*, 3(1):1–15, 2017. doi:10.1038/s41540-017-0030-3.

[HBC+18] Frank S Heldt, Alexis R Barr, Sam Cooper, Chris Bakal, and Béla Novák. A comprehensive model for the proliferation–quiescence decision in response to endogenous dna damage in human cells. *Proceedings of the National Academy of Sciences*, 115(10):2532–2537, 2018. URL: https://www.pnas.org/doi/10.1073/pnas.1715345115, doi:https://doi.org/10.1073/pnas.1715345115.

[IYO22] Hiroaki Imoto, Sawa Yamashiro, and Mariko Okada. A text-based computational framework for patient-specific modeling for classification of cancers. *iScience*, 25(3):103944, 2022. URL: https://www.cell.com/iscience/fulltext/S2589-0042(22)00214-0, doi:10.1016/ j.isci.2022.103944.

[KHWB97] Boris N Kholodenko, Jan B Hoek, Hans V Westerhoff, and Guy C Brown. Quantification of information transfer via cellular signal transduction pathways. *FEBS letters*, 414(2):430–434, 1997. doi:https://doi.org/10.1016/S0014-5793(97)01018-1.

[Kho00] Boris N. Kholodenko. Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. *European Journal of Biochemistry*, 267(6):1583–1588, 2000. URL: https://febs.onlinelibrary.wiley.com/doi/abs/10.1046/j.1432-1327.2000.01197.x, arXiv:https://febs.onlinelibrary.wiley.com/doi/pdf/10.1046/j.1432-1327.2000.01197.x, doi:https://doi.org/10.1046/j.1432-1327.2000.01197.x.

[KNT+12] Hiroyuki Kubota, Rei Noguchi, Yu Toyoshima, Yu-ichi Ozaki, Shinsuke Uda, Kanako Watanabe, Wataru Ogawa, and Shinya Kuroda. Temporal coding of insulin action through multiplexing of the akt pathway. *Molecular Cell*, 46(6):820–832, 2012. URL: https://www.sciencedirect.com/science/article/pii/S1097276512003401, doi:https://doi.org/10.1016/j.molcel.2012.04.018.

[LG03] Jean-Christophe Leloup and Albert Goldbeter. Toward a detailed computational model for the mammalian circadian clock. *Proceedings of the National Academy of Sciences*, 100(12):7051–7056, 2003. URL: https://www.pnas.org/content/100/12/7051, arXiv:https://www.pnas.org/content/100/12/7051.full.pdf, doi:10.1073/pnas.1132112100.

[LSK+18] Philippe Lucarelli, Marcel Schilling, Clemens Kreutz, Artyom Vlasov, Martin E. Boehm, Nao Iwamoto, Bernhard Steiert, Susen Lattermann, Marvin Wäsch, Markus Stepath, Matthias S. Matter, Mathias Heikenwälder, Katrin Hoffmann, Daniela Deharde, Georg Damm, Daniel Seehofer, Maria Muciek, Norbert Gretz, Wolf D. Lehmann, Jens Timmer, and Ursula Klingmüller. Resolving the combinatorial complexity of smad protein complex formation and its link to gene ex-

pression. *Cell Systems*, 6(1):75–89.e11, 2018. URL: https://www.sciencedirect.com/science/article/pii/S2405471217305380, doi:https://doi.org/10.1016/j.cels.2017.11.010.

[MT08] Thomas Maiwald and Jens Timmer. Dynamical modeling and multi-experiment fitting with potterswheel. *Bioinformatics*, 24(18):2037–2043, 2008. doi:10.1093/bioinformatics/btn350.

[NBS+10] Takashi Nakakuki, Marc R. Birtwistle, Yuko Saeki, Noriko Yumoto, Kaori Ide, Takeshi Nagashima, Lutz Brusch, Babatunde A. Ogunnaike, Mariko Okada-Hatakeyama, and Boris N. Kholodenko. Ligand-specific c-fos expression emerges from the spatiotemporal control of erbb network dynamics. *Cell*, 141(5):884–896, 2010. URL: https://www.sciencedirect.com/science/article/pii/S0092867410003739, doi:https://doi.org/10.1016/j.cell.2010.03.054.

[OKH+18] Angela Oppelt, Daniel Kaschek, Suzanna Huppelschoten, Rowena Sison-Young, Fang Zhang, Marie Buck-Wiese, Franziska Herrmann, Sebastian Malkusch, Carmen L Krüger, Mara Meub, and others. Model-based identification of tnf-induced ikk-mediated and ib-mediated regulation of nfb signal transduction as a tool to quantify the impact of drug-induced liver injury compounds. *NPJ systems biology and applications*, 4(1):1–16, 2018. URL: https://www.nature.com/articles/s41540-018-0058-z, doi:https://doi.org/10.1038/s41540-018-0058-z.

PYTHON MODULE INDEX

b

INDEX

Symbols

B

C

D

E

P

R

S

T

U

V

W

X

Y