EARM Documentation

Release 2.0

Carlos Lopez, Jeremy Muhlich, John Bachman, Peter Sorger

January 27, 2017

Contents

1 Introduction **3**
- 1.1 What is EARM? . . . 3
- 1.2 Goals . . . 3
- 1.3 Installation . . . 4
- 1.4 How to use the documentation . . . 4
- 1.5 Staying current . . . 4
- 1.6 Contributing . . . 4

2 Overview of the models **7**
- 2.1 The models in EARM . . . 9
- 2.2 How the model code is organized . . . 9
- 2.3 MOMP module "boundaries" . . . 22
- 2.4 How to use the models . . . 22
- 2.5 Parameter values . . . 23
- 2.6 The code is meant to be read! . . . 23

3 Implementation details (code and documentation) **25**

4 Tests **27**
- 4.1 Basic tests of all models . . . 27
- 4.2 Lopez model tests . . . 27
- 4.3 Albeck model tests . . . 28
- 4.4 Shen/Howells model tests . . . 30

5 References **35**

6 Indices and tables **37**

Bibliography **39**

Python Module Index **41**

Python Module Index **43**

The Extrinsic Apoptosis Reaction Model, or EARM, is a family of models of the extrinsic apoptosis pathway written using the Python software framework PySB. Model variants focus on exploring alternative hypotheses for the mechanism of Mitochondrial Outer Membrane Permeabilization (MOMP), controlled by the Bcl-2 family of proteins.

Contents:

CHAPTER 1

Introduction

1.1 What is EARM?

The Extrinsic Apoptosis Reaction Model (EARM), is **a family of novel and previously published models of extrinsic apoptosis, focusing on variant hypotheses for how the Bcl-2 protein family regulates mitochondrial outer membrane permeabilization (MOMP).** All models are written using the Python software framework PySB.

The models in this Python package implement 15 different mechanistic hypotheses for MOMP regulation by the Bcl-2 protein family, including

- 3 novel MOMP models of expanded scope that are unique to EARM and are described in Lopez et al. (2013), (*[Lopez2013]*)
- 5 MOMP models with hypothetical reaction topologies previously described in Albeck et al. (2008) PLoS Biology, Figure 11 (*[Albeck2008]*)
- 6 MOMP models drawn from three papers from the research group of Pingping Shen (*[Chen2007biophysj]*, *[Chen2007febs]*, *[Cui2008]*)
- 1 MOMP model focusing on Bad phosphorylation drawn from Howells et al. (2011), J. Theor. Biol. (*[Howells2011]*)

Moreover, for each of these there is a MOMP-only ("mito") version of the model and a full-apoptosis version of the model, in which the MOMP model is linked to the upstream and downstream pathways of extrinsic apoptosis from the previously published EARM 1.0 from *[Albeck2008]*. **This gives a total of 30 different models.**

1.2 Goals

Our goals in creating EARM were to:

- Create a newly updated model of the extrinsic apoptosis pathway incorporating current knowledge of Bcl-2 interactions and MOMP mechanism
- Demonstrate the use of PySB for transparent and reusable model development
- Make previous apoptosis modeling work from the community available for reuse in a common format
- Explore approaches for working with, and discriminating among, multiple hypotheses for biological mechanisms
- Engage the modeling community in a pilot effort to use software engineering tools and approaches for incremental, collaborative model development.

1.3 Installation

EARM requires PySB, but itself is pure Python (i.e., does not require installation or compilation of dependencies written in languages other than Python). PySB is available at http://pysb.org and http://github.com/pysb/pysb. EARM has its own website at http://sorgerlab.github.com/earm.

Note: The PySB VM includes EARM!

If you are using PySB via the downloadable virtual machine, EARM is already installed.

You can get the latest version of EARM from GitHub at http://github.com/sorgerlab/earm. If you are a Git user you can get the source code with:

```
git clone https://github.com/sorgerlab/earm.git
```

which will create a directory called `earm` containing all EARM files.

If you are not a Git user you can download a ZIP file at https://github.com/sorgerlab/earm/zipball/master.

Once you have downloaded the EARM source code, install EARM by switching to the top-level EARM source directory and running:

```
python setup.py install
```

1.4 How to use the documentation

In addition to this introduction, a general description of how the model implementations are organized into Python modules can be found in the Overview of the models.

More detailed descriptions of each model, with links to the Python source code that serves as the actual model specification, can be found at Implementation details (code and documentation).

In addition to the model specifications, EARM also contains a set of tests that can be run to ensure that the models can be successfully loaded and that the PySB re-implementations of previously published models exactly match their prior implementations (specified as systems of ordinary differential equations). Documentation and code for these tests can be found at Tests.

1.5 Staying current

This package is subject to change. In particular it may be refactored to stay current with ongoing updates to PySB, or to add new models and fix errors in existing ones. To be notified of updates, follow EARM on GitHub at http://github.com/sorgerlab/earm. If you use Git, you can get updates by going to the EARM source directory and running `git pull`. After getting the update source code, you will need to re-install by running `python setup.py install`.

1.6 Contributing

If you find bugs or errors, please notify us by submitting issues to the EARM issues page on GitHub.

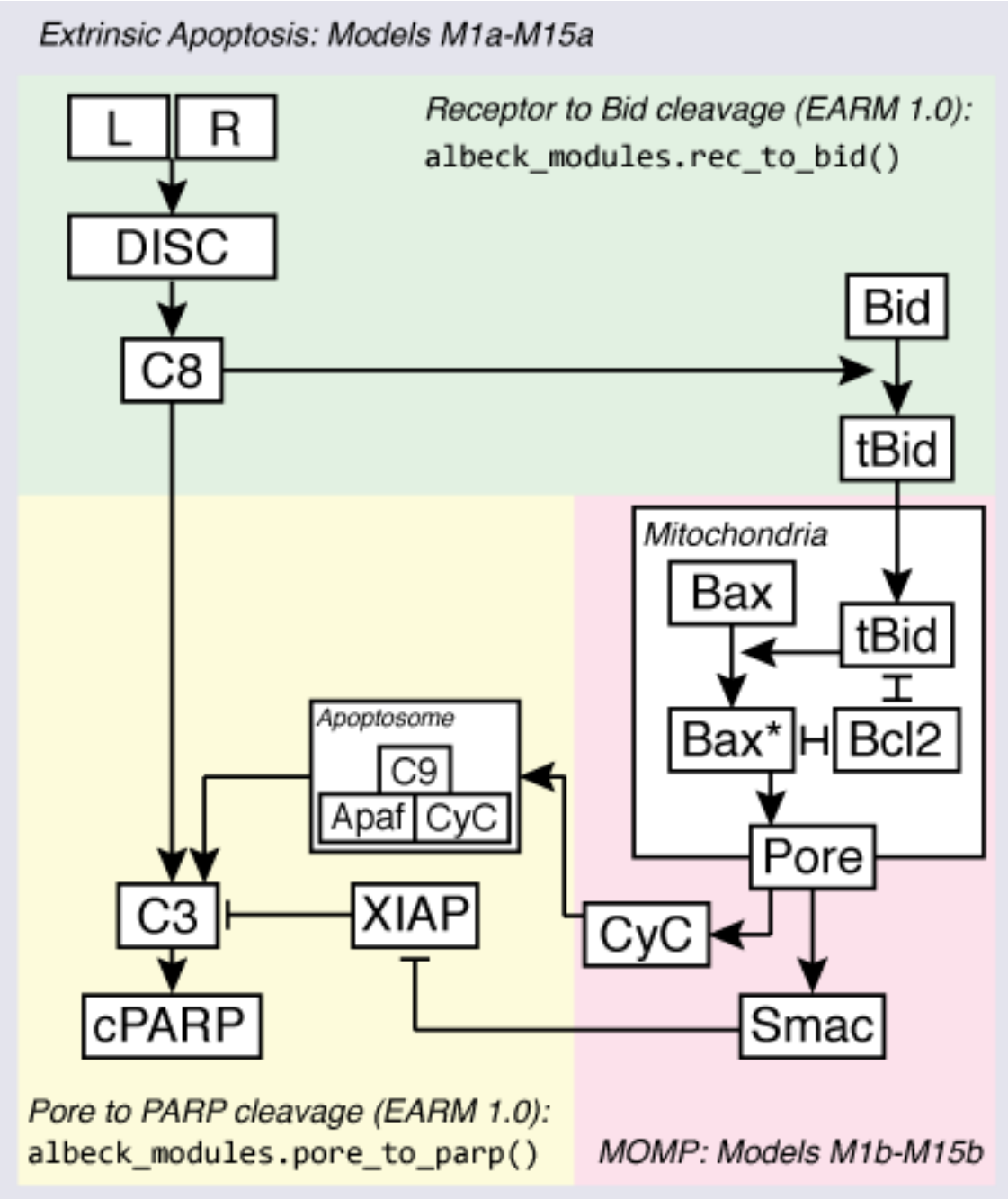
Also, we welcome contributions from other researchers studying extrinsic apoptosis! To contribute, fork EARM on GitHub and make your revisions and additions. To request that your contributions be incorporated in the EARM repository, submit a pull request.

CHAPTER 2

Overview of the models

As the following figure shows, the extrinsic apoptosis pathway can be considered to have roughly three major regulatory focal points, or "modules": *[Albeck2008]*

- Activation of initiator caspases of Bid by receptor ligation ("Receptor to Bid" in the diagram)
- Mitochondrial outer membrane permeabilization (MOMP)
- Effector caspase activation and substrate cleavage ("Pore to PARP cleavage" in the diagram)



The models in EARM are focused on exploring alternate hypotheses for the regulation of MOMP by Bcl-2 family proteins, both as an isolated module and in the overall context of the extrinsic apoptosis pathway.

Each hypothesis for MOMP regulation by Bcl-2 proteins in EARM thus has two models associated with it: a **"MOMP-only"** model that can be used to study the properties of a Bcl-2 reaction topology as an isolated module, and a **"full apoptosis"** form, in which the different MOMP models are embedded in the full extrinsic apoptosis pathway which begins with TRAIL or FasL stimulation. EARM contains 15 alternative Bcl-2 topologies for MOMP; thus there are 15 Bcl-2 topologies x two versions = 30 models, enumerated *M1a, M1b, ..., M15a, M15b*. **The "a" suffix denotes the full apoptosis model for a given Bcl-2 topology, while the "b" suffix denotes the corresponding MOMP-only model.**

For the full apoptosis models, the upstream and downstream pathway components and reaction topologies are re-used from the previously published EARM 1.0 *[Albeck2008]*.

2.1 The models in EARM

Below is a list of the 15 alternative Bcl-2 reaction topologies incorporated into EARM. More detailed descriptions of each model, along with the source code, are found in Implementation details (code and documentation).

- M1a/b: EARM 2.0, Embedded *[Lopez2013]*
- M2a/b: EARM 2.0, Direct *[Lopez2013]*
- M3a/b: EARM 2.0, Indirect *[Lopez2013]*
- M4a/b: "Minimal Model" (Figure 11b) from Albeck et al. (2008) *[Albeck2008]*
- M5a/b: "Model B + Bax multimerization" (Figure 11c) from Albeck et al. (2008) *[Albeck2008]*
- M6a/b: "Model C + mitochondrial transport" (Figure 11d) from Albeck et al. (2008) *[Albeck2008]*
- M7a/b: "Current model" (Figure 11e) from Albeck et al. (2008) *[Albeck2008]*
- M8a/b: "Current model + cooperativity" (Figure 11f) from Albeck et al. (2008) *[Albeck2008]*
- M9a/b: Deterministic model from Chen et al. (2007), Biophysical Journal *[Chen2007biophysj]*
- M10a/b: Indirect model from Chen et al. (2007), FEBS Letters *[Chen2007febs]*
- M11a/b: Direct model from Chen et al. (2007), FEBS Letters *[Chen2007febs]*
- M12a/b: Direct model from Cui et al. (2008) *[Cui2008]*
- M13a/b: Direct model 1 from Cui et al. (2008) *[Cui2008]*
- M14a/b: Direct model 2 from Cui et al. (2008) *[Cui2008]*
- M15a/b: Model incorporating Bad phosphorylation from Howells et al. (2011) *[Howells2011]*

2.2 How the model code is organized

For each of the 30 models, there is a corresponding `.py` file containing the model definition. This way any model can be imported using the straightforward syntax (for example, for model M1a):

```
from earm.lopez_embedded import model
```

However, the Python files for each individual model in general do not contain much code–they mainly call functions from other modules. For example, here is the source code for the file `earm/lopez_embedded.py`, which implements model M1a:

```
"""
Model M1a: Extrinsic apoptosis model with expanded, "embedded together"
model of MOMP.
"""

from pysb import *
from earm import shared
from earm import lopez_modules
from earm import albeck_modules

Model()
```

```
# Declare monomers
albeck_modules.ligand_to_c8_monomers()
lopez_modules.momp_monomers()
albeck_modules.apaf1_to_parp_monomers()

# Generate the upstream and downstream sections
albeck_modules.rec_to_bid()
albeck_modules.pore_to_parp()

# The specific MOMP model to use
lopez_modules.embedded()

# Declare observables
shared.observables()
```

As this example shows, the model file calls a series of macros that declare the monomers (*earm.albeck_modules.ligand_to_c8_monomers()*, *earm.lopez_modules.momp_monomers()*, and *earm.albeck_modules.apaf1_to_parp_monomers()*), then calls the macros implementing the upstream and downstream pathway elements (*earm.albeck_modules.rec_to_bid()* and *earm.albeck_modules.pore_to_parp()*), and finally calls the macro for the specific Bcl-2 topology involved: *earm.lopez_modules.embedded()*. Since the observables for all of the full apoptosis model variants is the same, these are declared in the final macro that is called, *earm.shared.observables()*.

All of the model `.py` files follow this pattern, calling a handful of macros to declare monomers and observables and select implementations of different pathway modules.

Note that the `.py` model files for the full-apoptosis models (M1a - M15a) are found in the top-level `earm` module, but the files for the MOMP-only models (M1b - M15b) are found in the submodule `earm.mito`.

The documentation for all model files (with links to source code) can be found at the following links:

2.2.1 Full extrinsic apoptosis model files (M1a-M15a)

Lopez Models (M1a - M3a)

[source]

Model M1a: Extrinsic apoptosis model with expanded, "embedded together" model of MOMP. [source]

Model M2a: Extrinsic apoptosis model with expanded "direct" model of MOMP. [source]

Model M3a: Extrinsic apoptosis model with expanded "indirect" model of MOMP.

Albeck Models (M4a - M8a)

[source]

Model M4a: Extrinsic apoptosis model incorporating the MOMP model "Minimal Model" (Figure 11b) from *[Albeck2008]*. [source]

Model M5a: Extrinsic apoptosis model incorporating the MOMP model "Model B + Bax multimerization" (Figure 11c) from *[Albeck2008]*. [source]

Model M6a: Extrinsic apoptosis model incorporating the MOMP model "Model C + mitochondrial transport" (Figure 11d) from *[Albeck2008]*. [source]

Model M7a: Extrinsic apoptosis model incorporating the MOMP model "Current model" (Figure 11e) from *[Albeck2008]*. [source]

Model M8a: Extrinsic apoptosis model incorporating the MOMP model "Current model + cooperativity" (Figure 11f) from *[Albeck2008]*.

Shen Models (M9a - M15a)

[source]

Model M9a: Extrinsic apoptosis model incorporating the MOMP model from *[Chen2007biophysj]*. [source]

Model M10a: Extrinsic apoptosis model incorporating the "Direct" MOMP model from *[Chen2007febs]*. [source]

Model M11a: Extrinsic apoptosis model incorporating the "Indirect" MOMP model from *[Chen2007febs]*. [source]

Model M12a: Extrinsic apoptosis model incorporating the "Direct" MOMP model from *[Cui2008]*. [source]

Model M13a: Extrinsic apoptosis model incorporating the "Direct 1" MOMP model from *[Cui2008]*. [source]

Model M14a: Extrinsic apoptosis model incorporating the "Direct 2" MOMP model from *[Cui2008]*. [source]

Model M15a: Extrinsic apoptosis model incorporating the MOMP model from *[Howells2011]*.

2.2.2 MOMP-only model files (M1b-M15b)

Lopez Models (M1b - M3b)

[source]

Model M1b: Expanded "embedded together" model of MOMP. [source]

Model M2b: Expanded "direct" model of MOMP. [source]

Model M3b: Expanded "indirect" model of MOMP.

Albeck Models (M4b - M8b)

[source]

Model M4b: MOMP model "Minimal Model" (Figure 11b) from *[Albeck2008]*. [source]

Model M5b: MOMP model "Model B + Bax multimerization" (Figure 11c) from *[Albeck2008]*. [source]

Model M6b: MOMP model "Model C + mitochondrial transport" (Figure 11d) from *[Albeck2008]*. [source]

Model M7b: MOMP model "Current model" (Figure 11e) from *[Albeck2008]*. [source]

Model M8b: MOMP model "Current model + cooperativity" (Figure 11f) from *[Albeck2008]*.

Shen Models (M9b - M15b)

[source]

Model M9b: MOMP model from *[Chen2007biophysj]*. [source]

Model M10b: "Direct" MOMP model from *[Chen2007febs]*. [source]

Model M11b: "Indirect" MOMP model from *[Chen2007febs]*. [source]

Model M12b: "Direct" MOMP model from *[Cui2008]*. [source]

Model M13b: "Direct 1" MOMP model from *[Cui2008]*. [source]

Model M14b: "Direct 2" MOMP model from *[Cui2008]*. [source]

Model M15b: MOMP model from *[Howells2011]*.

The details of the various macro implementations are found in the following four files:

2.2.3 lopez_modules.py

[source]

Overview

Three models of MOMP (`direct()`, `indirect()`, and `embedded()`), each incorporating a larger repertoire of Bcl-2 family members than previously published models, including:

- One **activator,** Bid.
- Two **sensitizers,** Bad and Noxa.
- Two **effectors,** Bax and Bak.
- Three **anti-apoptotics**, Bcl-2, Bcl-xL, and Mcl-1.

The Models

Note that in each of the three models, interactions between Bcl-2 proteins only occur at the mitochondrial membrane. The following are brief descriptions of each model.

- `direct()`. In this model, tBid directly activates both Bax and Bak; the anti-apoptotics bind tBid and the sensitizers (Bad and Noxa) but not Bax and Bak.
- `indirect()`. Bax and Bak are not explicitly activated by tBid, but rather are in an equilibrium between inactive and active states. The anti-apoptotics bind tBid, sensitizers, and Bax and Bak.
- `embedded()`. Combines elements of both direct and indirect: tBid activates Bax and Bak; the anti-apoptotics bind tBid, sensitizers and Bax and Bak. In addition, Bax and Bak are able to auto-activate.

Organization of models into Motifs

Because the three models share many aspects, the mechanisms that they share have been written as small "motifs" implemented as subroutines. These are:

- `translocate_tBid_Bax_BclxL()`
- `tBid_activates_Bax_and_Bak()`
- `tBid_binds_all_anti_apoptotics()`
- `sensitizers_bind_anti_apoptotics()`
- `effectors_bind_anti_apoptotics()`
- `lopez_pore_formation()`

The implementation details of these motifs can be seen by examining the source code.

Monomer and initial declaration functions

The models share the same set of Monomer and initial condition declarations, which are contained with the following two functions:

- *momp_monomers()*
- *declare_initial_conditions()*

Functions

earm.lopez_modules.**momp_monomers**()

Declare the monomers for the Bcl-2 family proteins, Cyto c, and Smac.

Annotation() declarations embedded in this function associate UniProt identifiers with each protein.

'bf' is the site to be used for all binding reactions (with the exception of Bax and Bak, which have additional sites used for oligomerization).

The 'state' site denotes various localization and/or activity states of a Monomer, with 'C' denoting cytoplasmic localization and 'M' mitochondrial localization. Most Bcl-2 proteins have the potential for both cytoplasmic and mitochondrial localization, with the exceptions of Bak and Bcl-2, which are apparently constitutively mitochondrial.

earm.lopez_modules.**declare_initial_conditions**()

Declare initial conditions for Bcl-2 family proteins, Cyto c, and Smac.

earm.lopez_modules.**translocate_tBid_Bax_BclxL**()

tBid, Bax and BclXL translocate to the mitochondrial membrane.

earm.lopez_modules.**tBid_activates_Bax_and_Bak**()

tBid activates Bax and Bak.

earm.lopez_modules.**tBid_binds_all_anti_apoptotics**()

tBid binds and inhibits Bcl2, Mcl1, and Bcl-XL.

The entries given in the *bind_table* are dissociation constants taken from Certo et al. (see ref). Dissociation constants in Certo et al. were published as nanomolar binding affinities; here they are converted into units of numbers of molecules by multiplying by *N_A* (Avogadro's number) and *V* (a default cell volume, specified in shared.py.

The default forward rate represents diffusion limited association (1e6 Molar^-1 s^-1) and is converted into units of molec^-1 s^-1 by dividing by *N_A*V*.

Certo, M., Del Gaizo Moore, V., Nishino, M., Wei, G., Korsmeyer, S., Armstrong, S. A., & Letai, A. (2006). Mitochondria primed by death signals determine cellular addiction to antiapoptotic BCL-2 family members. Cancer Cell, 9(5), 351-365. *doi:10.1016/j.ccr.2006.03.027*

earm.lopez_modules.**sensitizers_bind_anti_apoptotics**()

Binding of Bad and Noxa to Bcl2, Mcl1, and Bcl-XL.

See comments on units for *tBid_binds_all_anti_apoptotics()*.

earm.lopez_modules.**effectors_bind_anti_apoptotics**()

Binding of Bax and Bak to Bcl2, BclxL, and Mcl1.

Affinities of Bak for Bcl-xL and Mcl-1 are taken from Willis et al.

Preferential affinity of Bax for Bcl-2 and Bcl-xL were taken from Zhai et al. Bax:Bcl2 and Bax:Bcl-xL affinities were given order of magnitude estimates of 10nM.

See comments on units for *tBid_binds_all_anti_apoptotics()*.

Willis, S. N., Chen, L., Dewson, G., Wei, A., Naik, E., Fletcher, J. I., Adams, J. M., et al. (2005). Proapoptotic Bak is sequestered by Mcl-1 and Bcl-xL, but not Bcl-2, until displaced by BH3-only proteins. Genes & Development, 19(11), 1294-1305. *doi:10.1101/gad.1304105*

Zhai, D., Jin, C., Huang, Z., Satterthwait, A. C., & Reed, J. C. (2008). Differential regulation of Bax and Bak by anti-apoptotic Bcl-2 family proteins Bcl-B and Mcl-1. The Journal of biological chemistry, 283(15), 9580-9586. *doi:10.1074/jbc.M708426200*

`earm.lopez_modules.`**`lopez_pore_formation`**(*do_pore_transport=True*)
Pore formation and transport process used by all modules.

`earm.lopez_modules.`**`embedded`**(*do_pore_transport=True*)
Direct and indirect modes of action, occurring at the membrane.

`earm.lopez_modules.`**`indirect`**(*do_pore_transport=True*)
Bax and Bak spontaneously form pores without activation. The "activator" tBid binds all of the anti-apoptotics.

`earm.lopez_modules.`**`direct`**(*do_pore_transport=True*)
Anti-apoptotics prevent BH3-onlies from activating Bax and Bak.

Bax and Bak require activation to be able to form pores. The anti-apoptotics don't inhibit activated Bax and Bak; their only role is to bind BH3-onlies.

2.2.4 albeck_modules.py

[source]

Overview

PySB implementations of the extrinsic apoptosis reaction model version 1.0 (EARM 1.0) originally published in *[Albeck2008]*.

This file contains functions that implement the extrinsic pathway in three modules:

- Receptor ligation to Bid cleavage (`rec_to_bid()`)
- Mitochondrial Outer Membrane Permeabilization (MOMP, see below)
- Pore transport to effector caspase activation and PARP cleavage (`pore_to_parp()`).

For the (MOMP) segment there are five variants, which correspond to the five models described in Figure 11 of *[Albeck2008]*:

- "Minimal Model" (Figure 11b, `albeck_11b()`)
- "Model B + Bax multimerization" (Figure 11c, `albeck_11c()`)
- "Model C + mitochondrial transport" (Figure 11d, `albeck_11d()`)
- "Current model" (Figure 11e, `albeck_11e()`)
- "Current model + cooperativity" (Figure 11f, `albeck_11f()`)

Functions

`earm.albeck_modules.`**`ligand_to_c8_monomers`**()
Declares ligand, receptor, DISC, Flip, Bar and Caspase 8.

'bf' is the site to be used for all binding reactions.

The 'state' site denotes various localization and/or activity states of a Monomer, with 'C' denoting cytoplasmic localization and 'M' mitochondrial localization.

`earm.albeck_modules.`**`momp_monomers`**`()`
Declare the monomers used in the Albeck MOMP modules.

`earm.albeck_modules.`**`apaf1_to_parp_monomers`**`()`
Declares CytochromeC, Smac, Apaf-1, the Apoptosome, Caspases 3, 6, 9, XIAP and PARP.

The package variable 'bf' specifies the name of the site to be used for all binding reactions.

The 'state' site denotes various localization and/or activity states of a Monomer, with 'C' denoting cytoplasmic localization and 'M' mitochondrial localization.

`earm.albeck_modules.`**`all_monomers`**`()`
Shorthand for calling ligand_to_c8, momp, and apaf1_to_parp macros.

Internally calls the macros ligand_to_c8_monomers(), momp_monomers(), and apaf1_to_parp_monomers() to instantiate the monomers for each portion of the pathway.

`earm.albeck_modules.`**`rec_to_bid`**`()`
Defines the interactions from ligand (e.g. TRAIL) binding to Bid activation as per EARM 1.0.

Uses L, R, DISC, flip, C8, BAR, and Bid monomers and their associated parameters to generate the rules that describe Ligand/Receptor binding, DISC formation, Caspase-8 activation and inhibition by flip and BAR as originally specified in EARM 1.0.

Declares initial conditions for ligand, receptor, Flip, C8, and Bar.

`earm.albeck_modules.`**`pore_to_parp`**`()`
Defines what happens after the pore is activated and Cytochrome C and Smac are released.

Uses CytoC, Smac, Apaf, Apop, C3, C6, C8, C9, PARP, XIAP monomers and their associated parameters to generate the rules that describe apoptosome formation, XIAP inhibition, activation of caspases (including caspase-6-mediated feedback), and cleavage of effector caspase substrates as specified in EARM 1.0.

Declares initial conditions for CytoC, Smac, Apaf-1, Apoptosome, caspases 3, 6, and 9, XIAP, and PARP.

`earm.albeck_modules.`**`Bax_tetramerizes`**`(`*bax_active_state='A'*, *rate_scaling_factor=1*`)`
Creates rules for the rxns Bax + Bax <> Bax2, and Bax2 + Bax2 <> Bax4.

Parameters **bax_active_state** : string: 'A' or 'M'

The state value that should be assigned to the site "state" for dimerization and tetramerization to occur.

rate_scaling_factor : number

A scaling factor applied to the forward rate constants for dimerization and tetramerization.

`earm.albeck_modules.`**`Bcl2_binds_Bax1_Bax2_and_Bax4`**`(`*bax_active_state='A'*, *rate_scaling_factor=1*`)`
Creates rules for binding of Bcl2 to Bax monomers and oligomers.

Parameters **bax_active_state** : string: 'A' or 'M'

The state value that should be assigned to the site "state" for the Bax subunits in the pore.

rate_scaling_factor : number

A scaling factor applied to the forward rate constants for binding between Bax (monomers, oligomers) and Bcl2.

`earm.albeck_modules.albeck_11b(do_pore_transport=True)`

Minimal MOMP model shown in Figure 11b.

Features:

- Bid activates Bax
- Active Bax is inhibited by Bcl2
- Free active Bax binds to and transports Smac to the cytosol

`earm.albeck_modules.albeck_11c(do_pore_transport=True)`

Model incorporating Bax oligomerization.

Features:

- Bid activates Bax
- Active Bax dimerizes; Bax dimers dimerize to form tetramers
- Bcl2 binds/inhibits Bax monomers, dimers, and tetramers
- Bax tetramers bind to and transport Smac to the cytosol

`earm.albeck_modules.albeck_11d(do_pore_transport=True)`

Model incorporating mitochondrial transport.

Features:

- Bid activates Bax
- Active Bax translocates to the mitochondria
- All reactions on the mito membrane have increased association rates
- Mitochondrial Bax dimerizes; Bax dimers dimerize to form tetramers
- Bcl2 binds/inhibits Bax monomers, dimers, and tetramers
- Bax tetramers bind to and transport Smac to the cytosol

`earm.albeck_modules.albeck_11e(do_pore_transport=True)`

Model incorporating mitochondrial transport and pore "insertion."

Features:

- Bid activates Bax
- Active Bax translocates to the mitochondria
- All reactions on the mitochondria have increased association rates
- Mitochondrial Bax dimerizes; Bax dimers dimerize to form tetramers
- Bcl2 binds/inhibits Bax monomers, dimers, and tetramers
- Bax tetramers bind to mitochondrial "sites" and become active pores
- Active pores bind to and transport Smac to the cytosol

`earm.albeck_modules.albeck_11f(do_pore_transport=True)`

Model as in 11e, but with cooperative assembly of Bax pores.

Association rate constants for Bax dimerization, tetramerization, and insertion are set so that they increase at each step (from 1e-8 to 1e-7 and then 1e-6), thereby creating cooperative assembly.

See also the documentation for albeck_11e().

2.2.5 shen_modules.py

[source]

Overview

PySB implementations of Bcl2-models from the group of Pingping Shen, along with other derived, closely related models.

In a series of papers from 2007-2010, the research group of Pingping Shen implemented and investigated models of Bcl-2 family interactions. In this file we have re-implemented these models using PySB. We have also included a model from *[Howells2011]* which is a fairly straightforward extension of a Shen group model from *[Chen2007biophysj]*.

MOMP model implementations

The implementations of the various models are contained within the following functions:

- `chen_biophys_j()`
- `chen_febs_indirect()`
- `chen_febs_direct()`
- `cui_direct()`
- `cui_direct1()`
- `cui_direct2()`
- `howells()`

Model descriptions (with references) are available in the documentation for each function.

The models are closely related, and many of the later models are derived from earlier ones. The models have been implemented in such a way as to make this hierarchy transparent.

Shared functions

In addition to the implementations of the models themselves, this file also contains two macros that are re-used by the various models:

- `momp_monomers()`, which declares the Bcl-2 molecule types used in the models, and
- `shen_pore_transport()`, which declares the set of transport reactions required for the release of Cytochrome c and Smac.

Parameter values

In the original papers, species quantities and forward rate constants were either given in units of micromolar (*[Chen2007biophysj]*, *[Howells2011]*) or nanomolar (*[Chen2007febs]*, *[Cui2008]*). For consistency, these values have been written in terms of their molar equivalents (for example, 0.1uM = 0.1e-6 M; 3 uM^-1 s^1 = 3e6 M^-1 s^-1). Concentrations have been converted into units of numbers of molecules according to:

No. of molecules = Conc * N_A * vol

where N_A is Avogadro's number and vol is the cell volume, which is given a default value in the global variable *V* defined in shared.py. Similarly, forward rate constants are converted into stochastic rate constants according to:

Stoch. rate constant = Det. rate constant / (N_A * vol)

Functions

`earm.shen_modules.`**`momp_monomers`**`()`

Declares the signatures of the Bcl-2 family monomers used in all of the Shen models.

In principle, each Shen MOMP model implementation could declare its own set of Bcl-2 monomers, each with its own site and state signature. In the interest of consistency, a unified set of monomer signatures that supports all of the models is defined here.

`earm.shen_modules.`**`shen_pore_transport`**`(`*pore_size=4*`)`

Implements release of Cytochrome C and Smac.

Uses the same model as the original EARM 1.0 (*[Albeck2008]*), in which pore transport is modeled as binding of the cargo (cytochrome C or Smac) to the active pore, and then release, in a catalysis-like mechanism.

The initial conditions for cytochrome C and Smac, and the rate constants for transport, are also taken from EARM 1.0.

`earm.shen_modules.`**`chen_biophys_j`**`(`*do_pore_assembly=True*, *do_pore_transport=False*`)`

Model drawn from *[Chen2007biophysj]*.

Model features (see the source code):

- Activation of Bax by an activator (tBid) in a one-step, hit-and-run manner; Bax activation is reversible.
- Bcl2 binds both tBid and Bax Bax can displace tBid from Bcl-2 (but not the reverse).
- If Bax oligomerization is incorporated into the model (see *do_pore_assembly* argument, below), then this occurs as a spontaneous, order 4 reaction.

This model combines both "direct" type and "indirect" type elements in that Bcl-2 is capable of binding both Bid and Bax (see *bind_table* call in the source code).

Parameters **do_pore_assembly** : True (default) or False

If True, adds the formation of Bax oligomers to the model. If False, the model's most downstream element is Bax activation. This is included for two reasons: first, the original publication included two variant models, one with and one without Bax oligomerization, so this allows this aspect of the original models to be explored. Second, it allows a model that extends this model to implement a different model of Bax pore assembly (for example, as is the case with cui_direct).

do_pore_transport : True or False (default)

If True, adds the release of Cytochrome C and Smac to the model by calling the function `shen_pore_transport()`. If CytoC/Smac release are not incorporated into the model, the model matches the originally published model but can't be composed into the full extrinsic apoptosis pathway.

`earm.shen_modules.`**`chen_febs_indirect`**`(`*do_pore_assembly=True*, *do_pore_transport=False*`)`

The "indirect" model drawn from *[Chen2007febs]*.

Model features (see the source code):

- There is no activation of Bax by tBid. Bax starts out constitutively "active" in that in its initial state, it is able to form oligomers.
- Bcl-2 can bind tBid and Bax.

Parameters **do_pore_assembly** : True (default) or False

As for `chen_biophys_j()`.

do_pore_transport : True or False (default)

As for `chen_biophys_j()`.

`earm.shen_modules.`**`chen_febs_direct`**(*do_pore_assembly=True*, *do_pore_transport=False*)

The "direct" model drawn from *[Chen2007febs]*.

Model features (see the source code):

- Activation of Bax by an activator (tBid) in a one-step, hit-and-run manner; Bax activation is reversible.
- Bcl-2 can bind tBid, but not Bax.

Parameters **do_pore_assembly** : True (default) or False

As for `chen_biophys_j()`.

do_pore_transport : True or False (default)

As for `chen_biophys_j()`.

`earm.shen_modules.`**`cui_direct`**(*do_pore_transport=False*)

The "direct" model drawn from *[Cui2008]*.

Builds on the direct model from *[Chen2007febs]*, implemented in `chen_febs_direct()` (see source code).

Parameters **do_pore_transport** : True or False (default)

As for `chen_biophys_j()`.

`earm.shen_modules.`**`cui_direct1`**(*do_pore_transport=False*)

The "direct 1" model drawn from *[Cui2008]*.

Builds on the (base) direct model from *[Cui2008]*, implemented in `cui_direct()` (see source code).

Parameters **do_pore_transport** : True or False (default)

As for `chen_biophys_j()`.

`earm.shen_modules.`**`cui_direct2`**(*do_pore_transport=False*)

The "direct 2" model drawn from *[Cui2008]*.

Builds on the "direct 1" model from *[Cui2008]*, implemented in `cui_direct1()` (see source code).

Parameters **do_pore_transport** : True or False (default)

As for `chen_biophys_j()`.

`earm.shen_modules.`**`howells`**(*do_pore_assembly=True*, *do_pore_transport=False*)

The model drawn from *[Howells2011]*.

This model builds on the model from *[Chen2007biophysj]*, implemented in `chen_biophys_j()`. The core reactions from the Chen et al. model are the same, but Howells et al. modify some parameter values and add a number of Bad-related reactions, including (see source code):

- Unphosphorylated Bad spontaneously translocates between cytosol and mitochondria
- Bad binds Bcl-2
- Bad displaces tBid from Bcl-2
- Cytosolic, mitochondrial, and Bad in a mitochondrial Bad:Bcl2 complex can be phosphorylated at various rates (this is modeled as a first-order reaction with no explicit representation of kinases)

- Bad can be sequestered by, and released from, 14-3-3 domains in the cytosol (modeled as a first-order reaction with no explicit representation of 14-3-3-domain-containing proteins)

Parameters **do_pore_assembly** : True (default) or False

As for `chen_biophys_j()`.

do_pore_transport : True or False (default)

As for `chen_biophys_j()`.

2.2.6 shared.py

[source]

Overview

This module declares a number of functions and variables that are used by many of the EARM 2 models. The functions can be divided into the following four categories:

1. Functions that are specific to the models in EARM 2, but are used by all of them. The only macro of this type is
   - `observables()`
2. Aliases to generalized macros in pysb.macros that provide default values for site names or other arguments. Macros of this type include:
   - `catalyze()`
   - `bind()`
   - `bind_table()`
   - `assemble_pore_sequential()`
   - `pore_transport()`
3. Macros for mechanisms that appear within the models previously published by the research group of Pingping Shen (or the model from *[Howells2011]*, which is derived from one of Shen's models):
   - `assemble_pore_spontaneous()`
   - `displace()`
   - `displace_reversibly()`
4. Macros for mechanisms that appear within the models described in our group's earlier work, specifically the models described in *[Albeck2008]*:
   - `catalyze_convert()`
   - `one_step_conv()`
   - `pore_bind()`

Functions

`earm.shared.`**`observables`**`()`

Declare observables commonly used for the TRAIL pathway.

Declares truncated (and mitochondrial) Bid, cytosolic (i.e., released) Smac, and cleaved PARP.

`earm.shared.`**`catalyze`**(*enz*, *sub*, *product*, *klist*)
Alias for pysb.macros.catalyze with default binding sites.

`earm.shared.`**`bind`**(*a*, *b*, *klist*)
Alias for pysb.macros.bind with default binding sites.

`earm.shared.`**`bind_table`**(*table*, ***kwargs*)
Alias for pysb.macros.bind_table with default binding sites.

`earm.shared.`**`assemble_pore_sequential`**(*subunit*, *size*, *klist*)
Alias for pysb.macros.assemble_pore_sequential with default sites.

Uses default pore site names as the sites for subunit-subunit binding in the pore.

`earm.shared.`**`pore_transport`**(*subunit*, *size*, *csource*, *cdest*, *ktable*)
Alias for pysb.macros.pore_transport with default arguments.

- Uses the default binding site names for the binding site on the pore and on the cargo
- Uses the default pore site names for subunit-subunit binding
- Uses only a single size (not a min and max size) for the size of transport-competent pores

`earm.shared.`**`assemble_pore_spontaneous`**(*subunit*, *klist*)
Generate the order-4 assembly reaction 4*Subunit <> Pore.

`earm.shared.`**`displace`**(*lig1*, *lig2*, *target*, *k*)
Generate unidirectional displacement reaction L1 + L2:T >> L1:T + L2.

The signature can be remembered with the following formula: "lig1 displaces lig2 from target."

`earm.shared.`**`displace_reversibly`**(*lig1*, *lig2*, *target*, *klist*)
Generate reversible displacement reaction L1 + L2:T <> L1:T + L2.

The signature can be remembered with the following formula: "lig1 displaces lig2 from target." The first rate given in in klist specifies the forward rate of this reaction; the second specifies the reverse rate.

`earm.shared.`**`catalyze_convert`**(*sub1*, *sub2*, *product*, *klist*, *site='bf'*)
Automation of the Sub1 + Sub2 <> Sub1:Sub2 >> Prod two-step reaction.

Because product is created by the function, it must be fully specified.

`earm.shared.`**`one_step_conv`**(*sub1*, *sub2*, *product*, *klist*, *site='bf'*)
Bind sub1 and sub2 to form one product: sub1 + sub2 <> product.

`earm.shared.`**`pore_bind`**(*subunit*, *sp_site1*, *sp_site2*, *sc_site*, *size*, *cargo*, *c_site*, *klist*)
Generate rules to bind a monomer to a circular homomeric pore.

The pore structure is defined by the *pore_species* macro – *subunit* monomers bind to each other from *sp_site1* to *sp_site2* to form a closed ring. The binding reaction takes the form pore + cargo <> pore:cargo.

Parameters **subunit** : Monomer or MonomerPattern

Subunit of which the pore is composed.

sp_site1, sp_site2 : string

Names of the sites where one copy of *subunit* binds to the next.

sc_site : string

Name of the site on *subunit* where it binds to the cargo *cargo*.

size : integer

Number of subunits in the pore at which binding will occur.

cargo : Monomer or MonomerPattern

Cargo that binds to the pore complex.

c_site : string

Name of the site on *cargo* where it binds to *subunit*.

klist : list of Parameters or numbers

List containing forward and reverse rate constants for the binding reaction (in that order). Rate constants should either be both Parameter objects or both numbers. If Parameters are passed, they will be used directly in the generated Rules. If numbers are passed, Parameters will be created with automatically generated names based on <TODO> and these parameters will be included at the end of the returned component list.

2.3 MOMP module "boundaries"

In the interest of consistency, all of the models have been defined with the same boundaries in terms of their position in the overall extrinsic apoptosis pathway: they are all triggered by the addition of an active BH3-only species (e.g., tBid) as their most "upstream" event, and they all result in the release in one or more mitochondrial substances (e.g. Cytochrome C and/or Smac) as their most downstream event. This represents a compromise between the approach of the MOMP models described in Albeck et al (in which caspase-8, rather than tBid, served as the input) and the models of the Shen group, in which active Bax or Bax pores, rather than Cytochrome C or Smac, served as the output.

While these interface boundaries represent the default condition, they can be modified by passing parameters in to the module macro. For example, by setting *do_pore_transport=False* in the call to one of the Shen models, the Cytochrome C and Smac release reactions are not added, and the models can be directly compared to their originally published versions. Similarly, the upstream caspase-8/Bid reactions can be added to the Albeck MOMP models to make them consistent with their published versions.

Since our purpose in using these models is primarily to embed them in a common pathway context, rather than to reproduce previous results for posterity, our conclusion in working with them was that it is better to have a consistent interface by default and reproduce published results by modifying the model rather than implement the model as published by default and then have to specifically modify each one separately to fit the pathway context appropriately.

2.4 How to use the models

To import a model use the syntax:

```
from earm.lopez_embedded import model
```

That's it. You now have a model object that you can query, simulate, perform parameter estimation on, etc. If you wanted the MOMP-only version, which is in the sub-module `mito`, simply run:

```
from earm.mito.lopez_embedded import model
```

If you want to work with multiple models at the same time (e.g., to compare them), you can write:

```
from earm.chen2007_indirect import model as indirect
from earm.chen2007_direct import model as direct
```

For more information on the kinds of analysis you can do using PySB models, see the PySB documentation.

2.5 Parameter values

Parameter values (both rate constants and initial protein concentrations) are embedded directly in the model code rather than in a separate table or file. The values in the model definition represent estimates or nominal values and can be easily overridden using values obtained (for example) by measurement or parameter estimation algorithms. We do not maintain a separate list or table of parameter values, as we have found that the clearest description of the meaning of a rate parameter is the macro or rule statement in which it is embedded.

If desired, lists of all model parameters can be obtained via the parameters instance variable of the model object, i.e.:

```
model.parameters
```

A list of all parameter names can be obtained using the list comprehension:

```
[p.name for p in model.parameters]
```

2.6 The code is meant to be read!

As much as possible, we have attempted to make the code for models themselves transparent and well-documented. The documentation for each model topology has been embedded inline in the model source code, and the documentation provided in the Implementation details (code and documentation) section of the documentation is drawn directly from this source.

Moreover, the models have been written using a high-level vocabulary of frequently re-used macros and motifs, with the aim of revealing broad similarities and differences between models. The models thus consist of statements such as:

```
translocate_tBid_Bax_BclXL()
catalyze(Bid(state='T'), Bax(state='M'), Bax(state='A'), klist)
```

which can be read as saying that "tBid, Bax and BclXL translocate [to the mitochondrial membrane], and tBid catalyzes Bax from a Mitochondrial (but inactive) state to an Active state." Understanding the precise mechanisms of these macros (as expressed in terms of rules and reactions) takes some familiarity with their implementation, but as there is a fairly limited set of macros, this should hopefully not present a significant barrier.

CHAPTER 3

Implementation details (code and documentation)

This section contains documentation of the implementations of each model.

CHAPTER 4

Tests

The EARM repository contains a number of tests that ensure that the PySB versions of previously published models accurately duplicate their ODE-based counterparts. The tests guarantee that this continues to be the case despite any future changes in PySB or EARM.

Tests are written using the Python testing modules *unittest* and *nose*.

4.1 Basic tests of all models

[source]

A suite of tests to ensure that each of the models in the EARM repository can be successfully loaded and have its reaction network generated.

For every model, pysb.bng.generate_network(model) is called; if there are no errors, the test passes.

4.1.1 Functions

`earm.tests.test_models_net_gen.`**`test_generate_network`**`()`

Test all models for successful network generation by calling `check_generate_network()` for each model.

`earm.tests.test_models_net_gen.`**`check_generate_network`**`(`*model*`)`

Tests that network generation occurs without error for the given model.

4.2 Lopez model tests

[source]

Checks the MOMP-only versions of the Lopez models, that is

- `earm.mito.lopez_embedded`
- `earm.mito.lopez_direct`
- `earm.mito.lopez_indirect`

against a previously validated and serialized state. Each test in the module uses the function pysb.testing.check_model_against_component_list to perform the comparison.

4.2.1 Functions and Classes

`earm.tests.test_lopez_models.`**`test_lopez_embedded`**`()`
Test the earm.mito.lopez_embedded model against a serialized state.

`earm.tests.test_lopez_models.`**`test_lopez_direct`**`()`
Test the earm.mito.lopez_direct model against a serialized state.

`earm.tests.test_lopez_models.`**`test_lopez_indirect`**`()`
Test the earm.mito.lopez_indirect model against a serialized state.

`earm.tests.test_lopez_models.`**`pickle_lopez_models`**`()`
The pickling procedure that was used to serialize the components of the Lopez models as a record of a validated state.

4.3 Albeck model tests

[source]

This module contains a number of tests that can be used to verify that the behavior of the MOMP-only sub-models based on *[Albeck2008]* and re-written in PySB match the original publication. To verify that the PySB version of the models match the original models (which were written in MATLAB), timecourse data was generated in MATLAB using the original MATLAB code, and saved in tab-separated data files (*albeck_11{b-f}.tsv*). The PySB models are then run using the same input and the output is verified to match the original data (within integration tolerances).

This model verification procedure is written as a series of unit tests, one for each sub-model, using the built-in Python package unittest.

To run the tests, simply execute this file at the command line, i.e.:

```
python test_albeck_models.py
```

and all tests should pass.

4.3.1 Functions and Classes

`earm.tests.test_albeck_models.`**`add_caspase8`**`(`*model*`)`
Add the reaction C8 + Bid <-> C8:Bid -> C8 + tBid.

All of the MOMP sub-models in this model repository have been written to have tBid as their upstream "interface," and Smac/Cytochrome C release as their downstream "interface." However, in the original publication *[Albeck2008]*, the authors incorporated the additional upstream step of Bid cleavage by caspase 8 into all of the sub-models that they explored.

Therefore, to match the output of the PySB models in this repository to the output produced by the original MATLAB models, it is necessary to add the upstream caspase-8:Bid reactions.

This function takes a model object and adds the necessary elements:

- Caspase-8 Monomer
- Caspase-8 initial condition
- Caspase-8/Bid cleavage reaction and associated parameters

In addition, since in the original publication the plotted figures considered Smac release kinetics in the absence of Cytochrome C release, this function sets the Cytochrome C initial condition to 0 (this prevents Cytochrome C from competing with Smac for pore transport, which affects the observed Smac release kinetics slightly).

earm.tests.test_albeck_models.**run_figure_sim**(*model*)

Run the C8 dose-response series shown in Fig. 11 of *[Albeck2008]*.

Returns **[t, outputs]** : list containing two numpy.array objects

t: **The time coordinates of each timepoint, in seconds, from 0 to** 60,000 seconds (15 hours), at 60-second intervals.

outputs: **A 901 x 7 array. Each row corresponds to the fraction of Smac** released into the cytosol at each timepoint. Each column correspond to a distinct caspase 8 dose, from lowest to highest: [1, 5, 10, 50, 100, 500, 1000], in molecules per cell.

earm.tests.test_albeck_models.**plot_figure**(*model*, *data_file*)

Plot the PySB model output alongside the original MATLAB output.

This function is not used explicitly by any of the testing code, but it is useful for a visual comparison of the output of the PySB model to the original MATLAB model output.

Parameters **model** : pysb.model

The PySB MOMP model.

data_file : The original MATLAB data file.

earm.tests.test_albeck_models.**matches_figure**(*model*, *data_file*)

Test whether the PySB model output matches the original MATLAB output.

Calls *run_figure_sim()* to generate the PySB model output, then loads the MATLAB data file and compares the outputs using the function numpy.allclose. Returns True if every timepoint from the dose-response series matches the original MATLAB output to within one order of magnitude of the integration tolerance.

class earm.tests.test_albeck_models.**TestAlbeck11b**(*methodName='runTest'*)

Test the PySB model based on the topology shown in Figure 11b.

Methods

test_figure()

Test that the model reproduces the plots shown in Figure 11b.

class earm.tests.test_albeck_models.**TestAlbeck11c**(*methodName='runTest'*)

Test the PySB model based on the topology shown in Figure 11c.

Methods

test_figure()

Test that the model reproduces the plots shown in Figure 11c.

class earm.tests.test_albeck_models.**TestAlbeck11d**(*methodName='runTest'*)

Test the PySB model based on the topology shown in Figure 11d.

Methods

test_figure()

Test that the model reproduces the plots shown in Figure 11d.

class earm.tests.test_albeck_models.**TestAlbeck11e**(*methodName='runTest'*)

Test the PySB model based on the topology shown in Figure 11e.

Methods

test_figure()
Test that the model reproduces the plots shown in Figure 11e.

class earm.tests.test_albeck_models.**TestAlbeck11f**(*methodName='runTest'*)
Test the PySB model based on the topology shown in Figure 11f.

Methods

test_figure()
Test that the model reproduces the plots shown in Figure 11f.

4.4 Shen/Howells model tests

[source]

This file contains a series of tests to ensure that the ODEs generated by the PySB version of each model exactly match those of the original publication. The procedure for validating the models (and creating the tests) was as follows:

1. Generate the ODEs from the PySB model using pysb.bng.generate_equations
2. Programmatically rename all parameters and species to use the names from the original publication.
3. (For `TestChenBiophysJ` and `TestHowells`) Programmatically verify the values of all parameters against the values used in the original publication.
4. Manually verify the transformed PySB ODEs against the ODEs from the original publication.
5. Include the (verified) output in a test to ensure that the model continues to produce the correct output in the face of any future changes in macros, libraries, etc.

Note that the models in the earm.mito module, by default, have a zero initial condition for the activator species (Bid). To generate the ODEs used by the original publications, it is necessary to add an additional non-zero initial condition for Bid. This is accomplished in the setUp() method of each test.

4.4.1 Functions and Classes

earm.tests.test_shen_models.**convert_odes**(*model*, *p_name_map*, *s_name_map_by_pattern*)
Substitutes species and parameter names using the given name mappings.

Parameters **model** : pysb.core.Model

The model to be converted.

p_name_map : dict

A dict where the keys are the parameter names of the PySB model and the values are the parameter names of the original model, e.g. {'bind_BidT_Bcl2_kf': 'ka_tBid_Bcl2'}

s_name_map : dict

A dict where the keys are the string representations of the species from the PySB model, generated by calling str(species), and the values are the species names in the original model, e.g. {'Bid(bf=None, state=T)': 'Act'}

Returns **List of strings** :

One string entry for each ODE in the model. Each ODE is represented as a string, e.g. "d[Act]/dt = ..."

`earm.tests.test_shen_models.`**`odes_match`**(*generated_odes*, *validated_odes*)

Return True if the ODEs match.

Both args are dicts, where the key is the original species name, and the value is the string representation of the ODE.

`earm.tests.test_shen_models.`**`convert_parameters`**(*model*, *p_name_map*, *original_units='micromolar'*)

Convert the parameters from the PySB version of the model to have names and units that match the original publication.

Used to test that the nominal parameter values of the PySB model match those of the original publication.

Parameters **model** : pysb.core.Model

The model to be converted.

p_name_map : dict

A dict where the keys are the parameter names of the PySB model and the values are the parameter names of the original model, e.g. {'bind_BidT_Bcl2_kf': 'ka_tBid_Bcl2'}

original_units: string, 'micromolar' or 'nanomolar' :

By convention, the units in the PySB versions of all models are expressed as numbers of molecules with a default volume V, specified in earm.shared. Convert parameters converts the parameter values from numbers of molecules back into concentration units that match the original publication (either micromolar or nanomolar).

class `earm.tests.test_shen_models.`**`TestChenBiophysJ`**(*methodName='runTest'*)

Test the PySB version of the model from *[Chen2007biophysj]*.

Methods

`test_odes`()

Check the generated ODEs against manually validated ODEs.

The ODEs generated by the PySB model match those of the paper with the following two caveats:

1. In the equation for d[Bcl2]/dt, in the paper the authors group the terms

   •AcBaxBcl2*k4 + ActBcl2*k6'

   into the single term

   •k_Bcl2 * Bcl2_nonfree

   with the comment that "Bcl2_nonfree indicates the total concentration of Bcl2 associated with both activated Bax and Activator ([Bcl2_nonfree] = [AcBaxBcl2] + [ActBcl2]). We use k_bcl2 to represent the rate of non-free Bcl2 shifting to free Bcl2, assuming that free Bcl2 originates from both Bcl2 non-free forms at the same rate."

   In addition, in the legend for Table 1 (which lists parameters) they state that: "We set k_bcl2 (the rate of non-free Bcl2 shifting to free Bcl2)... equal to k6 assuming that free Bcl2 originate [sic] from AcBaxBcl2 at the same rate with ActBcl2."

   So, obviously this substitution of Bcl2_nonfree for AcBaxBcl2 and ActBcl2 works if k4 and k6 are equal, which they claim as an assumption; however, in their table of parameter values, they list k4 as having a value of 0.001 s^-1, and k6 as having a value of 0.04 s^-1.

2. It should also be noted that the parameter for spontaneous pore formation, k9, has already been multiplied by 4 from its nominal value listed in the paper. This accounts for BNG's (appropriate) addition of the coefficients of 0.25 to the Bax polymerization forward reaction, due to the reaction being a homomeric binding reaction.

3. Because the rate of displacement of Bax from Bcl2 by tBid is set to 0 in the original model, this reaction and its associated rate parameter k8 have been eliminated from the model.

test_parameters()

Check that the values of the parameters in the PySB model match those of the original.

The parameter values in the test below have been verified to match the values listed in Table 1 of *[Chen2007biophysj]*.

class earm.tests.test_shen_models.**TestChenFEBS_Indirect**(*methodName='runTest'*)

Test the PySB version of the "indirect" model from *[Chen2007febs]*.

Methods

test_odes()

Check the generated ODEs against manually validated ODEs.

class earm.tests.test_shen_models.**TestChenFEBS_Direct**(*methodName='runTest'*)

Test the PySB version of the "direct" model from *[Chen2007febs]*.

Methods

test_odes()

Check the generated ODEs against manually validated ODEs.

class earm.tests.test_shen_models.**TestCui_Direct**(*methodName='runTest'*)

Test the PySB version of the "direct" model from *[Cui2008]*.

Methods

test_odes()

Check the generated ODEs against manually validated ODEs.

class earm.tests.test_shen_models.**TestCui_Direct1**(*methodName='runTest'*)

Test the PySB version of the "direct 1" model from *[Cui2008]*.

Methods

test_odes()

Check the generated ODEs against manually validated ODEs.

class earm.tests.test_shen_models.**TestCui_Direct2**(*methodName='runTest'*)

Test the PySB version of the "direct 2" model from *[Cui2008]*.

Methods

test_odes()

Check the generated ODEs against manually validated ODEs.

class `earm.tests.test_shen_models.`**`TestHowells`**(*methodName='runTest'*)

Test the PySB version of the model from *[Howells2011]*.

Methods

`test_odes`()

Check the generated ODEs against manually validated ODEs.

The ODEs shown in the code match the ODEs listed in the paper, with the note that the parameter for spontaneous pore formation, ka_Bak_poly, has already been multiplied by 4 from its nominal value listed in the paper. This accounts for BNG's (appropriate) addition of the coefficients of 0.25 to the Bak polymerization forward reaction, due to the reaction being a homomeric binding reaction.

`test_parameters`()

Check that the values of the parameters in the PySB model match those of the original.

The parameter values shown in the test below have been validated against the list in Table 1 of Howells et al.

CHAPTER 5

References

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

[Albeck2008] Albeck, J. G., Burke, J. M., Spencer, S. L., Lauffenburger, D. A., and Sorger, P. K. (2008). Modeling a snap-action, variable-delay switch controlling extrinsic cell death. PLoS Biology, 6(12), 2831-2852. doi:10.1371/journal.pbio.0060299 PMID:19053173.

[Chen2007biophysj] Chen, C., Cui, J., Lu, H., Wang, R., Zhang, S., & Shen, P. (2007). Modeling of the role of a Bax-activation switch in the mitochondrial apoptosis decision. Biophysical Journal, 92(12), 4304-4315. doi:10.1529/biophysj.106.099606 PMID:17400705.

[Chen2007febs] Chen, C., Cui, J., Zhang, W., & Shen, P. (2007). Robustness analysis identifies the plausible model of the Bcl-2 apoptotic switch. FEBS letters, 581(26), 5143-5150. doi:10.1016/j.febslet.2007.09.063 PMID:17936275.

[Cui2008] Cui, J., Chen, C., Lu, H., Sun, T., & Shen, P. (2008). Two independent positive feedbacks and bistability in the Bcl-2 apoptotic switch. PLoS ONE, 3(1), e1469. doi:10.1371/journal.pone.0001469 PMID:18213378.

[Howells2011] Howells, C. C., Baumann, W. T., Samuels, D. C., & Finkielstein, C. V. (2011). The Bcl-2-associated death promoter (BAD) lowers the threshold at which the Bcl-2-interacting domain death agonist (BID) triggers mitochondria disintegration. Journal of Theoretical Biology. doi:10.1016/j.jtbi.2010.11.040 PMID:21130780.

[Lopez2013] Lopez, C. L.*, Muhlich, J. L.*, Bachman, J. A.*, & Sorger, P. K. (2013). PySB: A second-generation approach to programming biological models. Submitted.

e

earm.albeck_11b, 10
earm.albeck_11c, 10
earm.albeck_11d, 10
earm.albeck_11e, 10
earm.albeck_11f, 11
earm.albeck_modules, 14
earm.chen_biophys_j, 11
earm.chen_febs_direct, 11
earm.chen_febs_indirect, 11
earm.cui_direct, 11
earm.cui_direct1, 11
earm.cui_direct2, 11
earm.howells, 11
earm.lopez_direct, 10
earm.lopez_embedded, 10
earm.lopez_indirect, 10
earm.lopez_modules, 12
earm.mito.albeck_11b, 11
earm.mito.albeck_11c, 11
earm.mito.albeck_11d, 11
earm.mito.albeck_11e, 11
earm.mito.albeck_11f, 11
earm.mito.chen_biophys_j, 11
earm.mito.chen_febs_direct, 11
earm.mito.chen_febs_indirect, 11
earm.mito.cui_direct, 11
earm.mito.cui_direct1, 11
earm.mito.cui_direct2, 12
earm.mito.howells, 12
earm.mito.lopez_direct, 11
earm.mito.lopez_embedded, 11
earm.mito.lopez_indirect, 11
earm.shared, 20
earm.shen_modules, 17
earm.tests.test_albeck_models, 28
earm.tests.test_lopez_models, 27
earm.tests.test_models_net_gen, 27
earm.tests.test_shen_models, 30

e

earm.albeck_11b, 10
earm.albeck_11c, 10
earm.albeck_11d, 10
earm.albeck_11e, 10
earm.albeck_11f, 11
earm.albeck_modules, 14
earm.chen_biophys_j, 11
earm.chen_febs_direct, 11
earm.chen_febs_indirect, 11
earm.cui_direct, 11
earm.cui_direct1, 11
earm.cui_direct2, 11
earm.howells, 11
earm.lopez_direct, 10
earm.lopez_embedded, 10
earm.lopez_indirect, 10
earm.lopez_modules, 12
earm.mito.albeck_11b, 11
earm.mito.albeck_11c, 11
earm.mito.albeck_11d, 11
earm.mito.albeck_11e, 11
earm.mito.albeck_11f, 11
earm.mito.chen_biophys_j, 11
earm.mito.chen_febs_direct, 11
earm.mito.chen_febs_indirect, 11
earm.mito.cui_direct, 11
earm.mito.cui_direct1, 11
earm.mito.cui_direct2, 12
earm.mito.howells, 12
earm.mito.lopez_direct, 11
earm.mito.lopez_embedded, 11
earm.mito.lopez_indirect, 11
earm.shared, 20
earm.shen_modules, 17
earm.tests.test_albeck_models, 28
earm.tests.test_lopez_models, 27
earm.tests.test_models_net_gen, 27
earm.tests.test_shen_models, 30

A

add_caspase8() (in module earm.tests.test_albeck_models), 28
albeck_11b() (in module earm.albeck_modules), 15
albeck_11c() (in module earm.albeck_modules), 16
albeck_11d() (in module earm.albeck_modules), 16
albeck_11e() (in module earm.albeck_modules), 16
albeck_11f() (in module earm.albeck_modules), 16
all_monomers() (in module earm.albeck_modules), 15
apaf1_to_parp_monomers() (in module earm.albeck_modules), 15
assemble_pore_sequential() (in module earm.shared), 21
assemble_pore_spontaneous() (in module earm.shared), 21

B

Bax_tetramerizes() (in module earm.albeck_modules), 15
Bcl2_binds_Bax1_Bax2_and_Bax4() (in module earm.albeck_modules), 15
bind() (in module earm.shared), 21
bind_table() (in module earm.shared), 21

C

catalyze() (in module earm.shared), 20
catalyze_convert() (in module earm.shared), 21
check_generate_network() (in module earm.tests.test_models_net_gen), 27
chen_biophys_j() (in module earm.shen_modules), 18
chen_febs_direct() (in module earm.shen_modules), 19
chen_febs_indirect() (in module earm.shen_modules), 18
convert_odes() (in module earm.tests.test_shen_models), 30
convert_parameters() (in module earm.tests.test_shen_models), 31
cui_direct() (in module earm.shen_modules), 19
cui_direct1() (in module earm.shen_modules), 19
cui_direct2() (in module earm.shen_modules), 19

D

declare_initial_conditions() (in module earm.lopez_modules), 13
direct() (in module earm.lopez_modules), 14
displace() (in module earm.shared), 21
displace_reversibly() (in module earm.shared), 21

E

earm.albeck_11b (module), 10
earm.albeck_11c (module), 10
earm.albeck_11d (module), 10
earm.albeck_11e (module), 10
earm.albeck_11f (module), 11
earm.albeck_modules (module), 14
earm.chen_biophys_j (module), 11
earm.chen_febs_direct (module), 11
earm.chen_febs_indirect (module), 11
earm.cui_direct (module), 11
earm.cui_direct1 (module), 11
earm.cui_direct2 (module), 11
earm.howells (module), 11
earm.lopez_direct (module), 10
earm.lopez_embedded (module), 10
earm.lopez_indirect (module), 10
earm.lopez_modules (module), 12
earm.mito.albeck_11b (module), 11
earm.mito.albeck_11c (module), 11
earm.mito.albeck_11d (module), 11
earm.mito.albeck_11e (module), 11
earm.mito.albeck_11f (module), 11
earm.mito.chen_biophys_j (module), 11
earm.mito.chen_febs_direct (module), 11
earm.mito.chen_febs_indirect (module), 11
earm.mito.cui_direct (module), 11
earm.mito.cui_direct1 (module), 11
earm.mito.cui_direct2 (module), 12
earm.mito.howells (module), 12
earm.mito.lopez_direct (module), 11
earm.mito.lopez_embedded (module), 11
earm.mito.lopez_indirect (module), 11
earm.shared (module), 20
earm.shen_modules (module), 17
earm.tests.test_albeck_models (module), 28
earm.tests.test_lopez_models (module), 27

earm.tests.test_models_net_gen (module), 27
earm.tests.test_shen_models (module), 30
effectors_bind_anti_apoptotics() (in module earm.lopez_modules), 13
embedded() (in module earm.lopez_modules), 14

H

howells() (in module earm.shen_modules), 19

I

indirect() (in module earm.lopez_modules), 14

L

ligand_to_c8_monomers() (in module earm.albeck_modules), 14
lopez_pore_formation() (in module earm.lopez_modules), 14

M

matches_figure() (in module earm.tests.test_albeck_models), 29
momp_monomers() (in module earm.albeck_modules), 15
momp_monomers() (in module earm.lopez_modules), 13
momp_monomers() (in module earm.shen_modules), 18

O

observables() (in module earm.shared), 20
odes_match() (in module earm.tests.test_shen_models), 31
one_step_conv() (in module earm.shared), 21

P

pickle_lopez_models() (in module earm.tests.test_lopez_models), 28
plot_figure() (in module earm.tests.test_albeck_models), 29
pore_bind() (in module earm.shared), 21
pore_to_parp() (in module earm.albeck_modules), 15
pore_transport() (in module earm.shared), 21

R

rec_to_bid() (in module earm.albeck_modules), 15
run_figure_sim() (in module earm.tests.test_albeck_models), 28

S

sensitizers_bind_anti_apoptotics() (in module earm.lopez_modules), 13
shen_pore_transport() (in module earm.shen_modules), 18

T

tBid_activates_Bax_and_Bak() (in module earm.lopez_modules), 13
tBid_binds_all_anti_apoptotics() (in module earm.lopez_modules), 13
test_figure() (earm.tests.test_albeck_models.TestAlbeck11b method), 29
test_figure() (earm.tests.test_albeck_models.TestAlbeck11c method), 29
test_figure() (earm.tests.test_albeck_models.TestAlbeck11d method), 29
test_figure() (earm.tests.test_albeck_models.TestAlbeck11e method), 30
test_figure() (earm.tests.test_albeck_models.TestAlbeck11f method), 30
test_generate_network() (in module earm.tests.test_models_net_gen), 27
test_lopez_direct() (in module earm.tests.test_lopez_models), 28
test_lopez_embedded() (in module earm.tests.test_lopez_models), 28
test_lopez_indirect() (in module earm.tests.test_lopez_models), 28
test_odes() (earm.tests.test_shen_models.TestChenBiophysJ method), 31
test_odes() (earm.tests.test_shen_models.TestChenFEBS_Direct method), 32
test_odes() (earm.tests.test_shen_models.TestChenFEBS_Indirect method), 32
test_odes() (earm.tests.test_shen_models.TestCui_Direct method), 32
test_odes() (earm.tests.test_shen_models.TestCui_Direct1 method), 32
test_odes() (earm.tests.test_shen_models.TestCui_Direct2 method), 32
test_odes() (earm.tests.test_shen_models.TestHowells method), 33
test_parameters() (earm.tests.test_shen_models.TestChenBiophysJ method), 32
test_parameters() (earm.tests.test_shen_models.TestHowells method), 33
TestAlbeck11b (class in earm.tests.test_albeck_models), 29
TestAlbeck11c (class in earm.tests.test_albeck_models), 29
TestAlbeck11d (class in earm.tests.test_albeck_models), 29
TestAlbeck11e (class in earm.tests.test_albeck_models), 29
TestAlbeck11f (class in earm.tests.test_albeck_models), 30
TestChenBiophysJ (class in earm.tests.test_shen_models), 31
TestChenFEBS_Direct (class in

earm.tests.test_shen_models), 32
TestChenFEBS_Indirect (class in earm.tests.test_shen_models), 32
TestCui_Direct (class in earm.tests.test_shen_models), 32
TestCui_Direct1 (class in earm.tests.test_shen_models), 32
TestCui_Direct2 (class in earm.tests.test_shen_models), 32
TestHowells (class in earm.tests.test_shen_models), 32
translocate_tBid_Bax_BclxL() (in module earm.lopez_modules), 13