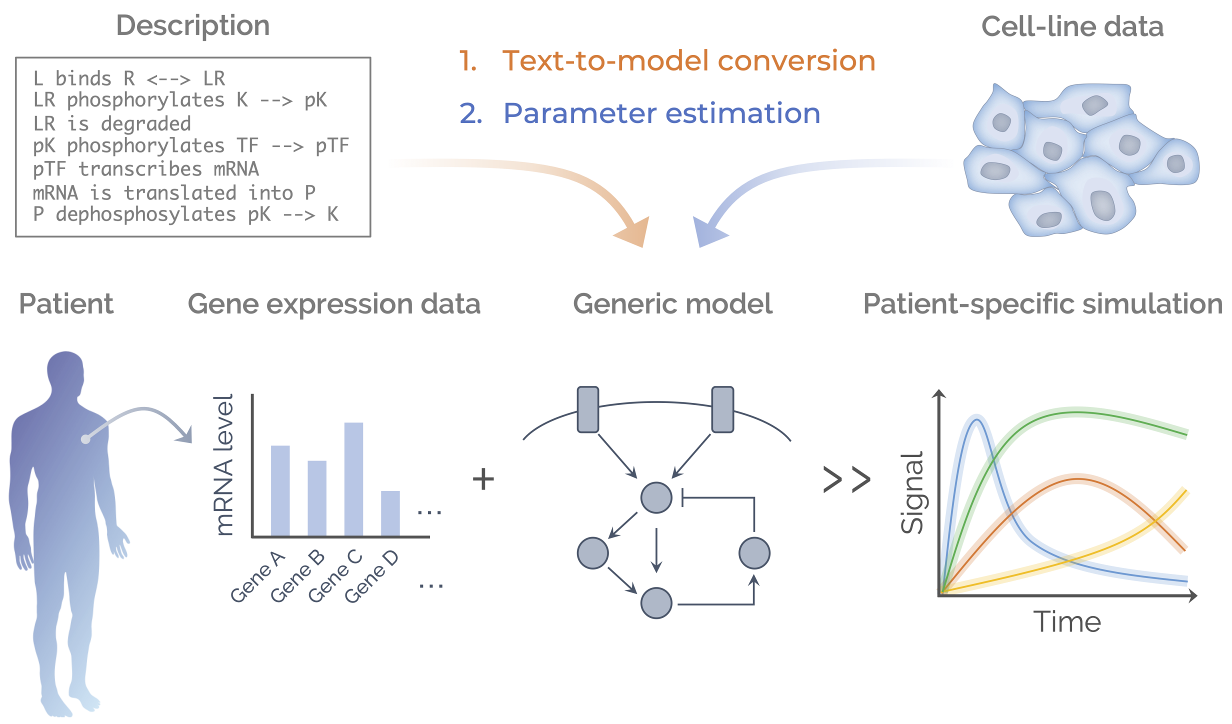
Pasmopy documentation

Release 0.5.0

Hiroaki Imoto

May 08, 2023

CONTENTS:



Pasmopy is an open-source Python package for the development of signaling pathway models that are individualized to patient-specific data. It includes modules for model construction, parameterization, *in silico* patient stratification, and more.

Source code: https://github.com/pasmopy/pasmopy

The open access publication describing Pasmopy is available here:

- 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

CHAPTER

ONE

ABOUT

1.1 What is Pasmopy?

Pasmopy is a scalable toolkit to identify prognostic factors for cancers based on intracellular signaling dynamics generated from personalized kinetic models. It is compatible with biomass and offers the following features:

- Construction of mechanistic models from text
- Personalization of the model using transcriptome data
- Prediction of patient outcome based on *in silico* signaling dynamics
- Sensitivity analysis for prediction of potential drug targets

1.2 License

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

1.3 Author

Hiroaki Imoto

1.4 Citation

If you use Pasmopy in a scientific publication, please cite the following papers:

- 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

```
@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},
```

(continues on next page)

(continued from previous page)

```
  doi = {10.1016/j.isci.2022.103944},
}
```

- 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). https://doi.org/10.1016/j.xpro.2022.101619

```
@article{imoto2022protocol,
  title = {Protocol for stratification of triple-negative breast cancer patients using in silico signaling dynamics},
  author = {Imoto, Hiroaki and Yamashiro, Sawa and Murakami, Ken and Okada, Mariko},
  journal = {STAR protocols},
  volume = {3},
  number = {3},
  pages = {101619},
  year = {2022},
  doi = {10.1016/j.xpro.2022.101619},
}
```

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

1.5 Contact

If you discovered an error or need help, please contact me via GitHub Issues. Please head over to GitHub Discussions if you have any questions or would like to start a new discussion. In either case, you can also always send me an email.

Any contributions to Pasmopy are more than welcome!

CHAPTER

TWO

INSTALLATION

Pasmopy requires Python 3.8+ to run.

2.1 Installing Pasmopy

2.1.1 PyPI

Install Pasmopy from PyPI using:

```
pip install pasmopy
```

2.1.2 Development version

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

```
pip install git+https://github.com/pasmopy/pasmopy
```

2.2 Upgrading

If you want to upgrade from a previous Pasmopy version, use:

```
pip install --upgrade pasmopy
```

CHAPTER

THREE

MODEL DEVELOPMENT

This section walk you through the creation of a mechanistic model from text.

3.1 How to use

`pasmopy.Text2Model` is a useful class to build an ordinary differential equation (ODE) model from a text file describing biochemical systems.

The text file you need to prepare can be divided into three parts:

1. *Reaction layer*
2. *Observable layer (Prefix: @obs)*
3. *Simulation layer (Prefix: @sim)*

Note: To write a comment, simply put the hash mark # before your desired comment.

 # This is a comment.

3.1.1 Reaction layer

description | parameters | initial conditions

In the reaction layer, you need to describe biochemical reactions. Each reaction described in the line number i will be converted into i^{th} rate equation. To specify parameters or initial conditions, you can put those information after `|`.

- If you don't specify parameters/initial_conditions, they are initialized to 1 and 0, respectively, and the parameter values will be estimated from experimental data.
- If you want to set a parameter value to 1 and don't want to estimate, you can add `const` prefix:

```
# The Hill coefficient is fixed to 1.
TF transcribes mRNA | const n=1
```

- You can impose parameter constraints by specifying line number in the **parameter** section.

```
# Nucleocytoplasmic Shuttling of DUSP
DUSPc translocates from the cytoplasm to the nucleus <--> DUSPn
pDUSPc translocates from the cytoplasm to the nucleus <--> pDUSPn |2|
```

In the example above, you can assume that import and export rates were identical for DUSP (line 2) and pDUSP (line 3).

- If the amount of a model species should be held fixed (never consumed) during simulation, you can add `fixed` prefix:

```
# [Ligand] will be held fixed to 10.0 during simulation
Ligand binds Receptor <--> LR | kf = 1e-6, kr = 1e-1 | fixed Ligand = 10.0
```

- To describe more complex rate equations, you can use `@rxn` prefix:

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

Please also refer to the following example: cfos_model

Note: The available rules can be found at https://biomass-core.readthedocs.io/en/latest/api/reaction_rules.html.

You can also supply your own terminology in a reaction rule via:

```python
from pasmopy import Text2Model

# Supply "releases" to the reaction rule: "dissociate"
mm_kinetics = Text2Model("michaelis_menten.txt")
mm_kinetics.register_word({"dissociate": ["releases"]})
# Now you can use "releases" in your text, e.g., 'ES releases E and P'
mm_kinetics.convert()
```

3.1.2 Observable layer (Prefix: `@obs`)

In the observable layer, you need to specify `biomass.observables`, which can correlate model simulations and experimental measurements. You can create an observable by using model parameters (`p`) and species (`u`). For example, if the total amount of SOS bound to EGFR should be the sum of RGS (EGFR-Grb2-SOS) and RShGS (EGFR-Shc-Grb2-SOS) complexes in your model, then you can write as follows:

```
@obs Total_SOS_bound_to_EGFR: u[RGS] + u[RShGS]
```

3.1.3 Simulation layer (Prefix: `@sim`)

In the simulation layer, you can set simulation conditions, e.g, the simulation time span, the initial concentration of model species, etc.

Example:

```
@sim tspan: [0, 120]
@sim unperturbed: init[EGF] = 0
@sim condition EGF20nM: init[EGF] = 680
@sim condition EGF2nM: init[EGF] = 68
```

- **tspan:**

  Two element vector `[t0, tf]` specifying the initial and final times.

- **unperturbed (optional):**

Description of the untreated condition to find the steady state.

- **condition (optional):**

  Experimental conditions. Use `p` and `init` to modify model parameters and initial conditions, respectively.

3.2 Examples

3.2.1 Michaelis-Menten enzyme kinetics

This example shows you how to build a simple Michaelis-Menten two-step enzyme catalysis model with Pasmopy.

$$E + S \quad ES \rightarrow E + P$$

An enzyme, E, binding to a substrate, S, to form a complex, ES, which in turn releases a product, P, regenerating the original enzyme.

1. Prepare a text file describing biochemical reactions (e.g., `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]
```

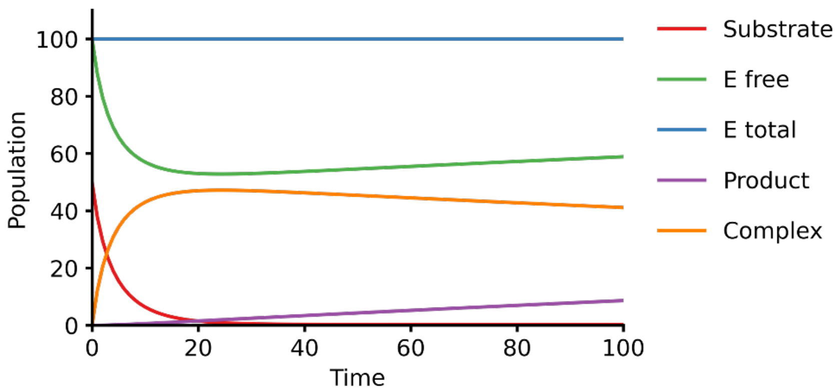
2. Convert the text into an executable model

```
$ python
```

```
>>> from pasmopy import Text2Model
>>> description = Text2Model("michaelis_menten.txt")
>>> description.convert()
Model information
-----------------
2 reactions
4 species
3 parameters
```

3. Run simulation

```
>>> from pasmopy import create_model, run_simulation
>>> model = create_model("michaelis_menten")
>>> run_simulation(model)
```



3.2.2 EGF signaling

Below is an example of Pasmopy in action to illustrate EGF signalling pathway.

Reference:

> Kholodenko, B. N., Demin, O. V, Moehren, G. & Hoek, J. B. Quantification of short term signaling by the epidermal growth factor receptor. *J. Biol. Chem.* **274**, 30169–30181 (1999). https://doi.org/10.1074/jbc.274.42.30169

1. Prepare a text describing EGF signaling in hepatocytes (`Kholodenko1999.txt`)

   You can learn how to build the model via `Text2Model` by comparing the desccription below and the network scheme in Figure 1 in the original paper. The description in the line *n* denotes the *n*-th reaction in the scheme.

```
1  EGF binds EGFR <--> Ra | kf=0.003, kr=0.06 | EGFR=100
2  Ra dimerizes <--> R2 | kf=0.01, kr=0.1
3  R2 is phosphorylated <--> RP | kf=1, kr=0.01
4  RP is dephosphorylated --> R2 | V=450, K=50
5  RP binds PLCg <--> RPL | kf=0.06, kr=0.2 | PLCg=105
6  RPL is phosphorylated <--> RPLP | kf=1, kr=0.05
7  RPLP is dissociated into RP and PLCgP | kf=0.3, kr=0.006
8  PLCgP is dephosphorylated --> PLCg | V=1, K=100
9  RP binds Grb2 <--> RG | kf=0.003, kr=0.05 | Grb2=85
10 RG binds SOS <--> RGS | kf=0.01, kr=0.06 | SOS=34
11 RGS is dissociated into RP and GS | kf=0.03, kr=4.5e-3
12 GS is dissociated into Grb2 and SOS | kf=1.5e-3, kr=1e-4
13 RP binds Shc <--> RSh | kf=0.09, kr=0.6 | Shc=150
14 RSh is phosphorylated <--> RShP | kf=6, kr=0.06
15 RShP is dissociated into ShP and RP | kf=0.3, kr=9e-4
16 ShP is dephosphorylated --> Shc | V=1.7, K=340
17 RShP binds Grb2 <--> RShG | kf=0.003, kr=0.1
18 RShG is dissociated into RP and ShG | kf=0.3, kr=9e-4
19 RShG binds SOS <--> RShGS | kf=0.01, kr=2.14e-2
20 RShGS is dissociated into ShGS and RP | kf=0.12, kr=2.4e-4
21 ShP binds Grb2 <--> ShG | kf=0.003, kr=0.1
22 ShG binds SOS <--> ShGS | kf=0.03, kr=0.064
23 ShGS is dissociated into ShP and GS | kf=0.1, kr=0.021
24 RShP binds GS <--> RShGS | kf=0.009, kr=4.29e-2
25 PLCgP is translocated to cytoskeletal or membrane structures <--> PLCgP_I | kf=1,
```

(continues on next page)

(continued from previous page)

```
kr=0.03

# observable layer
@obs Total_phosphorylated_Shc: u[RShP] + u[RShG] + u[RShGS] + u[ShP] + u[ShG] + u[ShGS]
@obs Total_Grb2_coprecipitated_with_Shc: u[RShG] + u[ShG] + u[RShGS] + u[ShGS]
@obs Total_phosphorylated_Shc_bound_to_EGFR: u[RShP] + u[RShG] + u[RShGS]
@obs Total_Grb2_bound_to_EGFR: u[RG] + u[RGS] + u[RShG] + u[RShGS]
@obs Total_SOS_bound_to_EGFR: u[RGS] + u[RShGS]
@obs ShGS_complex: u[ShGS]
@obs Total_phosphorylated_PLCg: u[RPLP] + u[PLCgP]

# simulation layer
@sim tspan: [0, 120]
@sim condition EGF20nM: init[EGF] = 680
@sim condition EGF2nM: init[EGF] = 68
@sim condition Absence_PLCgP_transloc: init[EGF] = 680; p[kf25] = 0; p[kr25] = 0
```

2. Convert the text into an executable model

```
$ python
```

To display thermodynamic restrictions, set `show_restrictions` to `True`.

```
>>> from pasmopy import Text2Model
>>> description = Text2Model("Kholodenko_JBC_1999.txt")
>>> description.convert(show_restrictions=True)
Model information
-----------------
25 reactions
23 species
50 parameters

Thermodynamic restrictions
--------------------------
{9, 12, 10, 11}
{15, 18, 21, 17}
{18, 22, 20, 19}
{17, 24, 12, 19}
{23, 24, 20, 15}
{23, 12, 22, 21}
```

The output of *Thermodynamic restrictions* shows the cyclic pathways in the biochemical reaction network. 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.

3. Run simulation

```
>>> from pasmopy import create_model, run_simulation
>>> model = create_model("Kholodenko_JBC_1999")
>>> run_simulation(model)
```

4. Plot simulation results

```
%matplotlib inline
import os
import matplotlib.pyplot as plt
import numpy as np


def plot_simulation_results(res):

    plt.figure(figsize=(9, 9))
    plt.rcParams['font.family'] = 'Arial'
    plt.rcParams['font.size'] = 12
    plt.rcParams['axes.linewidth'] = 1
    plt.rcParams['lines.linewidth'] = 2

    plt.subplots_adjust(wspace=0.5, hspace=0.4)

    plt.subplot(2, 2, 1)  # ---------------------------------------------------
    for obs_name, color in zip(
        ['Total_phosphorylated_Shc', 'Total_Grb2_coprecipitated_with_Shc'],
        ['g', 'm'],
    ):
        obs_idx = model.observables.index(obs_name)
        for j, condition in enumerate(['EGF20nM', 'EGF2nM']):
                plt.plot(
                    model.problem.t,
                    res[obs_idx, j],
                    color=color,
                    alpha=0.5 if condition == 'EGF2nM' else None,
                )
    plt.xlim(0, 120)
    plt.xticks([30*i for i in range(5)])
    plt.ylim(0, 150)
    plt.xlabel("TIME (s)")
    plt.ylabel("Protein concentrations (nM)")

    plt.subplot(2, 2, 2)  # ---------------------------------------------------
    for obs_name, color in zip(
        ['Total_phosphorylated_Shc_bound_to_EGFR', 'Total_Grb2_bound_to_EGFR'],
        ['g', 'm'],
    ):
        obs_idx = model.observables.index(obs_name)
        for j, condition in enumerate(['EGF20nM', 'EGF2nM']):
                plt.plot(
                    model.problem.t,
                    res[obs_idx, j],
                    color=color,
                    alpha=0.5 if condition == 'EGF2nM' else None,
                )
    plt.xlim(0, 120)
    plt.xticks([30*i for i in range(5)])
    plt.ylim(0, 25)
    plt.xlabel("TIME (s)")
    plt.ylabel("Protein concentrations (nM)")
```

(continues on next page)

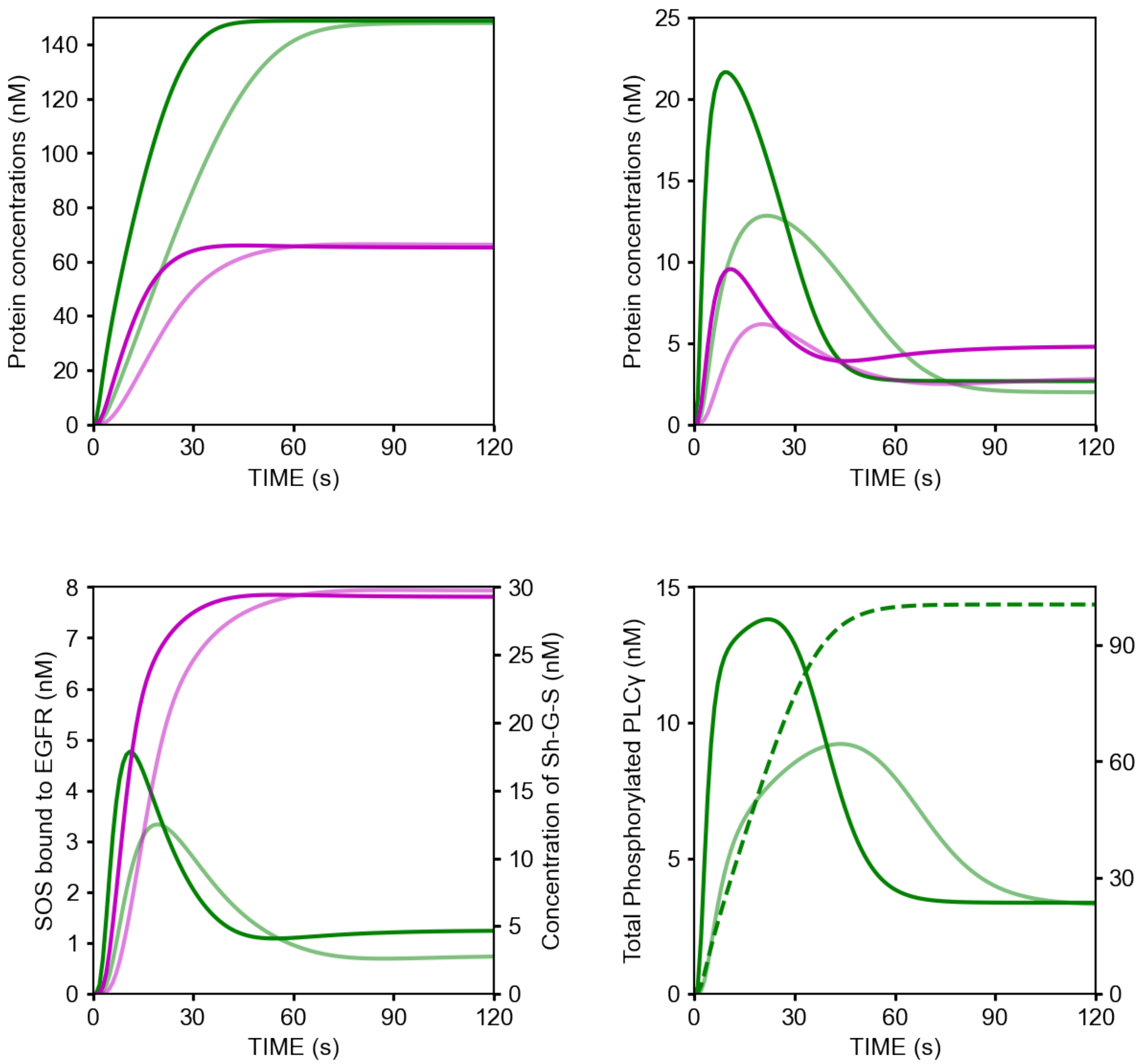
(continued from previous page)

```
ax1=plt.subplot(2, 2, 3)  # ---------------------------------------------------
ax2 = ax1.twinx()
for j, condition in enumerate(['EGF20nM', 'EGF2nM']):
   ax1.plot(
         model.problem.t,
         res[model.observables.index('Total_SOS_bound_to_EGFR'), j],
         color='g',
         alpha=0.5 if condition == 'EGF2nM' else None,
   )
   ax2.plot(
         model.problem.t,
         res[model.observables.index('ShGS_complex'), j],
         color='m',
         alpha=0.5 if condition == 'EGF2nM' else None,
   )
ax1.set_xlim(0, 120)
ax1.set_xticks([30*i for i in range(5)])
ax1.set_xlabel("TIME (s)")
ax1.set_ylim(0, 8)
ax2.set_ylim(0, 30)
ax1.set_ylabel("SOS bound to EGFR (nM)")
ax2.set_ylabel("Concentration of Sh-G-S (nM)")

ax1=plt.subplot(2, 2, 4)  # ---------------------------------------------------
ax2 = ax1.twinx()
obs_idx = model.observables.index('Total_phosphorylated_PLCg')
ax1.plot(
   model.problem.t,
   res[obs_idx, model.problem.conditions.index('EGF20nM')],
   'g',
)
ax1.plot(
   model.problem.t,
   res[obs_idx, model.problem.conditions.index('EGF2nM')],
   'g',
   alpha=0.5,
)
ax2.plot(
   model.problem.t,
   res[obs_idx, model.problem.conditions.index('Absence_PLCgP_transloc')],
   'g--',
)
ax1.set_xlim(0, 120)
ax1.set_xticks([30*i for i in range(5)])
ax1.set_ylim(0, 15)
ax1.set_yticks([5*i for i in range(4)])
ax1.set_xlabel("TIME (s)")
ax1.set_ylabel("Total Phosphorylated PLC (nM)")
ax2.set_ylim(0, 105)
ax2.set_yticks([30*i for i in range(4)])
```

(continues on next page)

(continued from previous page)

```
    plt.show()


if __name__ == '__main__':
    res = np.load(os.path.join(model.path, "simulation_data", "simulations_original.
↪npy"))
    plot_simulation_results(res)
```



3.2.3 c-Fos expression dynamics

Please refer to https://biomass-core.readthedocs.io/en/latest/tutorial/cfos.html.

CHAPTER

FOUR

PERSONALIZED SIGNALING MODELS

4.1 Breast cancer

The temporal activation dynamics of signaling pathways play important roles for cell fate decisions. Therefore, we hypothesized that signaling dynamics can be further utilized as prognostic biomarkers for human diseases. However, the majority of available data obtained from patients represent static snapshots taken at a single point in time, and not time-resolved dynamics. To overcome this problem, we developed a Patient-Specific Modeling in Python (Pasmopy), an open-source package for the development of dynamic pathway models that are individualized to patient-specific data.

Using Pasmopy, we built a mechanistic model of ErbB receptor signaling network, trained with protein quantification data obtained from cultured cell lines, and performed *in silico* simulation of the pathway activities on breast cancer patients using The Cancer Genome Atlas (TCGA) transcriptome datasets. The temporal activation dynamics of Akt, extracellular signal-regulated kinase (ERK), and c-Myc in each patient were able to accurately predict the difference in prognosis and sensitivity to kinase inhibitors in triple-negative breast cancer (TNBC).

4.2 Protocol

This protocol describes in detail the step-by-step method for model construction of the ErbB signaling network, parameterization of the models, integration of transcriptomic data, and stratification of breast cancer patients based on signaling dynamics.

- **Paper:** https://doi.org/10.1016/j.xpro.2022.101619
- **Code:** https://github.com/pasmopy/breast_cancer

CHAPTER

FIVE

PASMOPY MODULES REFERENCE

5.1 Patient-specific modeling (`pasmopy.patient_model`)

class `pasmopy.patient_model.`**`PatientModelSimulations`**(*path_to_models*, *patients*, *biomass_kws=None*)

Run simulations of patient-specific models.

`biomass_kws`

Keyword arguments to pass to `biomass.run_simulation`.

Type

dict, optional

`response_characteristics`

A dictionary containing functions to extract dynamic response characteristics from time-course simulations.

Type

dict[str, Callable[[1d-array], int ot float]]

`run`(*n_proc=None*, *context='spawn'*, *progress=True*)

Run simulations of multiple patient-specific models in parallel.

Parameters

- **`n_proc`** (*`int, optional`*) – The number of worker processes to use.
- **`context`** (*`Literal["spawn", "fork", "forkserver"] (default: "spawn")`*) – The context used for starting the worker processes.
- **`progress`** (bool (default: `True`)) – If `True`, the progress indicator will be shown.

Return type

`None`

`subtyping`(*fname*, *dynamical_features*, *normalization=None*, *progress=True*, ***, *clustermap_kws=None*)

Classify patients based on dynamic characteristics extracted from simulation results.

Parameters

- **`fname`** (str, path-like or `None`) – The clustermap is saved as fname if it is not `None`.
- **`dynamical_features`** (*`Dict[str, Dict[str, List[str]]]`*) – `{"observable": {"condition": ["metric", ...], ...}, ...}`. Characteristics in the signaling dynamics used for classification.
- **`normalization`** (dict, optional (default: `None`)) –

- **'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.

- **progress** (bool (default: True)) – If True, the progress indicator will be shown.
- **clustermap_kws** (*dict, optional*) – Keyword arguments to pass to seaborn.clustermap().

Examples

Subtype classification

```
>>> with open("models/breast/sample_names.txt", mode="r") as f:
...     TCGA_ID = f.read().splitlines()
>>> from pasmopy import PatientModelSimulations
>>> simulations = PatientModelSimulations("models.breast", TCGA_ID)
>>> simulations.subtyping(
...     "subtype_classification.pdf",
...     {
...         "Phosphorylated_Akt": {"EGF": ["max"], "HRG": ["max"]},
...         "Phosphorylated_ERK": {"EGF": ["max"], "HRG": ["max"]},
...         "Phosphorylated_c-Myc": {"EGF": ["max"], "HRG": ["max"]},
...     },
...     {
...         "Phosphorylated_Akt": {"timepoint": None, "condition": ["EGF", "HRG
↪"]},
...         "Phosphorylated_ERK": {"timepoint": None, "condition": ["EGF", "HRG
↪"]},
...         "Phosphorylated_c-Myc": {"timepoint": None, "condition": ["EGF", "HRG
↪"]},
...     },
...     clustermap_kws={"figsize": (9, 12)}
... )
```

Add new characteristics

```
>>> import numpy as np
>>> def get_droprate(time_course: np.ndarray) -> float:
...     return - (time_course[-1] - np.max(time_course)) / (len(time_course) -␣
↪np.argmax(time_course))
>>> simulations.response_characteristics["droprate"] = get_droprate
```

class pasmopy.patient_model.**PatientModelAnalyses**(*path_to_models*, *patients*, *biomass_kws=None*)

Run analyses of patient-specific models.

biomass_kws

Keyword arguments to pass to biomass.run_analysis.

Type

dict, optional

run(*n_proc=None*, *context='spawn'*, *progress=True*)

Run analyses of multiple patient-specific models in parallel.

Parameters

- **n_proc** (*int, optional*) – The number of worker processes to use.
- **context** (*Literal["spawn", "fork", "forkserver"] (default: "spawn")*) – The context used for starting the worker processes.
- **progress** (bool (default: `True`)) – If `True`, the progress indicator will be shown.

Return type

`None`

5.2 Preprocessing (pasmopy.preprocessing)

class pasmopy.preprocessing.weighting_factors.**WeightingFactors**(*model*, *gene_expression*)

Prepare for adding information about gene expression data to model.

model

BioMASS model object.

Type

`biomass.model_object.ModelObject`

gene_expression

Pairs of proteins and their related genes.

Type

dict

weighting_factors

List of weighting factors.

Type

list of strings

prefix

Prefix of weighting factors on gene expression levels.

Type

str (default: "**w_**")

indentation

How many spaces as indentation.

Type

str (default: 4 spaces)

Examples

```
>>> import erbb_network
>>> model = Model(erbb_network.__package__).create()
>>> gene_expression = {
...     "ErbB1": ["EGFR"],
...     "ErbB2": ["ERBB2"],
...     "ErbB3": ["ERBB3"],
...     "ErbB4": ["ERBB4"],
...     "Grb2": ["GRB2"],
...     "Shc": ["SHC1", "SHC2", "SHC3", "SHC4"],
...     "RasGAP": ["RASA1", "RASA2", "RASA3"],
...     "PI3K": ["PIK3CA", "PIK3CB", "PIK3CD", "PIK3CG"],
...     "PTEN": ["PTEN"],
...     "SOS": ["SOS1", "SOS2"],
...     "Gab1": ["GAB1"],
...     "RasGDP": ["HRAS", "KRAS", "NRAS"],
...     "Raf": ["ARAF", "BRAF", "RAF1"],
...     "MEK": ["MAP2K1", "MAP2K2"],
...     "ERK": ["MAPK1", "MAPK3"],
...     "Akt": ["AKT1", "AKT2"],
...     "PTP1B": ["PTPN1"],
...     "GSK3b": ["GSK3B"],
...     "DUSP": ["DUSP5", "DUSP6", "DUSP7"],
...     "cMyc": ["MYC"],
... }
>>> weighting_factors = WeightingFactors(model, gene_expression)
>>> weighting_factors.add_to_params()
>>> weighting_factors.set_search_bounds()
```

add_to_params()

Add weighting factors to model parameters.

Return type

None

set_search_bounds(*lb=0.01*, *ub=100.0*)

Set search bounds for weighting factors.

Parameters

- **lb** (*float (default: 0.01)*) – Lower bound.
- **ub** (*float (default: 100.0)*) – Upper bound.

Return type

None

5.3 Individualization of mechanistic models (pasmopy.individualization)

class pasmopy.individualization.**Individualization**(*parameters*, *species*, *transcriptomic_data*, *gene_expression*, *read_csv_kws=None*)

Individualize a mechanistic model by incorporating gene expression levels.

parameters

List of model parameters.

Type
List[str]

species

List of model species.

Type
List[str]

transcriptomic_data

Path to normalized gene expression data (CSV-formatted), e.g., (1) RLE-normalized and (2) post-ComBat TPM values. Below is an example of data table.

Description	patient1	patient2	patient3	…
gene1	value1,1	value1,2	value1,3	…
gene2	value2,1	value2,2	value2,3	…
gene3	value3,1	value3,2	value3,3	…
…	…	…	…	…

Type
str

gene_expression

Pairs of proteins and their related genes.

Type
Dict[str, List[str]]

read_csv_kws

Keyword arguments to pass to `pandas.read_csv`.

Type
dict, optional

prefix

Prefix of weighting factors on gene expression levels.

Type
str (default: "**w_**")

Examples

search_param.py

```
import os
import numpy as np
from pasmopy import Individualization
from . import __path__
from .name2idx import C, V
from .ode import initial_values, param_values

incorporating_gene_expression_levels = Individualization(
    parameters=C.NAMES,
    species=V.NAMES,
    transcriptomic_data=os.path.join("transcriptomic_data", "TPM_RLE_postComBat_
↪BRCA_BREAST.csv"),
    gene_expression={
        "ErbB1": ["EGFR"],
        "ErbB2": ["ERBB2"],
        "ErbB3": ["ERBB3"],
        "ErbB4": ["ERBB4"],
        "Grb2": ["GRB2"],
        "Shc": ["SHC1", "SHC2", "SHC3", "SHC4"],
        "RasGAP": ["RASA1", "RASA2", "RASA3"],
        "PI3K": ["PIK3CA", "PIK3CB", "PIK3CD", "PIK3CG"],
        "PTEN": ["PTEN"],
        "SOS": ["SOS1", "SOS2"],
        "Gab1": ["GAB1"],
        "RasGDP": ["HRAS", "KRAS", "NRAS"],
        "Raf": ["ARAF", "BRAF", "RAF1"],
        "MEK": ["MAP2K1", "MAP2K2"],
        "ERK": ["MAPK1", "MAPK3"],
        "Akt": ["AKT1", "AKT2"],
        "PTP1B": ["PTPN1"],
        "GSK3b": ["GSK3B"],
        "DUSP": ["DUSP5", "DUSP6", "DUSP7"],
        "cMyc": ["MYC"],
    },
    read_csv_kws={"index_col": "Description"}
)

...

def update(self, indiv):
    x = param_values()
    y0 = initial_values()
    for i, j in enumerate(self.idx_params):
        x[j] = indiv[i]
    for i, j in enumerate(self.idx_initials):
        y0[j] = indiv[i + len(self.idx_params)]
    # As maximal transcription rate
    x[C.V291] = incorporating_gene_expression_levels.as_reaction_rate(
        __path__[0].split(os.sep)[-1], x, "V291", "DUSP"
```

(continues on next page)

(continued from previous page)

```
    )
    x[C.V310] = incorporating_gene_expression_levels.as_reaction_rate(
        __path__[0].split(os.sep)[-1], x, "V310", "cMyc"
    )
    # As initial conditions
    y0 = incorporating_gene_expression_levels.as_initial_conditions(
        __path__[0].split(os.sep)[-1], x, y0
    )

    ...
```

as_initial_conditions(*id*, *x*, *y0*)

Gene expression levels are incorporated as initial conditions.

Parameters

- **id** (*str*) – CCLE_ID or TCGA_ID.
- **x** (*List[float]*) – List of parameter values.
- **y0** (*List[float]*) – List of initial values.

Returns
y0 (individualized) – Cell-line- or patient-specific initial conditions.

Return type
List[float]

as_reaction_rate(*id*, *x*, *param_name*, *protein*)

Gene expression levels are incorporated as a reaction rate.

Parameters

- **id** (*str*) – CCLE_ID or TCGA_ID.
- **x** (*List[float]*) – List of parameter values.
- **param_name** (*str*) – Name of the parameter incorporating gene_expression_data.
- **protein** (*str*) – Protein involved in the reaction.

Returns
param_value

Return type
float

5.4 Drug-response data analysis (pasmopy.validation)

class pasmopy.validation.**CancerCellLineEncyclopedia**

Cancer Cell Line Encyclopedia (CCLE) https://portals.broadinstitute.org/ccle

drug_alias

Other drug names.

Type
dict

_drug_response_data

Pharmacologic profiles for 24 anticancer drugs across 504 cell lines.

Type
pandas.DataFrame

save_activity_area(*expression_ratio*, *classifier*, *drug*, *, *labels*, *config=None*)

Save ActArea.

Return type
None

Examples

```
>>> from pasmopy.validation import CancerCellLineEncyclopedia
>>> ErbB_expression_ratio = pd.read_csv(
...     "https://raw.githubusercontent.com/pasmopy/breast_cancer/master/drug_
↪response/data/ErbB_expression_ratio.csv",
...     index_col=0,
... )
>>> ccle = CancerCellLineEncyclopedia()
>>> for drug in ["Erlotinib", "Lapatinib"]:
...     ccle.save_activity_area(
...         ErbB_expression_ratio,
...         {"value": ["high", "low"]},
...         drug,
...         labels=["EGFR high", "EGFR low"],
...     )
```

save_dose_response_curve(*expression_ratio*, *classifier*, *drug*, *, *labels*, *config=None*, *show_individual=False*)

Save dose-response curves.

Return type
None

Examples

```
>>> from pasmopy.validation import CancerCellLineEncyclopedia
>>> ErbB_expression_ratio = pd.read_csv(
...     "https://raw.githubusercontent.com/pasmopy/breast_cancer/master/drug_
↪response/data/ErbB_expression_ratio.csv",
...     index_col=0,
... )
>>> ccle = CancerCellLineEncyclopedia()
>>> for drug in ["Erlotinib", "Lapatinib"]:
...     ccle.save_dose_response_curve(
...         ErbB_expression_ratio,
...         {"value": ["high", "low"]},
...         drug,
...         labels=["EGFR high", "EGFR low"],
...     )
```

INDEX

T

W