Libstructural Documentation

Release 3.1.0

Yosef Bedaso, Herbert Sauro

May 16, 2018

Contents

CHAPTER 1

Introduction

The structural analysis of stoichiometric networks is an important step in a number of computational methods in systems biology. The structure of a network based on the stoichiometry matrix is divided into two areas, structural constraints imposed by moiety conservation and constraints imposed by flux distributions at steady state. The former constraints have important applications in numerical methods for simulation and the analysis of control, while the later constraints have important applications in flux balance analysis. The LibStructural API provides a wide variety of methods that permit access to the constraint information in the stoichiometry matrix.

Stoichiometric Constraints

Moiety constraints concern the conservation of molecular subgroups in stoichiometric networks. Their existence results in dependencies among the model differential equations and the emergence of additional model parameters in the form of moiety mass totals. In the API we provide robust methods for extracting the constraint information and include specific methods to obtain for example the number of moiety cycles, the number of independent and dependent species and all the pertinent matrices such as the link matrix, reduced stoichiometry matrix etc. In addition to moiety constraints the library also provides robust methods for determining the flux constraints in a model. These include the dependent and independent flux, and the K matrix (and corresponding terms) that relates the two.

All matrices provided by the API are fully labeled with reaction and species labels. The API can accept models either directly from standard SBML or by specifying the stoichiometry matrix. In the case of SBML the species and reaction labels are obtained directly from the SBML otherwise they are entered manually.

Further and more detailed information on this work can be found in Reder (1988), Sauro and Ingalls (2004), Vallabhajosyula et al. (2005).

CHAPTER 2

Installation on Python

Currently we support Python 2.7+ and 3.6+ for both Windows (64 bit and 32-bit) and MacOS. Follow the steps below to install LibStructural.

By far the easiest way to get LibStructural working is to install Tellurium (tellurium.analogmachine.org) and type at the Tellurium Python console the instruction:

```
import tellurium as te
te.installPackage ('structural')
```

This should work for Windows and Mas OSX version of Tellurium. That completes the installation.

If you want to install LibStructural on the standard python distribution found at python.org then follow the instructions below:

2.1 Dependencies

For those who want to install LibStructural on other distributions including the standard distibution at python.org, you'll need the following instructions.

LibStructural depends on numpy and scipy. If you don't have numpy and scipy installed you'll need to install it. To check if you have numpy and scipy installed type the following at the Python console:

```
import numpy
import scipy
```

If there are no errors then you have numpy and scipy installed. If there are errors follow the instructions below:

NOTE: To install the dependencies you will need to have pip available with your python distribution. Pip is already installed if you're using Python 2 >=2.7.9 or Python 3 >=3.4 binaries downloaded from python.org (by default, in **../PythonXX/Scripts** directory). If you don't have pip installed, go to the link below to install pip.

https://pip.pypa.io/en/stable/installing/

2.1.1 Numpy and scipy on Windows 7/8/10

Option 1: You can simply run **pip install numpy** from your command line (make sure to cd to your python directory).

Option 2: If option 1 causes errors, follow the following work around:

Download the numpy binaries from:

https://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy

If you are using a 32-bit Python download: numpy-1.14.0+mkl-cp37-cp37m-win32.whl

If you are using a 64-bit Python download: numpy-1.14.0+mkl-cp37-cp37m-win_amd64.whl

Copy which ever file is appropriate to c:\PythonXX\Scripts where XX is your python version eg 27.

At the windows command line, cd to c:\PythonXX\Scripts and type: pip install numpy-1.14..etc

This should install numpy, if it doesn't find someone who can help you.

2.1.2 Numpy on Max OSX

For the Mac installation please see: https://scipy.org/install.html

2.2 Using Pip to Install LibStructural

Once you have numpy installed you can use a standard pip installation to install Libstructural. See below if you are using Tellurium.

LibStructural is available via PyPI. To install LibStructural using pip, run following line on at your terminal window. If you are doing this on Windows you will need to make sure you are in c:\PythonXX\Scripts directory, otherwise it is likely it won't find pip.

```
pip install structural
```

2.3 From Python the Tellurium Console

If you are using Tellurium (tellurium.analogmachine.org) then you can install structural directly from the Python console:

```
import tellurium as te
te.installPackage ('structural')
```

2.4 From Python Terminal (Experts)

If you are familar with obtaining your code from the GitHub repo you can follow these instructions.

1 - Use git and type the following command in the console or terminal which will clone the source code under Libstructural folder.

```
git clone https://github.com/yosefmaru/Libstructural.git
```

2 - Open your console or terminal. Go to package's root directory and Run the installer script by using the following command line. This will install **structural** to the Python release associated with the console or terminal you are using.

```
python setup.py install
```

3 - Test the Libstructural by importing it in Python.

```
import structural
```

If you have trouble importing the module with the setup script, check to see if there are multiple Python installations on your machine and also check the output of the setup script to see which version of Python is the install target.

CHAPTER 3

Building From Source

The LibStructural API library can be built on your machine from the source code found on github. To build LibStructural, you will need:

- The LibStructural source code and dependencies from github (you will need git to clone/download the repo).
- libSBML dependencies package. Make sure to get the vs15_release_x64 version.
- libSBML version 5.15.0 source code. Get the "libSBML-5.15.0-core-src.zip" version.
- CMake version 3.9.3 for generating solution files.
- Visual Studio 2017 for Windows or Xcode for MacOS
- Python 2.7 for generating a python wrapper (optional)
- SWIG for generating python wrappers in Windows (optional).

NOTE: We provide libSBML and CLAPACK dependency libraries with our repo for both Mac and Windows. If you wish to build your own libSBML or CLAPACK dependency libraries follow the instructions on "Building libSBML" and "Building CLAPACK" sections.

3.1 Windows

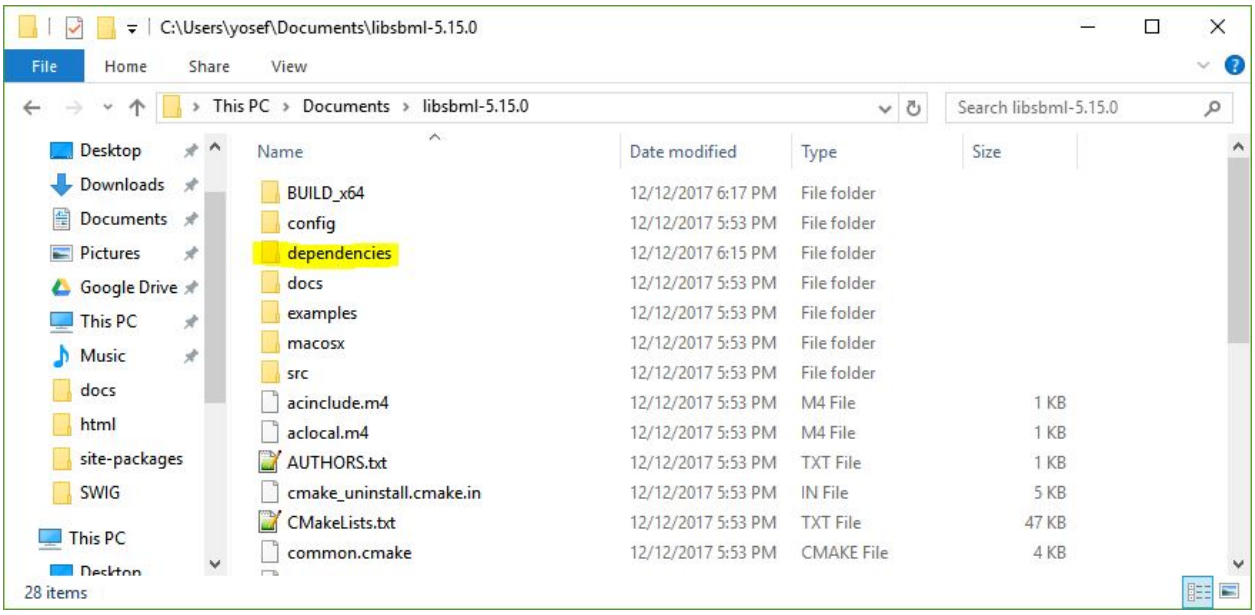
First, clone/download the Libstructural source code from the github repo by running the following command from the command line or git software:

```
git clone https://github.com/yosefmaru/Libstructural
```

Prior to building LibStructural on Windows, binaries for libSBML needs to be generated first. To do this you, will need a libSBML dependencies package and source code. An issue on Windows concerns the dependency libraries on which libSBML depends. CMake will try to find all the dependencies for the default options, which leads to problems if they do not exist.

1. Extract the dependencies folder you downloaded into the same folder where you unpacked the libSBML source distribution. CMake will look for these dependencies in a folder called "dependencies" directly below the libSBML root folder.

2. rename the extracted libsbml dependencies folder from it's default name to "dependencies".



3. Once you have the libSBML sources and the dependency libraries unpacked on your system, start up the CMake graphical user interface (GUI).

3.1.1 Building libSBML

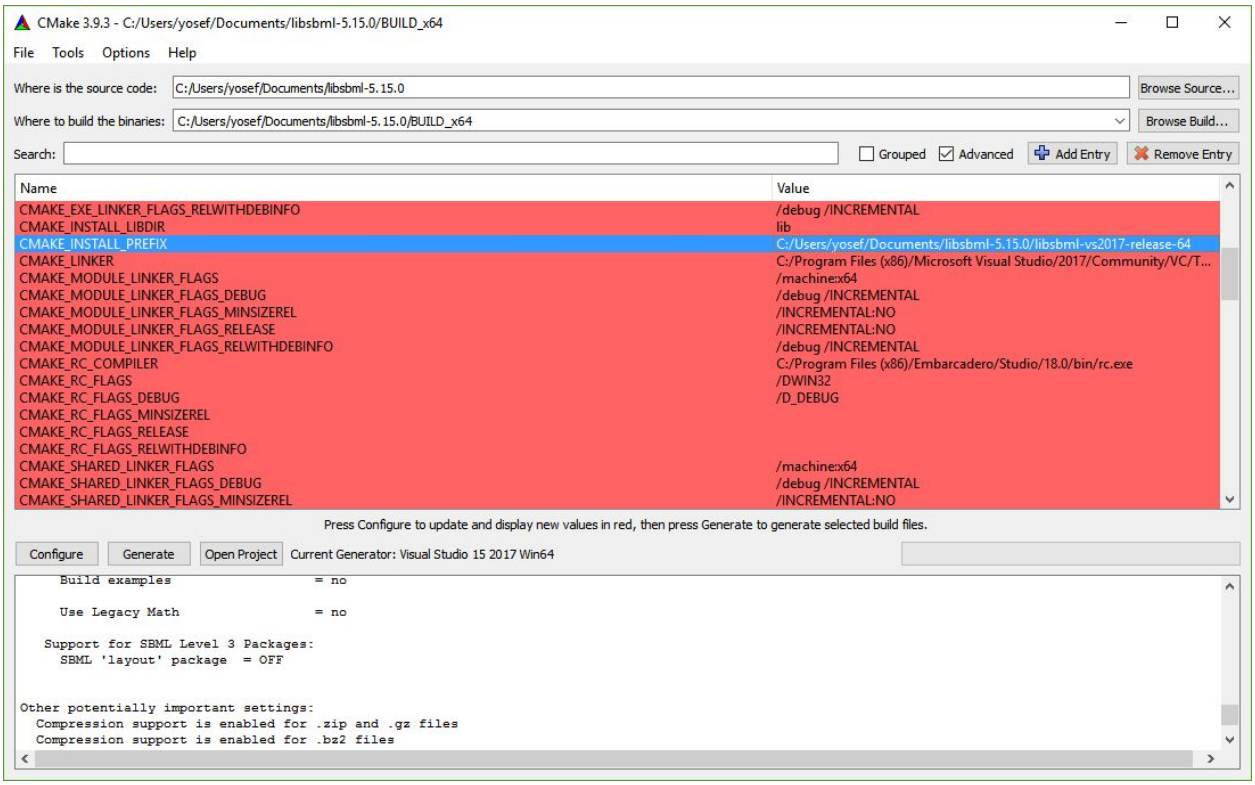
A detailed step-by-step instruction can be found on this link on how to generate a solution file using CMake and build libSBML libraries on Visual Studio.

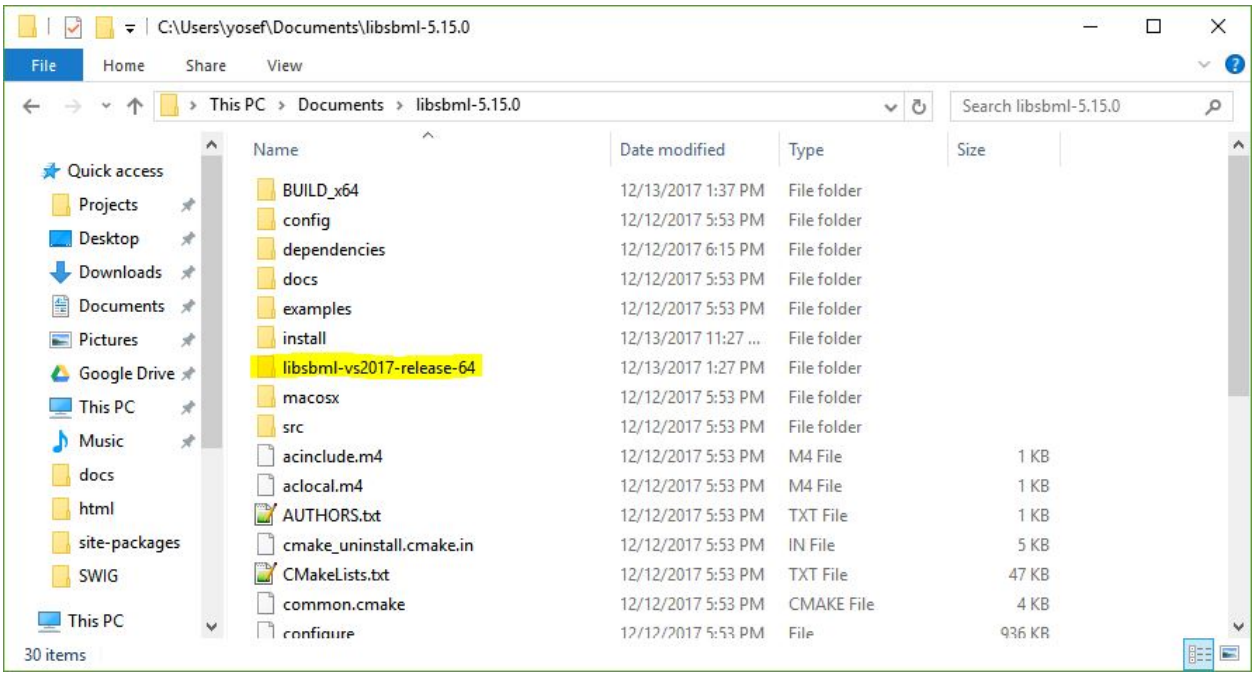
1. To be consistent with building LibStructural instructions, make sure to select "Visual Studio 15 2017 Win64" when selecting a generator on CMake.
2. Copy the directory inside "Where is the source code" inquiry to the CMAKE_INSTALL_PREFIX option and add "/libsbml-vs2017-release-64" to it as shown below.
3. After generating the solution file make sure you build with a **Release** configuration, instead of **Debug** configuration on visual studio.
4. Build the solution file from the "INSTALL" target (i.e. on Visual Studio, right click on "INSTALL" and select Build). This target will install the compiled binaries to the directory specified on the CMAKE_INSTALL_PREFIX option, which is "...libsbml-vs2017-release-64"
5. Once you complete building libSBML, copy the libsbml-vs2017-release-64 folder to "...Libstructural/dependencies" that you cloned/downloaded from github. This step is very important since the LibStructural CMakeLists configuration that generates the solution file looks for libSBML libraries inside a folder named "libsbml-vs2017-release-64". Thus, you should have "libsbml-vs2017-release-64" and "third-party" folders under dependencies.

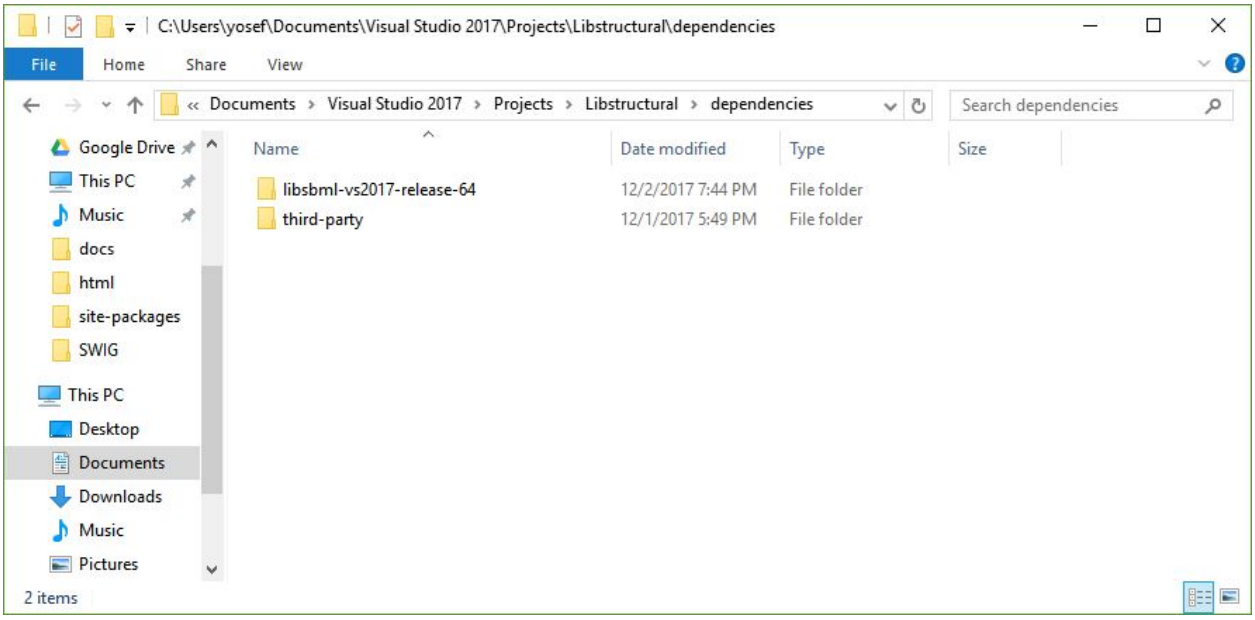
The next step is building LibStructural.

3.1.2 Building LibStructural

1. Open CMake-gui.


CMake 3.9.3 - C:/Users/yosef/Documents/libsbml-5.15.0/BUILD_x64
File Tools Options Help
Where is the source code: C:/Users/yosef/Documents/libsbml-5.15.0
Browse Source...
Where to build the binaries: C:/Users/yosef/Documents/libsbml-5.15.0/BUILD_x64
Browse Build...
Search:
Grouped
Advanced
Add Entry
Remove Entry
Name
Value
CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO /debug /INCREMENTAL
CMAKE_INSTALL_LIBDIR lib
CMAKE_INSTALL_PREFIX C:/Users/yosef/Documents/libsbml-5.15.0/libsbml-vs2017-release-64
CMAKE_LINKER C:/Program Files (x86)/Microsoft Visual Studio/2017/Community/VC/T...
CMAKE_MODULE_LINKER_FLAGS /machine:x64
CMAKE_MODULE_LINKER_FLAGS_DEBUG /debug /INCREMENTAL
CMAKE_MODULE_LINKER_FLAGS_MINSIZEREL /INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELEASE /INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELWITHDEBINFO /debug /INCREMENTAL
CMAKE_RC_COMPILER C:/Program Files (x86)/Embarcadero/Studio/18.0/bin/rc.exe
CMAKE_RC_FLAGS /DWIN32
CMAKE_RC_FLAGS_DEBUG /D_DEBUG
CMAKE_RC_FLAGS_MINSIZEREL
CMAKE_RC_FLAGS_RELEASE
CMAKE_RC_FLAGS_RELWITHDEBINFO
CMAKE_SHARED_LINKER_FLAGS /machine:x64
CMAKE_SHARED_LINKER_FLAGS_DEBUG /debug /INCREMENTAL
CMAKE_SHARED_LINKER_FLAGS_MINSIZEREL /INCREMENTAL:NO
Press Configure to update and display new values in red, then press Generate to generate selected build files.
Configure
Generate
Open Project
Current Generator: Visual Studio 15 2017 Win64
Build examples = no
Use Legacy Math = no
Support for SBML Level 3 Packages:
SBML 'layout' package = OFF
Other potentially important settings:
Compression support is enabled for .zip and .gz files
Compression support is enabled for .bz2 files


C:\Users\yosef\Documents\libsbml-5.15.0
File Home Share View
This PC › Documents › libsbml-5.15.0
Search libsbml-5.15.0
Quick access
Projects
Desktop
Downloads
Documents
Pictures
Google Drive
This PC
Music
docs
html
site-packages
SWIG
This PC
Name
Date modified
Type
Size
BUILD_x64 12/13/2017 1:37 PM File folder
config 12/12/2017 5:53 PM File folder
dependencies 12/12/2017 6:15 PM File folder
docs 12/12/2017 5:53 PM File folder
examples 12/12/2017 5:53 PM File folder
install 12/13/2017 11:27 ... File folder
libsbml-vs2017-release-64 12/13/2017 1:27 PM File folder
macosx 12/12/2017 5:53 PM File folder
src 12/12/2017 5:53 PM File folder
acinclude.m4 12/12/2017 5:53 PM M4 File 1 KB
aclocal.m4 12/12/2017 5:53 PM M4 File 1 KB
AUTHORS.txt 12/12/2017 5:53 PM TXT File 1 KB
cmake_uninstall.cmake.in 12/12/2017 5:53 PM IN File 5 KB
CMakeLists.txt 12/12/2017 5:53 PM TXT File 47 KB
common.cmake 12/12/2017 5:53 PM CMAKE File 4 KB
30 items

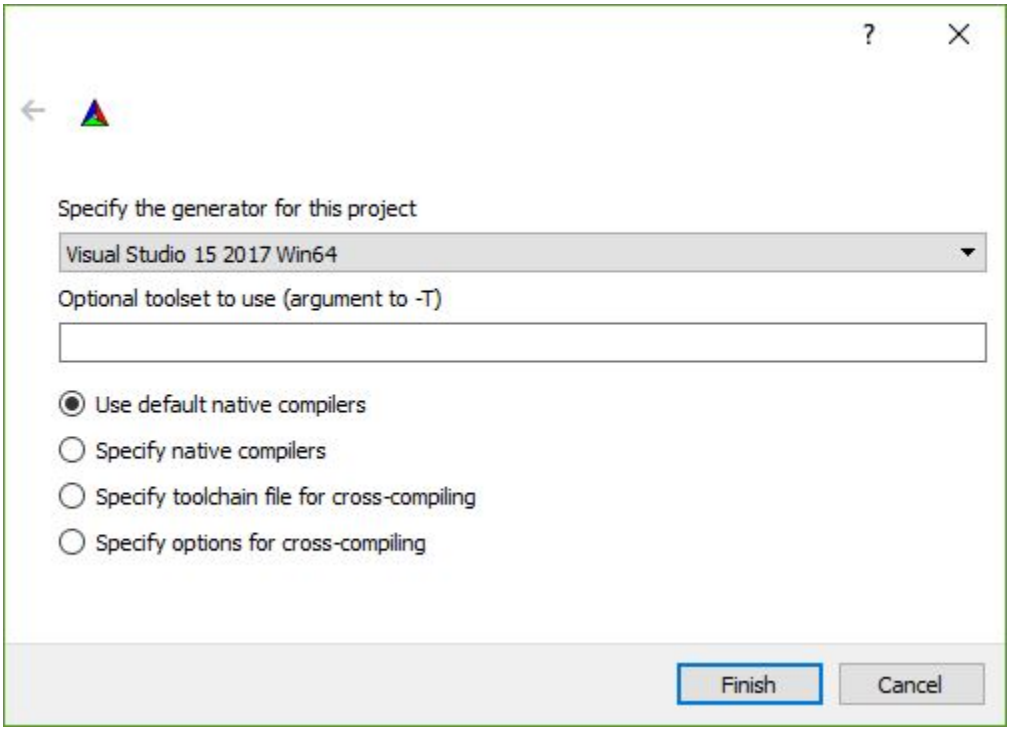


2. Click on the **Browse Source...** button and navigate to the directory where you cloned/downloaded the LibStructural folder on your file system.
3. For the "Where to build the binaries" inquiry, use the same directory, but add /BUILD_x64 to it.
4. On the CMake-gui click on **Configure**. A prompt will ask you to create a "BUILD_x64" folder inside LibStructural. Click on **Yes**.
5. Next, it will ask you to specify a generator for the project and what compiler to use as shown below. For the generator, select "Visual Studio 15 2017 Win64".
6. Select "Use default native compilers" option and click on **Finish**.
7. If you wish to generate a python wrapper, you will have to checkmark the "ENABLE_PYTHON_WRAPPER" option.
8. Click on **Configure** again and an error will appear in red on the top, asking for the location of a SWIG executable file.
9. Click on the "SWIG_EXECUTABLE-NOTFOUND" and copy or navigate to the swig.exe directory. The swig.exe file will be in the SWIG folder you downloaded.

10. Click on **Generate**, and a LibStructural.sln file will be stored inside the BUILD_x64 folder. **NOTE:** If you don't want to generate a python wrapper, simply click **Configure** and then **Generate** after step 2.

11. To build the libraries, open the solution file "LibStructural.sln" located in BUILD_x64 folder using Visual Studio 2017.
12. On Visual Studio, set the build target to "Release" instead of "Debug" with "x64" configuration.
13. Right click on **ALL_BUILD** and click on Build.

The built libraries and binaries will be stored in ".../BUILD_x64/LibStructural/Release" directory. If you chose to generate a python wrapper, it will be located inside INSTALL_x64/wrapper/structural.

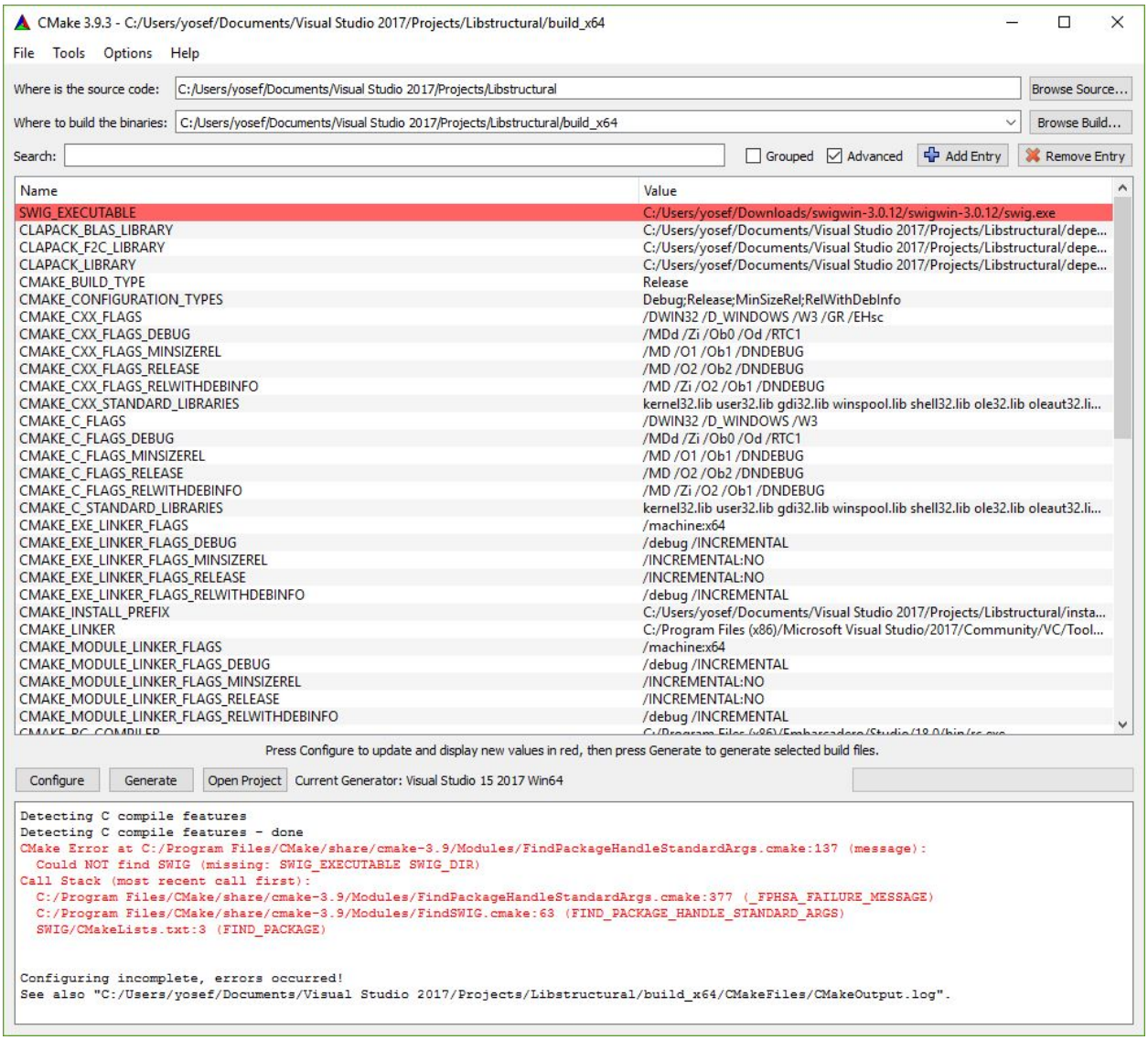


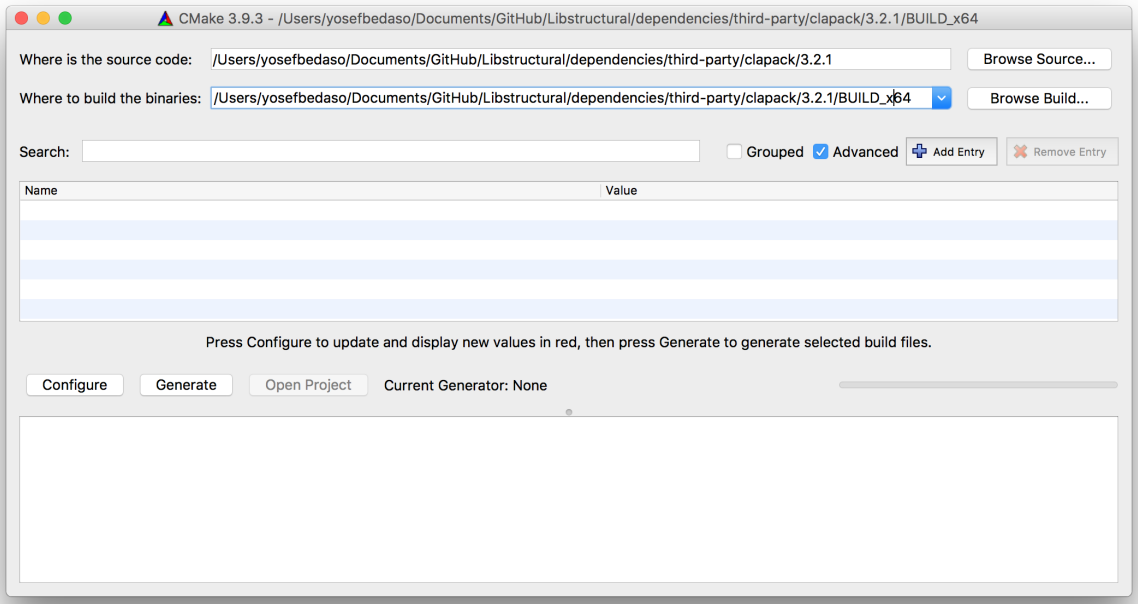
3.2 Mac

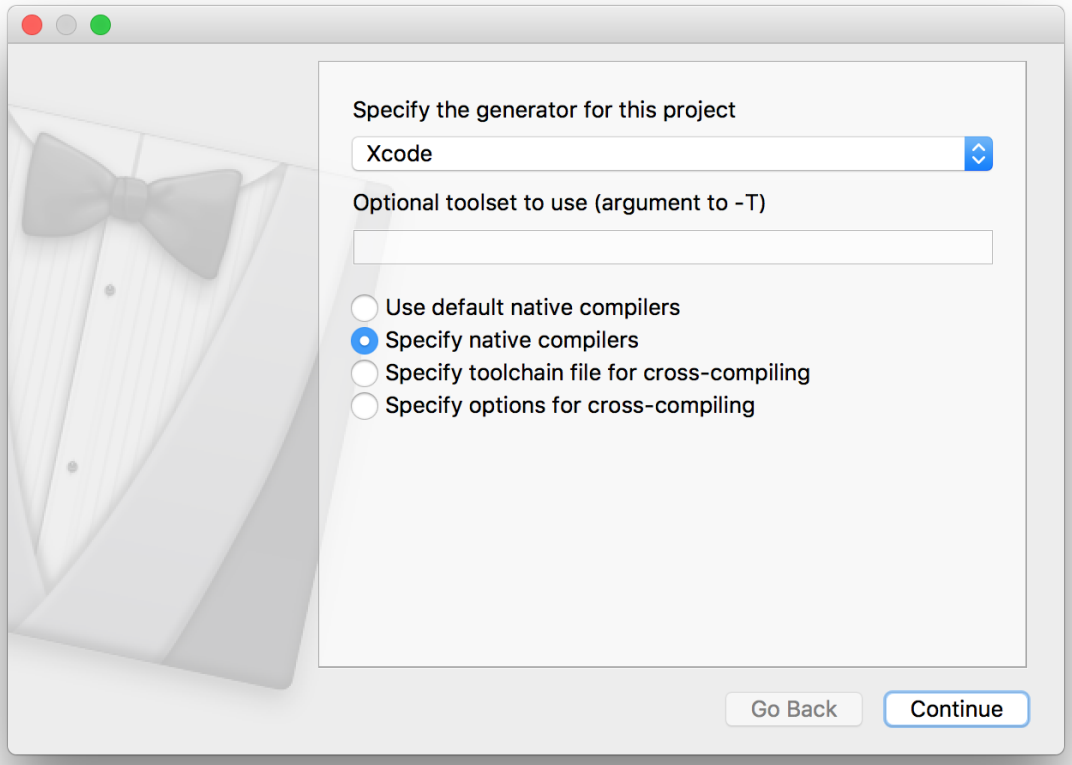
For Mac, you will have to generate CLAPACK libraries in addition to libSBML. You need to generate MacOS compatible dependency libraries needed for building LibStructural. The CLAPACK source code is included in the LibStructural distribution that is cloned/downloaded from github. It is located inside the directory "...Libstructural/dependencies/third-party". However, the libSBML source code is the one you should download from sourceforge.

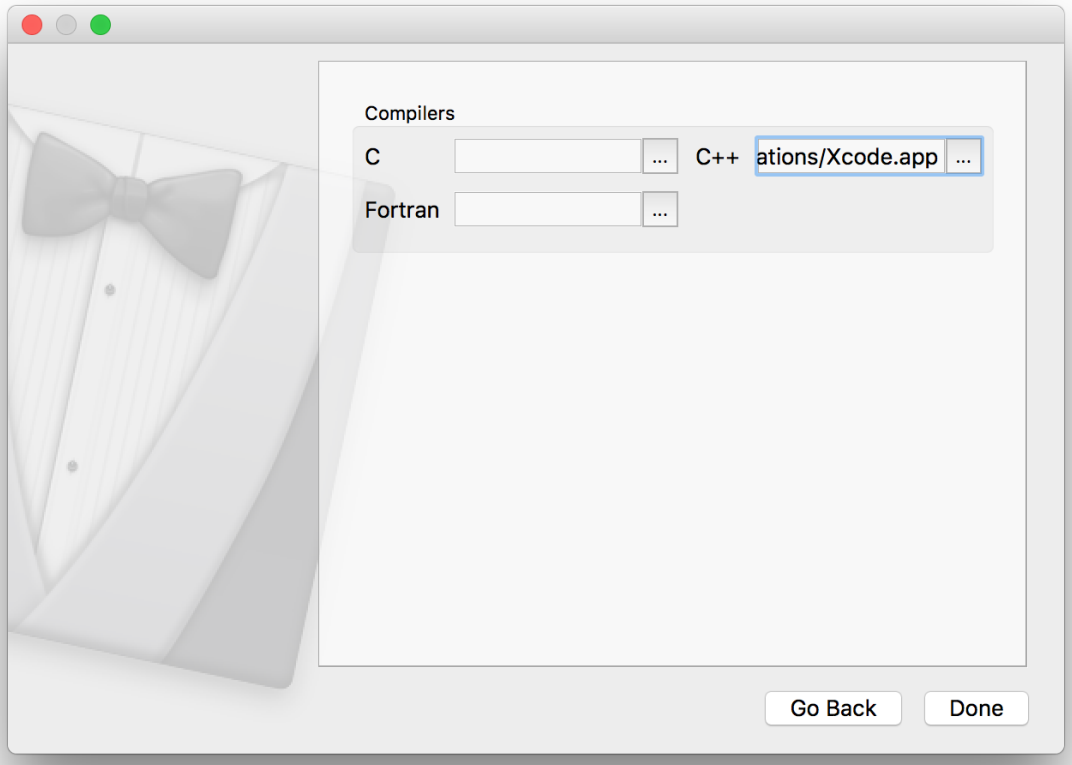
3.2.1 Building CLAPACK

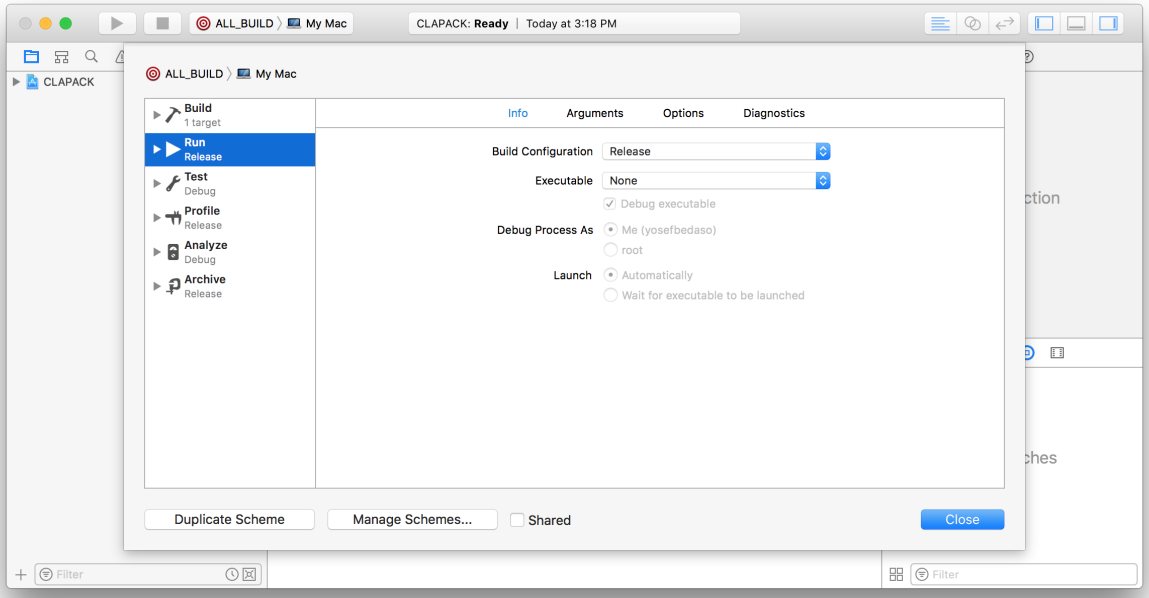
1. Open CMake-gui.
2. Click on the **Browse Source...** button and navigate to ".../dependencies/third-party/clapack/3.2.1" inside the Libstructural folder you cloned/downloaded.
3. For the "Where to build the binaries" inquiry, use the same directory, but add /BUILD_x64 to it as shown below.
4. Click on **Configure**. CMake will ask for what generator and compiler to use.
5. Find and select "Xcode" from the dropdown menu.
6. Select "Specify native compilers" and click on **Continue**.
7. On the next window, under C++ box, click on the three dots and navigate to your Xcode application. It should be located inside the Applications folder. If not, you need to move it to the Application folder. Click on **Done**.
8. Click on **Configure** and then **Generate.
9. Your Xcode project file will be generated inside "BUILD_x64" folder, or you can simply click on **Open Project** on CMake to automatically launch the project in Xcode.
10. Once Xcode finishes loading/indexing the project, go to **Product -> Scheme -> Edit Scheme** (keyboard shortcut: **Command + Shift + <**). You will need to set the Build Configuration to "Release".


CMake 3.9.3 - C:/Users/yosef/Documents/Visual Studio 2017/Projects/Libstructural/build_x64
File Tools Options Help
Where is the source code: C:/Users/yosef/Documents/Visual Studio 2017/Projects/Libstructural
Browse Source...
Where to build the binaries: C:/Users/yosef/Documents/Visual Studio 2017/Projects/Libstructural/build_x64
Browse Build...
Search:
Grouped
Advanced
Add Entry
Remove Entry
Name
Value
SWIG_EXECUTABLE C:/Users/yosef/Downloads/swigwin-3.0.12/swigwin-3.0.12/swig.exe
CLAPACK_BLAS_LIBRARY C:/Users/yosef/Documents/Visual Studio 2017/Projects/Libstructural/depe...
CLAPACK_F2C_LIBRARY C:/Users/yosef/Documents/Visual Studio 2017/Projects/Libstructural/depe...
CLAPACK_LIBRARY C:/Users/yosef/Documents/Visual Studio 2017/Projects/Libstructural/depe...
CMAKE_BUILD_TYPE Release
CMAKE_CONFIGURATION_TYPES Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_CXX_FLAGS /DWIN32 /D_WINDOWS /W3 /GR /EHsc
CMAKE_CXX_FLAGS_DEBUG /MDd /Zi /Ob0 /Od /RTC1
CMAKE_CXX_FLAGS_MINSIZEREL /MD /O1 /Ob1 /DNDEBUG
CMAKE_CXX_FLAGS_RELEASE /MD /O2 /Ob2 /DNDEBUG
CMAKE_CXX_FLAGS_RELWITHDEBINFO /MD /Zi /O2 /Ob1 /DNDEBUG
CMAKE_CXX_STANDARD_LIBRARIES kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.li...
CMAKE_C_FLAGS /DWIN32 /D_WINDOWS /W3
CMAKE_C_FLAGS_DEBUG /MDd /Zi /Ob0 /Od /RTC1
CMAKE_C_FLAGS_MINSIZEREL /MD /O1 /Ob1 /DNDEBUG
CMAKE_C_FLAGS_RELEASE /MD /O2 /Ob2 /DNDEBUG
CMAKE_C_FLAGS_RELWITHDEBINFO /MD /Zi /O2 /Ob1 /DNDEBUG
CMAKE_C_STANDARD_LIBRARIES kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.li...
CMAKE_EXE_LINKER_FLAGS /machine:x64
CMAKE_EXE_LINKER_FLAGS_DEBUG /debug /INCREMENTAL
CMAKE_EXE_LINKER_FLAGS_MINSIZEREL /INCREMENTAL:NO
CMAKE_EXE_LINKER_FLAGS_RELEASE /INCREMENTAL:NO
CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO /debug /INCREMENTAL
CMAKE_INSTALL_PREFIX C:/Users/yosef/Documents/Visual Studio 2017/Projects/Libstructural/insta...
CMAKE_LINKER C:/Program Files (x86)/Microsoft Visual Studio/2017/Community/VC/Tool...
CMAKE_MODULE_LINKER_FLAGS /machine:x64
CMAKE_MODULE_LINKER_FLAGS_DEBUG /debug /INCREMENTAL
CMAKE_MODULE_LINKER_FLAGS_MINSIZEREL /INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELEASE /INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELWITHDEBINFO /debug /INCREMENTAL
Press Configure to update and display new values in red, then press Generate to generate selected build files.
Configure
Generate
Open Project
Current Generator: Visual Studio 15 2017 Win64
Detecting C compile features
Detecting C compile features - done
CMake Error at C:/Program Files/CMake/share/cmake-3.9/Modules/FindPackageHandleStandardArgs.cmake:137 (message):
Could NOT find SWIG (missing: SWIG_EXECUTABLE SWIG_DIR)
Call Stack (most recent call first):
C:/Program Files/CMake/share/cmake-3.9/Modules/FindPackageHandleStandardArgs.cmake:377 (_FPHSA_FAILURE_MESSAGE)
C:/Program Files/CMake/share/cmake-3.9/Modules/FindSWIG.cmake:63 (FIND_PACKAGE_HANDLE_STANDARD_ARGS)
SWIG/CMakeLists.txt:3 (FIND_PACKAGE)
Configuring incomplete, errors occurred!
See also "C:/Users/yosef/Documents/Visual Studio 2017/Projects/Libstructural/build_x64/CMakeFiles/CMakeOutput.log".


CMake 3.9.3 - /Users/yosefbedaso/Documents/GitHub/Libstructural/dependencies/third-party/clapack/3.2.1/BUILD_x64
Where is the source code:
/Users/yosefbedaso/Documents/GitHub/Libstructural/dependencies/third-party/clapack/3.2.1
Browse Source...
Where to build the binaries:
/Users/yosefbedaso/Documents/GitHub/Libstructural/dependencies/third-party/clapack/3.2.1/BUILD_x64
Browse Build...
Search:
Grouped
Advanced
Add Entry
Remove Entry
Name
Value
Press Configure to update and display new values in red, then press Generate to generate selected build files.
Configure
Generate
Open Project
Current Generator: None


Specify the generator for this project
Xcode
Optional toolset to use (argument to -T)
Use default native compilers
Specify native compilers
Specify toolchain file for cross-compiling
Specify options for cross-compiling
Go Back
Continue


Compilers
C
...
C++
ations/Xcode.app
...
Fortran
...
Go Back
Done



11. Click on **Product -> Build** or the Play button.
12. The libraries will be stored inside BUILD_x64. There are three libraries you will need to locate:
    - **libblas.a** which will be found in the directory "…/BUILD_x64/BLAS/SRC/Release".
    - **libf2c.a** which will be found in the directory "…/BUILD_x64/F2CLIBS/libf2c/Release".
    - **liblapack.a** inside "…/BUILD_x64/SRC/Release".
13. Copy these three libraries to "…/Libstructural/dependencies/third-party/clapack/3.2.1/LIB".

3.2.2 Building libSBML

You will use the same steps as shown above with minor changes. Your source code is a libSBML package you downloaded from sourceforge.

1. On CMake, locate the folder to fill "Where is the source" inquiry (i.e. "…/Downloads/libsbml-5.15.0"). Then click on **Browse Build…* and use the same folder directory, but add "/BUILD_x64" to it. Click on **Configure**
2. Before generating the project file, copy the directory inside "Where is the source code" inquiry to the CMAKE_INSTALL_PREFIX option and add "/libsbml-vs2017-release-64" to it.
3. Click on **Configure** then **Generate**.
4. Open the Xcode project generated and go to **Product -> Scheme** to select "install".
5. Go to **Product -> Scheme -> Edit Scheme** (keyboard shortcut: Command + Shift + <). You will need to set the Build Configuration to "Release".
6. Click on **Product -> Build** or the Play button (keyboard shortcut: Command + B).
7. After building succeeds, a "libsbml-vs2017-release-64"" folder will be created inside "…/libsbml-5.15.0". Copy this folder in to "…/Libstructural/dependencies" directory. This step is very important since the Lib-

Structural CMakeLists configuration that generates the Xcode project file looks for libSBML libraries inside a folder named "libsbml-vs2017-release-64".

3.2.3 Building LibStructural

1. Open CMake-gui. For the "Where is the source code" inquiry, click on **Browse Source** and navigate to the directory where LibStructural is located at. For the "Where to build the binaries" inquiry, use the same directory, but add "/BUILD_x64" to it.
2. Follow steps 4-7 from the "Building CLAPACK" section.
3. If you chose to build a python wrapper, checkmark the "ENABLE_PYTHON_WRAPPER" value and configure again. If no errors are raised click on **Generate**. **Note:** You don't need to download SWIG as it is included in with your Mac OS.
4. Follow step three from the "Building CLAPACK" section.
5. Once Xcode finishes loading/indexing the project, go to **Product -> Scheme -> Edit Scheme** (keyboard shortcut: Command + Shift + <). You will need to set the Build Configuration to "Release". If you chose to generate python wrapper from step three, go to **Product -> Scheme** and select "install". Otherwise, click on **Product -> Build** or the Play button (keyboard shortcut: Command + B).
6. The built libraries and binaries will be stored in ". . . /BUILD_x64/LibStructural/Release" directory. If you chose to generate a python wrapper, the wrapper will be stored inside ". . . /INSTALL_x64/wrapper/structural".

CHAPTER 4

Getting Started with LibStructural

The following examples demonstrate how to load a biochemical reaction network using the LibStructural API. A model should be available in at least one of the following formats: SBML model file (.xml format), or a 2D array. SBML can either be loaded as a string or directly from a file.

4.1 Testing LibStructural

To test the LibStructural module, you can use the **runLibstructTests()** method. This will print out a summary of various tests. These include: testing the integrity of the model loading methods, testing for error messages and conservation analysis methods. Thirty one toy models and a Glycolysis/Gluconeogenesis SBML model (BMID000000101155) are used for testing. For example:

```
import structural
ls = structural.LibStructural()
ls.runLibstructTests()
```

In addition, the test method **runElementaryModeTests()** runs an internal test suite on 31 toy models anyalysing the integrity of the elementary modes returned by **getElementaryModesInteger()** and **getElementaryModesDouble()**.

```
import structural
ls = structural.LibStructural()
ls.runElementaryModeTests()
```

The following sections describe different ways of loading a model into Libstructural. Once a model is loaded it will automatically call `analyzeWithQR` for analysing the stoichiometry matrix. At this point a summary of the analysis can be obtained by calling **getSummary()**:

```
ls.getSummary()
```

4.2 Loading a Model

To load a model into LibStructural, a LibStructural variable must first be created. All methods call are then routed via this variable.

```
import structural
ls = structural.LibStructural()
```

4.2.1 Loading a Model from a File

A model can be loaded from an SBML file using the **loadSBMLFromFile()** method. For example:

```
ls.loadSBMLFromFile("iYO844.xml") # Pass file path if file is in different directory
```

4.2.2 Loading a model from a string

If a model is available as an SBML string, use the following code:

```
ls.loadSBMLFromString(sbmlString)
```

where **sbmlString** has been previously assigned an SBML string. SBML strings can be obtained either by loading the contents of a file than contains SBML, creating an SBML model via simpleSBML or libSBML, or more commonly by converting an Antimony description of a model into SBML.

4.2.3 Loading a Model from a Stoichiometric Matrix

Models can also be loaded by specifying the stoichiometry matrix directly as an array:

```
ls = structural.LibStructural()
matrix = [[  1, -1, -1], [  0, -1,  1], [  0,  1, -1]] # matrix can be a numpy 2d␣
↪array
ls.loadStoichiometryMatrix(matrix)
```

The load command will also by default add reaction ids of the form '_Jx' and species ids of the form 'Sx'. To override these default names, see the section below.

4.2.3.1 Assigning Reaction and Species Ids

When loading a model from a stoichiometry matrix, reactions and species Ids can be changed form their default values as follows.

```
import structural
ls = structural.LibStructural()
matrix = [[  1, -1, -1], [  0, -1,  1], [  0,  1, -1]] # matrix can also be a numpy␣
↪2d array
ls.loadStoichiometryMatrix(matrix)
print ls.getStoichiometryMatrix()
print ls.getFloatingSpeciesIds()
print ls.getReactionIds()

# load new Ids
```

(continues on next page)

(continued from previous page)

```
ls.loadSpeciesIdsWithValues (['a', 'b', 'c'], [0, 0, 0]) # The array length for both␣
↪ids list and values list should be equal to the number of species
ls.loadReactionIdsWithValues (['F1', 'F2', 'F3'],[0, 0, 0])

# Reanalyze with the new labels
ls.analyzeWithQR()

print ls.getFloatingSpeciesIds()
print ls.getReactionIds()
```

4.2.4 Loading a model Using the Antimony Model Description Language

If you use tellurium you can load a model by converting an antimony string into a SBML string. For example:

```
import structural
import tellurium as te

r = te.loada('''
    // Reactions. All reactions have the dummy rate law 'v'
    // since we're not interested in dynamic simulation.
    J1: S1 -> S2; v;
    J2: -> S3; v;
    J3: S3 -> S1; v;

    # Initialize species
    S1 = 10; S2 = 10; S3 = 10;
    v = 0;
''')

sbmlstr = r.getSBML() # this creates an SBML string from the antimony model, r.
ls = structural.LibStructural()
ls.loadSBMLFromString(sbmlstr)
print(ls.getSummary())

# A libRoadRunner model can be converted into SBML file as well
r.exportToSBML('Test_model.xml') # creates an xml file in the current directory
ls = structural.LibStructural()
ls.loadSBMLFromFile('Test_model.xml') # loads the xml file from the current directory
print(ls.getSummary())
```

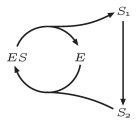
4.3 Structural Analysis

The following examples show some of LibStructural's methods on two different models. The models were generated using Tellurium by converting the antimony description language into an SBML string.

4.3.1 Conserved Metabolic Network

Below is a network diagram that shows two interlinked conserved cycles: S1 + S2 + ES and ES + E:

To generate an SBML string and load the model to LibStructural, run:




```
import structural
import tellurium as te

r = te.loada('''
    // Reactions:
    J1: ES -> S1 + E; v;
    J2: S2 + E -> ES; v;
    J3: S1 -> S2; v;

    // Species Initialization
    S1 = 10; S2 = 10; ES = 10; E = 10;
    v = 0;
''')

sbmlstr = r.getSBML() # this creates an SBML string from the antimony model, r.
ls = structural.LibStructural()
ls.loadSBMLFromString(sbmlstr)
```

Once the model is loaded we can run the getSummary function to get the analysis result of the analyzeWithQR function. NOTE: when loading a model on LibStructural, analyzeWithQR is called automatically.

```
print(ls.getSummary()) # Prints out if the model is passed some internal structural␣
↪validation tests.
```

Which returns:

```
Out[1]:
----------------------------------------------------------------
STRUCTURAL ANALYSIS MODULE : Results
----------------------------------------------------------------
Size of Stochiometric Matrix: 4 x 3 (Rank is  2)
Nonzero entries in Stochiometric Matrix: 8  (66.6667% full)

Independent Species (2) :
ES, S1

Dependent Species (2) :
E, S2

L0 : There are 2 dependencies. L0 is a 2x2 matrix.

Conserved Entities
1:  + ES + E
2:  + ES + S1 + S2
```

To see the internal test suites results and the types of the tests, run:

```
print(ls.validateStructuralMatrices()) # Prints out if the model passed some internal␣
↪structural validation tests.

# see what tests were run, call ls.getTestDetails()
```

(continues on next page)

(continued from previous page)

```
tests = ls.getTestDetails()
print(tests)
```

```
Out[1]:
('Pass', 'Pass', 'Pass', 'Pass', 'Pass', 'Pass')
Testing Validity of Conservation Laws.

Passed Test 1 : Gamma*N = 0 (Zero matrix)
Passed Test 2 : Rank(N) using SVD (2) is same as m0 (2)
Passed Test 3 : Rank(NR) using SVD (2) is same as m0 (2)
Passed Test 4 : Rank(NR) using QR (2) is same as m0 (2)
Passed Test 5 : L0 obtained with QR matches Q21*inv(Q11)
Passed Test 6 : N*K = 0 (Zero matrix)
```

To get the model's stoichiometry matrix we can run the following code:

```
# get the default, unaltered stoichiometric matrix
print ls.getStoichiometryMatrix()
```

```
Out[1]:
[[-1.  1.  0.]
 [ 1.  0. -1.]
 [ 1. -1.  0.]
 [ 0. -1.  1.]]
```

A stoichiometry matrix can be converted into a reordered matrix in which the rows are partitioned into N0 (linearly dependent rows) and Nr (linearly independent rows/reduced stoichiometry matrix). Dependent rows will be located on the top and independent rows will at the bottom.

```
# get a row reordered matrix (into dependent and independent rows)
ls.getReorderedStoichiometryMatrix()
```

The reordered stoichiometry matrix will be the same as the stoichiometry matrix since there are no dependent species (rows) as we can see below.

```
Out[1]:
[[-1.,  1.,  0.],
 [ 1.,  0., -1.],
 [ 1., -1.,  0.],
 [ 0., -1.,  1.]]
```

We can also get species and reaction information from the model.

```
# get the number of dependent reactions (run respective methods for species)
ls.getNumDepReactions()
ls.getNumIndReactions()

# identify dependent reactions (run respective methods for species)
ls.getDependentReactionIds()

# identify independent reactions (run respective methods for species)
ls.getIndependentReactionIds()

# check if a reaction is reversible or not by passing the reaction index.
ls.isReactionReversible(0)
```

There are few methods that compute conserved moeties in a model. We mentioned that there are two interlinked conserved cycles: S1 + S2 + ES and ES + E in the model we generated above.

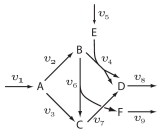
```
# get the conserved matrix (species in columns, and conserved laws in rows)
print(ls.getGammaMatrix())

# get which species are contained in each row of the conserved matrix
print(ls.getGammaMatrixIds())

# get conserved laws associated with them
print(ls.getConservedLaws())

# Get the sums of concentrations as given by the conserved Laws
print(ls.getConservedSums())
```

```
Out[1]:
[[1. 0. 1. 0.]
 [1. 1. 0. 1.]]

(('0', '1'), ('ES', 'S1', 'E', 'S2'))

(' + ES + E', ' + ES + S1 + S2')

(20.0, 30.0)
```

As we can see from the output, there are two conserved laws, ES + E and ES + S1 + S2. As the a model gets very complex, this methods are very useful to analyse conservation laws.

4.3.2 Branched Metabolic Network

A metabolic network with nine reactions and six floating species is shown below. This model was originally developed by Jeremy Zucker.



To generate an SBML string and load the model to LibStructural, run:

```
import structural
import tellurium as te

r = te.loada('''
// Reactions:
J1: $X0 -> A; v;
J2: A -> B ; v;
J3: A -> C; v;
J4: B + E -> 2D; v;
J5: $X1 -> E; v;
J6: B -> C + F; v;
J7: C -> D; v;
J8: D -> ; v;
J9: F -> ; v;
```

(continues on next page)

(continued from previous page)

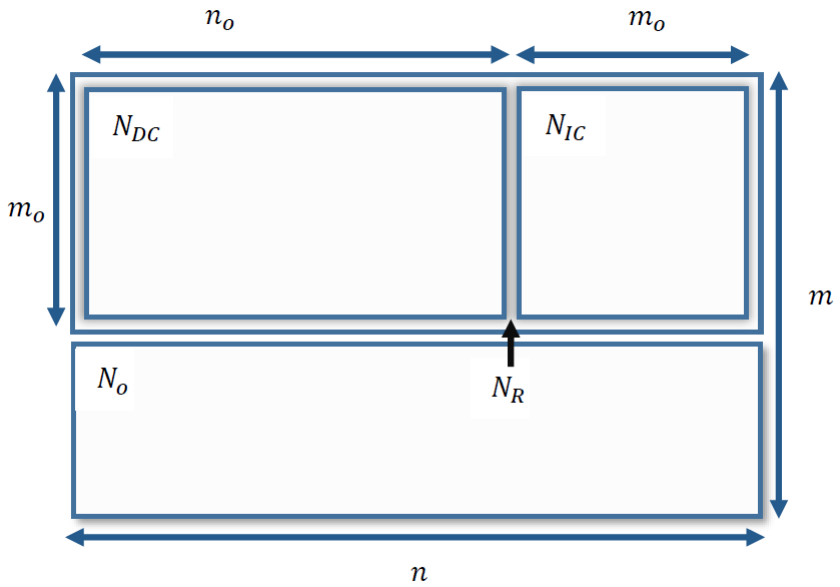
```
// Variable initializations:
    v = 0;

// Species initializations:
A = 10; B = 10; C = 10;  D = 10; E = 10; F = 10;
X0 = 10; X1 = 10; X2 = 10; X3 = 10;
''')

sbmlstr = r.getSBML() # this creates an SBML string from the antimony model, r.
ls = structural.LibStructural()
ls.loadSBMLFromString(sbmlstr)
```

To get the summary result of analyzeWithQR:

```
print(ls.getSummary()) # Prints out if the model is passed some internal structural␣
↪validation tests.
```

```
----------------------------------------------------------------
STRUCTURAL ANALYSIS MODULE : Results
----------------------------------------------------------------
Size of Stochiometric Matrix: 6 x 9 (Rank is  6)
Nonzero entries in Stochiometric Matrix: 16  (29.6296% full)

Independent Species (6) :
D, A, C, F, E, B

Dependent Species : NONE

L0 : There are no dependencies. L0 is an EMPTY matrix

Conserved Entities: NONE
```

A fully reordered stoichiometry matrix is a matrix where the Nr section of the reordered stoichiometry matrix partitioned into NDC (linearly dependent columns) and NIC (linearly independent columns).



```
# get a column and row reordered stoichiometry matrix, run:
print(ls.getFullyReorderedStoichiometryMatrix())
# get the NIC and NDC matrices
print(ls.getNDCMatrix())
print(ls.getNICMatrix()) # NIC matrix is always a square matrix
```

Returns:

```
Out[1]:
[[ 1. -1.  0.]
 [ 0.  1. -1.]
 [-1.  1.  0.]
 [-1.  0.  1.]]

[[1.]
 [0.]]

[[-1.  0.]
 [ 1. -1.]]
```

To compute elementary modes, LibStructural offers three methods; **getElementaryModesInteger**, **getElementaryModesDouble** and **getgElementaryModes**. Elementary modes are the simplest pathways within a metabolic network that can sustain a steady state and at the same time are thermodynamically feasible. **getElementaryModesDouble** and **getgElementaryModes** methods are useful when working with reaction networks containing species with floating (fraction) coefficients, i.e. biomass reactions. Whereas, **getElementaryModesInteger** deals with integer coefficients only. These methods return an array where each row is an elementary mode in the model. For bigger models, the **saveElementaryModes** method computes and writes elementary modes to a file and returns the file path. Note: **getgElementaryModes** and **saveElementaryModes** will only work for models that have less than 448 reactions.

```
print (ls.getElementaryModesInteger())
print (ls.getElementaryModesDouble())
print (ls.getgElementaryModes())

# Compute and write elementary modes to a file:
outPutPath = ls.saveElementaryModes() #File will have a default format.
outPutPath_csv = ls.saveElementaryModes(csv_format=True) #File will have a csv format
```

```
Out[1]:
[[1. 1. 1.]]
[[1. 1. 1.]]

[[1. 1. 1.]]
[[1. 1. 1.]]

[[1. 1. 1.]]
[[1. 1. 1.]]
```

Row numbers indicate the number of elementary modes available and the columns correspond with the reaction number. The order of the reaction index is similar to **getreactionIds** order, except for **getElementaryModesDouble** which sometimes has it's own reaction index order. As such, LibStructural provides **getElementaryModesMetaToolRxnIds** method to obtain the reaction order found in the elementary modes found from **getElementaryModesDouble**.

```
import structural
import pkg_resources

ls = structural.LibStructural()
```

(continues on next page)

(continued from previous page)

```
modelPath = pkg_resources.resource_filename('structural','test/testModel24.xml')

ls.loadSBMLFromFile(modelPath)

# Using Integer version
print (ls.getElementaryModesInteger())
print (ls.getReactionIds())

# Using Double Version
print('\n')
print (ls.getElementaryModesDouble())
print (ls.getElementaryModesMetaToolRxnIds())
```

```
Out[1]:
[[ 1.  1.  0.  0.  0.  0.  0.  0.]
 [ 1.  0.  1.  1.  0.  0.  0.  0.]
 [ 1.  0.  1.  0.  1.  0.  0.  0.]
 [-1.  0. -1.  0.  0.  1.  1.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  1.]
 [ 0.  0.  0.  1.  0.  1.  1.  0.]
 [ 0.  0.  0.  0.  1.  1.  1.  0.]
 [ 0.  1. -1.  0.  0.  1.  1.  0.]
 [ 1.  0.  1.  0.  0.  0. -1.  1.]]
('J1', 'J2', 'J3', 'J4', 'J5', 'J6', 'J7', 'J8')

[[ 1.  0.  0.  0.  1.  0.  0.  0.]
 [ 1.  1.  0.  0.  0.  1.  0.  0.]
 [ 1.  1.  0.  0.  0.  0.  1.  0.]
 [-1. -1.  1.  1.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0.  1.]
 [ 0.  0.  1.  1.  0.  1.  0.  0.]
 [ 0.  0.  1.  1.  0.  0.  1.  0.]
 [ 0. -1.  1.  1.  1.  0.  0.  0.]
 [ 1.  1.  0. -1.  0.  0.  0.  1.]]
['J1', 'J3', 'J6', 'J7', 'J2', 'J4', 'J5', 'J8']
```

In addition, a test script for elementary modes is distributed with LibStructural package that contains 31 different test models. It calculates elementary modes (for both integer and double versions) in each model and test the validity of the elementary modes returned. You can run the script as shown below:

```
import structural
ls = structural.LibStructural()
ls.runElementaryModeTests()
```

4.3.3 References for Elementary Modes Software Tools

Ullah, Ehsan, et al. "gEFM: An Algorithm for Computing Elementary Flux Modes Using Graph Traversal." IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 13, no. 1, 6 May 2015, pp. 122–134., doi:10.1109/tcbb.2015.2430344.

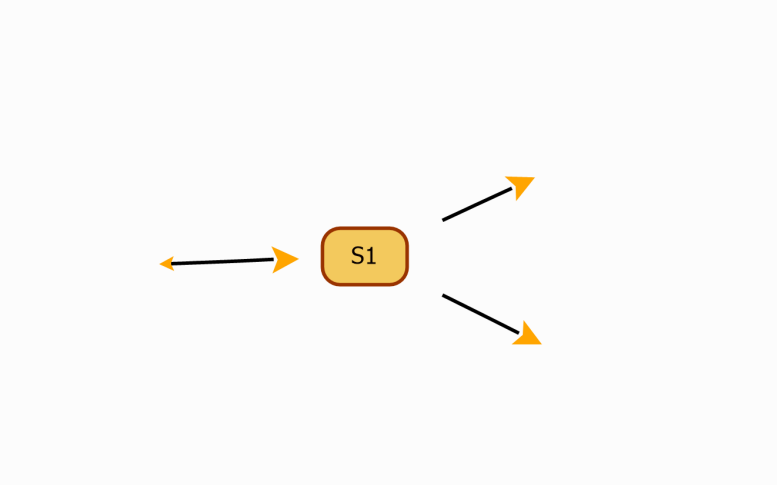
T Pfeiffer, I Sanchez-Valdenebro, J C Nuno, F Montero, S Schuster; METATOOL: for studying metabolic networks., Bioinformatics, Volume 15, Issue 3, 1 March 1999, Pages 251–257, https://doi.org/10.1093/bioinformatics/15.3.251

CHAPTER 5

Example Models

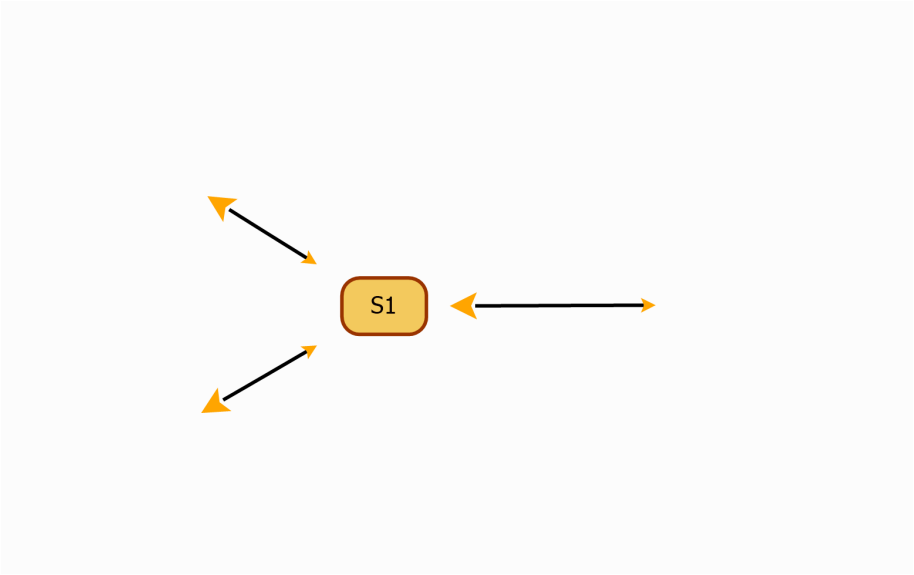
The models found in the package directory, **.../test/** , are shown below with their respective Antimony string representation. The J's indicate reaction id and species with "$" sign in front represent boundary species which are not shown in the figures. The "v" represents reaction rate.

Test Model 1:
J1: $Xo -> S1; v;
J2: S1 => $X1; v;
J3: S1 => $X2; v;
v = 0

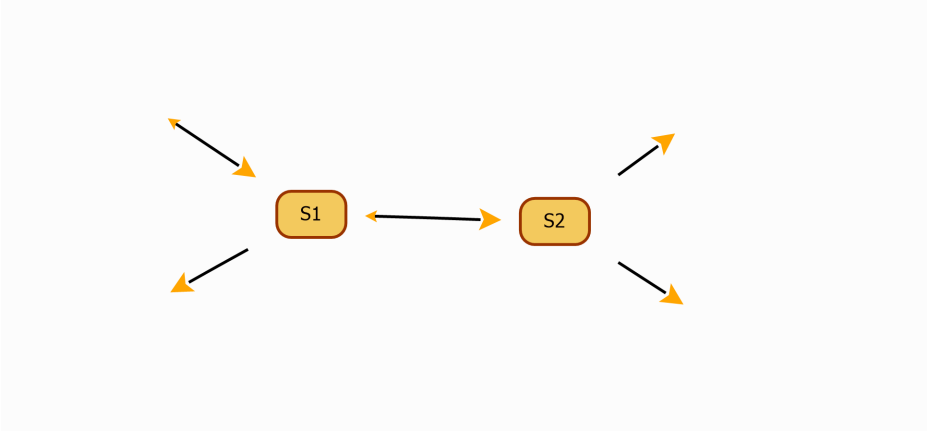


Test Model 2:

J1: $Xo -> S1; v;
J2: S1 -> $X1; v;
J3: S1 -> $X2; v;
v = 0

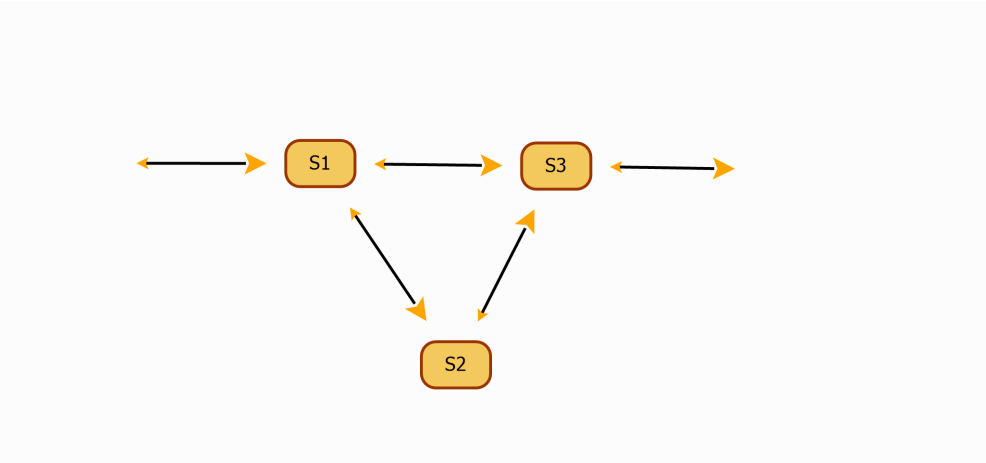


Test Model 3:
J1: $Xo -> S1; v;
J2: S1 -> S2; v;
J3: S2 => $X1; v;
J4: S2 => $X2; v;
J5: S1 => $X3; v;
v = 0

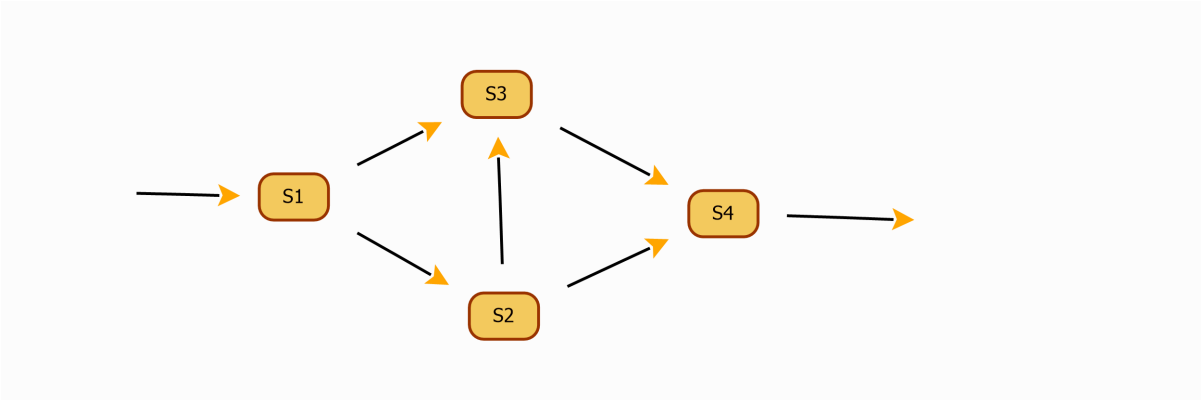


Test Model 4:

J1: $Xo -> S1; v;
J2: S1 -> S2; v;
J3: S2 -> S3; v;
J4: S1 -> S3; v;
J5: S3 -> $X1; v;
v = 0

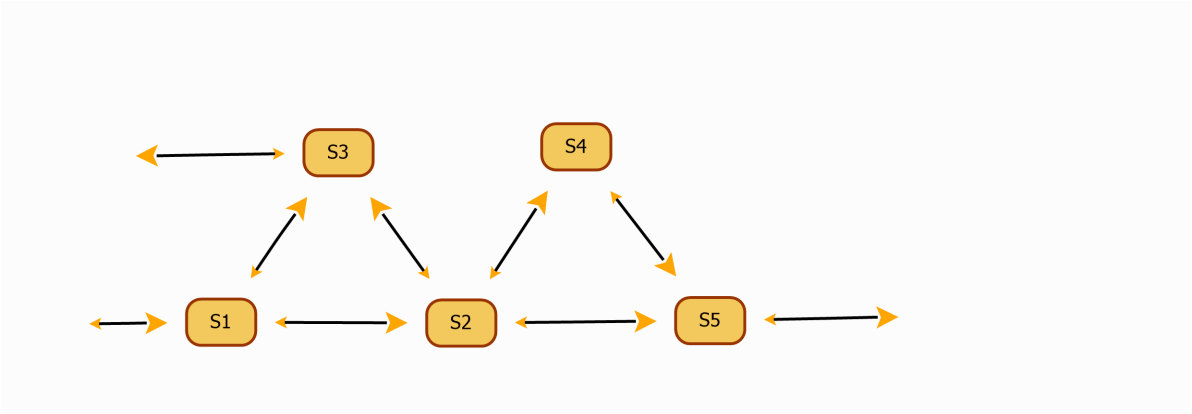


Test Model 5:
J1: $Xo => S1; v;
J2: S1 => S2; v;
J3: S2 => S3; v;
J4: S1 => S3; v;
J5: S2 => S4; v;
J6: S3 => S4; v;
J7: S4 => $X1; v;
v = 0

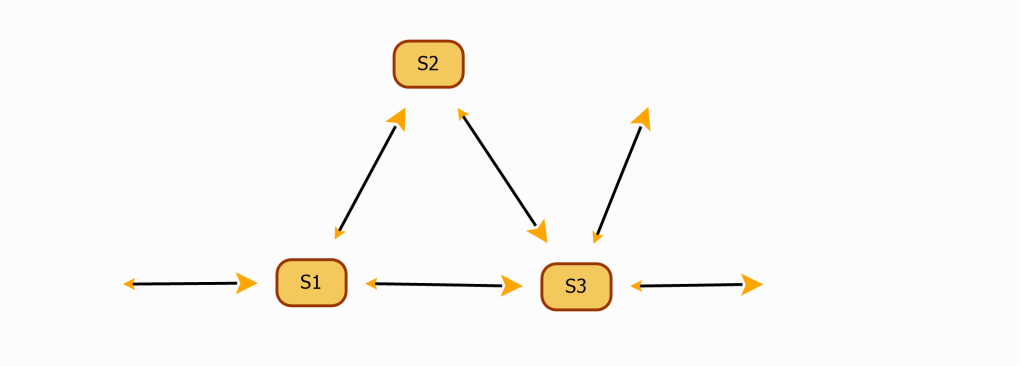


Test Model 6:

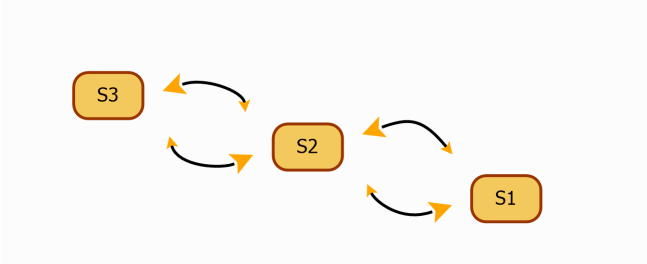
J1: $Xo -> S1; v;
J2: S1 -> S2; v;
J3: S2 -> S3; v;
J4: S1 -> S3; v;
J5: S3 -> $X1; v;
J6: S2 -> S4; v;
J7: S2 -> S5; v;
J8: S4 -> S5; v;
J9: S5 ->; v;
v = 0



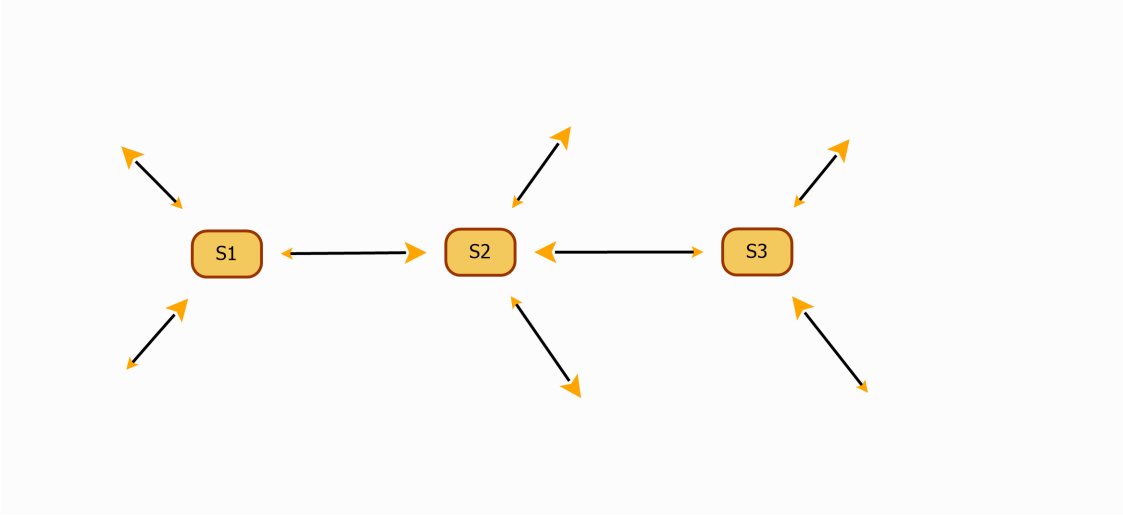
Test Model 7:
J1: $Xo -> S1; v;
J2: S1 -> S2; v;
J3: S2 -> S3; v;
J4: S1 -> S3; v;
J5: S3 -> $X1; v;
J6: S3 -> $X2; v;
v = 0



Test Model 8:
J1: S1 -> S2; v;
J2: S2 -> S1; v;
J3: S2 -> S3; v;
J4: S3 -> S2; v;
v = 0

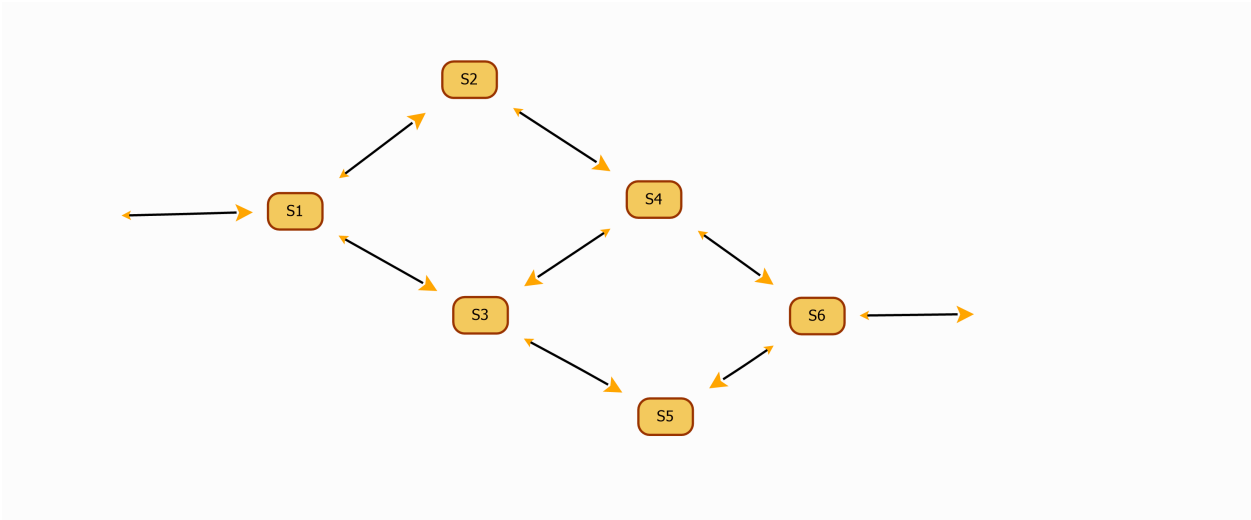


Test Model 9:
J1: $Xo -> S1; v;
J2: S1 -> $X2; v;
J3: S1 -> S2; v;
J4: S2 -> $X3; v;
J5: S2 -> $X4; v;
J6: $X1 -> S3; v;
J7: S3 -> S2; v;
J8: S3 -> $X5; v;
v = 0



Test Model 10:
J1: $Xo -> S1; v;

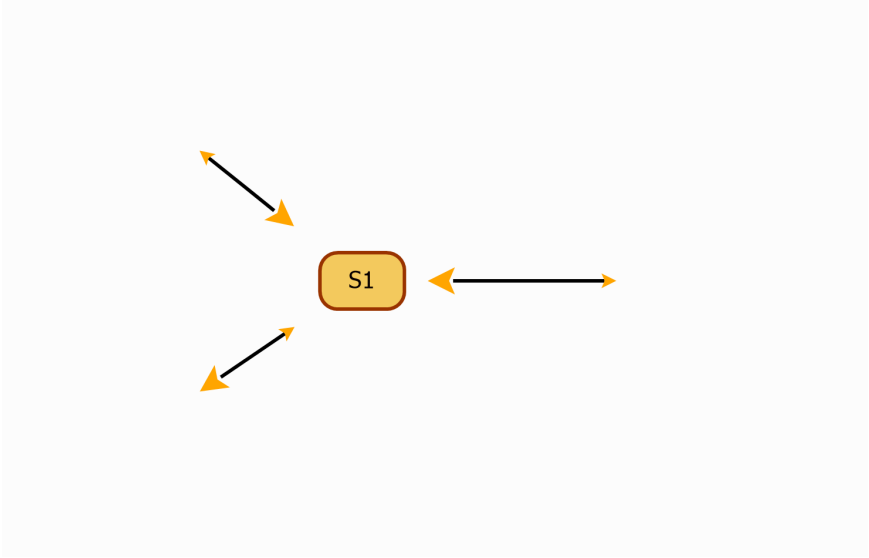
J2: S1 -> S3; v;
J3: S1 -> S2; v;
J4: S2 -> S4; v;
J5: S4 -> S3; v;
J6: S3 -> S5; v;
J7: S4 -> S6; v;
J8: S6 -> $X1; v;
J9: S6 -> S5; v;
v = 0

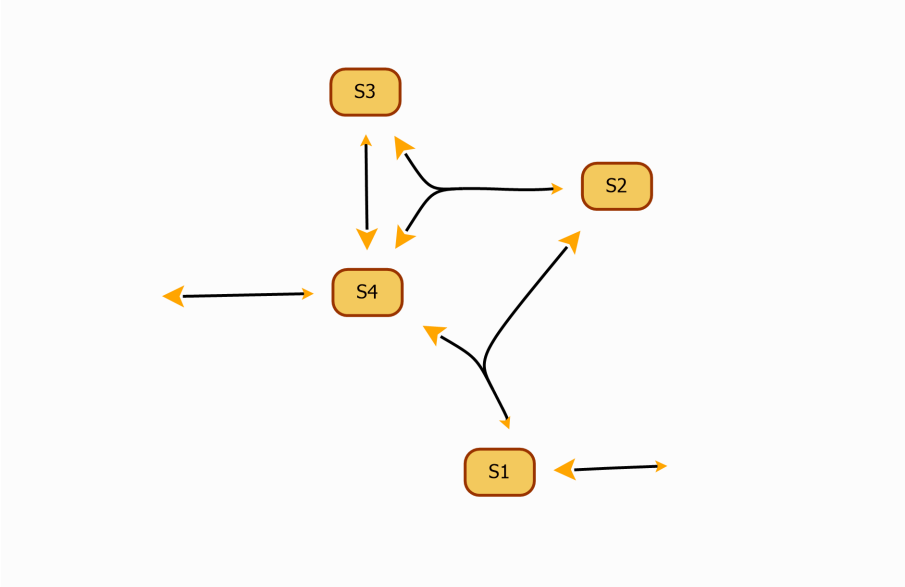


Test Model 11:
J1: $Xo -> S1; v;
J2: $X1 -> S1; v;
J3: S1 -> $X2; v;
v = 0

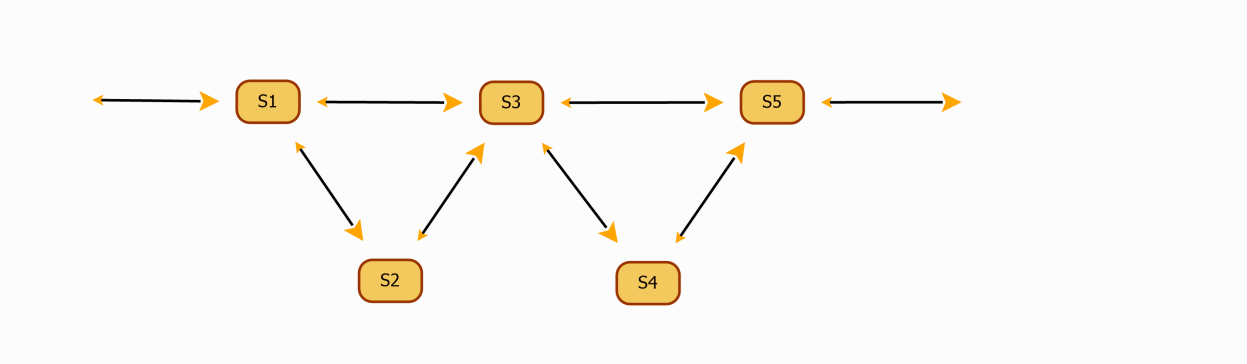
Test Model 12:
J1: $Xo -> S1; v;
J2: S1 -> S2 + S4; v;
J3: S2 -> S3 + S4 ; v;
J4: S3 -> S4; v;
J5: S4 -> $X1; v;
v = 0

Test Model 13:
J1: $Xo -> S1; v;

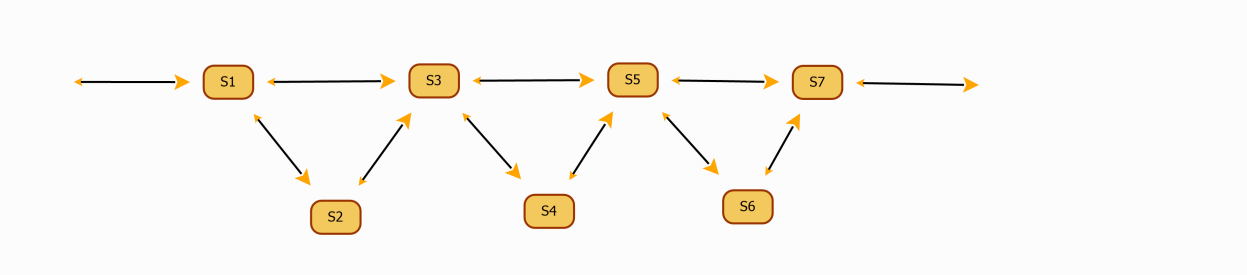

S1


S3
S2
S4
S1

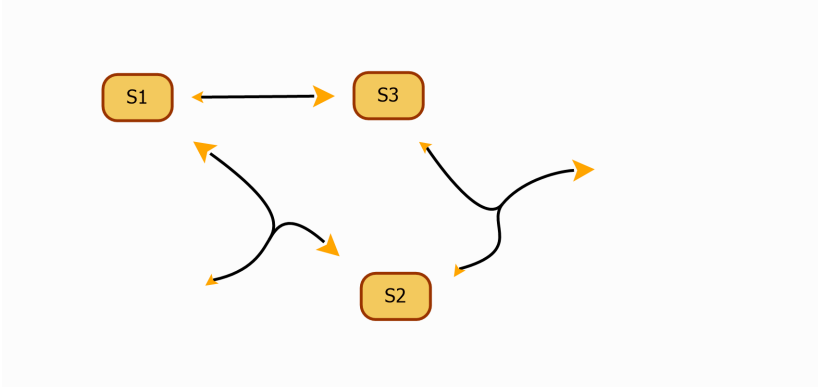
J2: S1 -> S2; v;
J3: S2 -> S3; v;
J4: S1 -> S3; v;
J5: S3 -> S4; v;
J6: S3 -> S5; v;
J7: S4 -> S5; v;
J8: S5 -> $X1; v;
v = 0



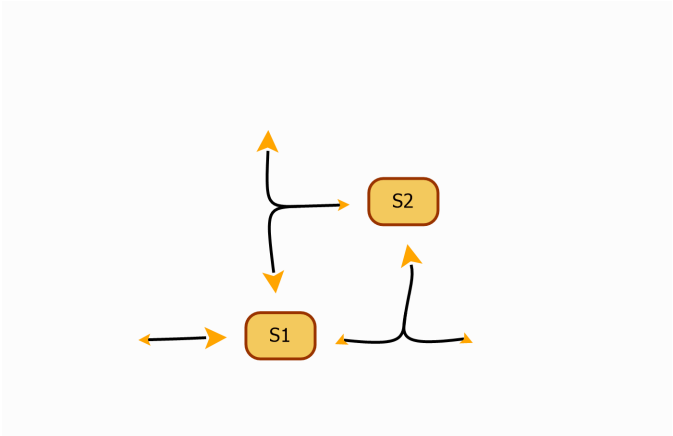
Test Model 14:
J1: $Xo -> S1; v;
J2: S1 -> S2; v;
J3: S2 -> S3; v;
J4: S1 -> S3; v;
J5: S3 -> S4; v;
J6: S3 -> S5; v;
J7: S4 -> S5; v;
J8: S5 -> S6; v;
J9: S5 -> S7; v;
J10: S6 -> S7; v;
J11: S7 -> $X1; v;
v = 0



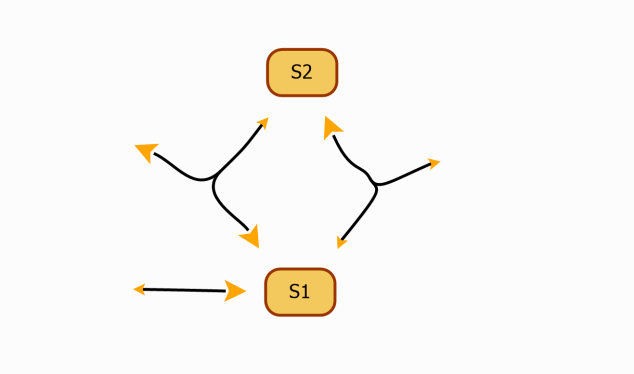
Test Model 15:
J1: $Xo -> S1 + S2; v;
J2: S1 -> S3; v;
J3: S2 + S3 -> $X1; v;
v = 0



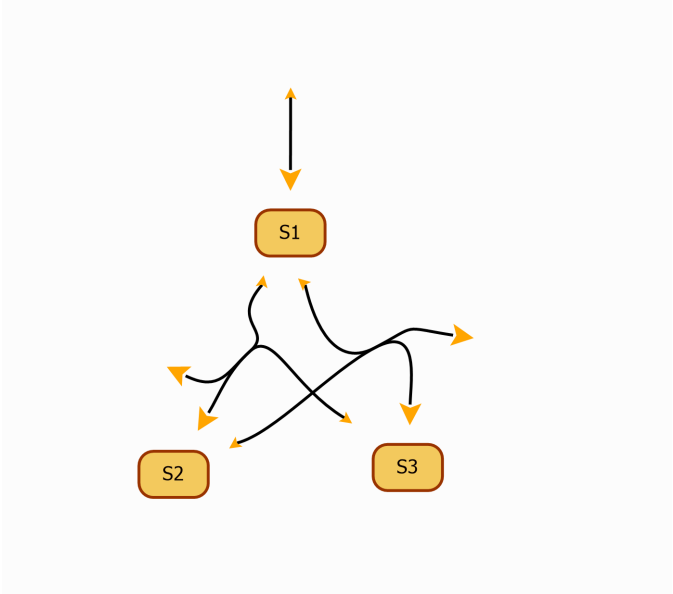
Test Model 16:
J1: $Xo -> S1; v;
J2: S1 + $X1 -> S2; v;
J3: S2 -> S1 + $X2; v;
v = 0



Test Model 17:
J1: $Xo => S1; v;
J2: S1 + $X1 => S2; v;
J3: S2 => S1 + $X2; v;
v = 0

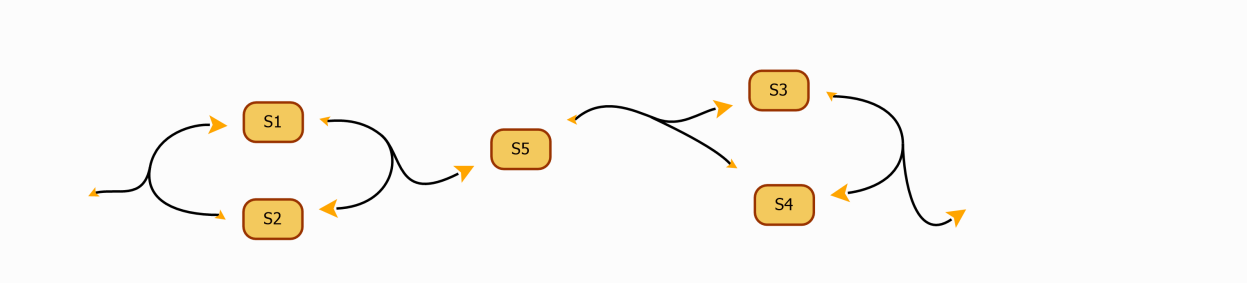


Test Model 18:
J1: $Xo -> S1; v;
J2: S1 + S2 -> S3 + $X1; v;
J3: S1 + S3 -> S2 + $X2; v;
v = 0

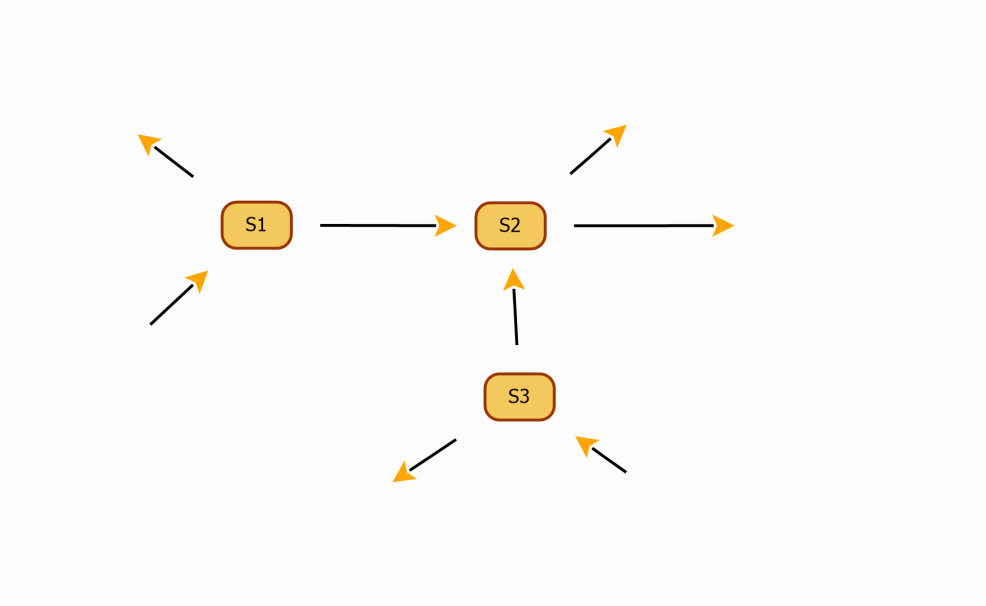


Test Model 19:
J1: $Xo + S2 -> S1; v;
J2: S1 -> S2 + S5; v;
J3: S5 + S4 -> S3; v;
J4: S3 -> S4 + $X1; v;
v = 0

Test Model 20:

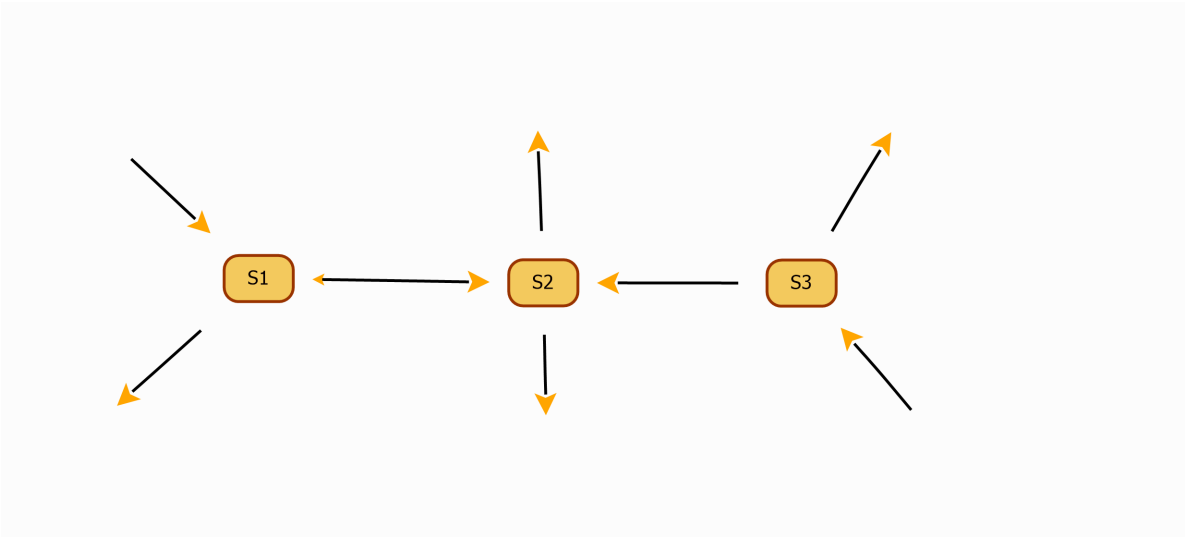


J1: $Xo => S1; v;
J2: S1 => $X2; v;
J3: S1 => S2; v;
J4: S2 => $X3; v;
J5: S2 => $X4; v;
J6: $X1 => S3; v;
J7: S3 => S2; v;
J8: S3 => $X5; v;
v = 0

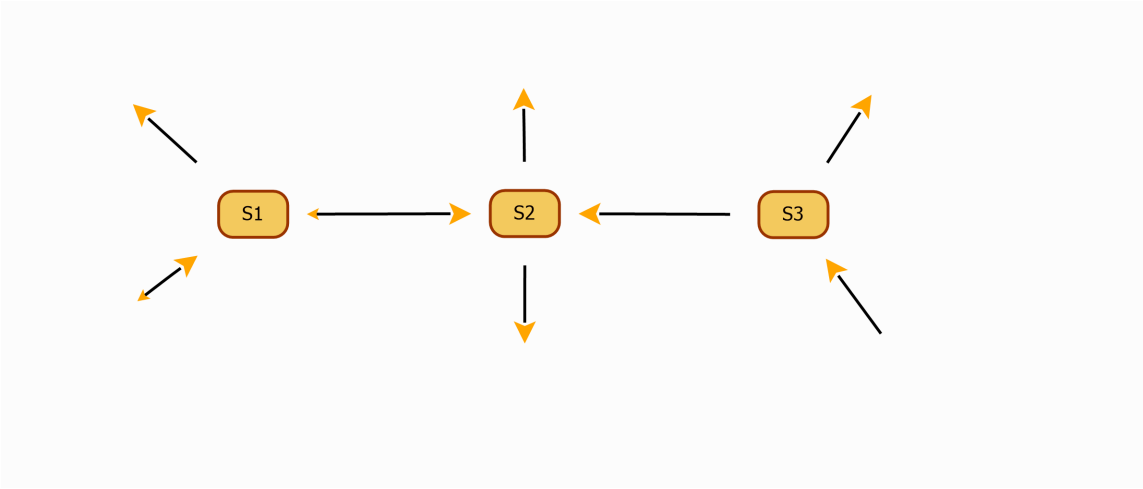


Test Model 21:
J1: $Xo => S1; v;
J2: S1 => $X2; v;
J3: S1 -> S2; v;
J4: S2 => $X3; v;
J5: S2 => $X4; v;
J6: $X1 => S3; v;
J7: S3 => S2; v;

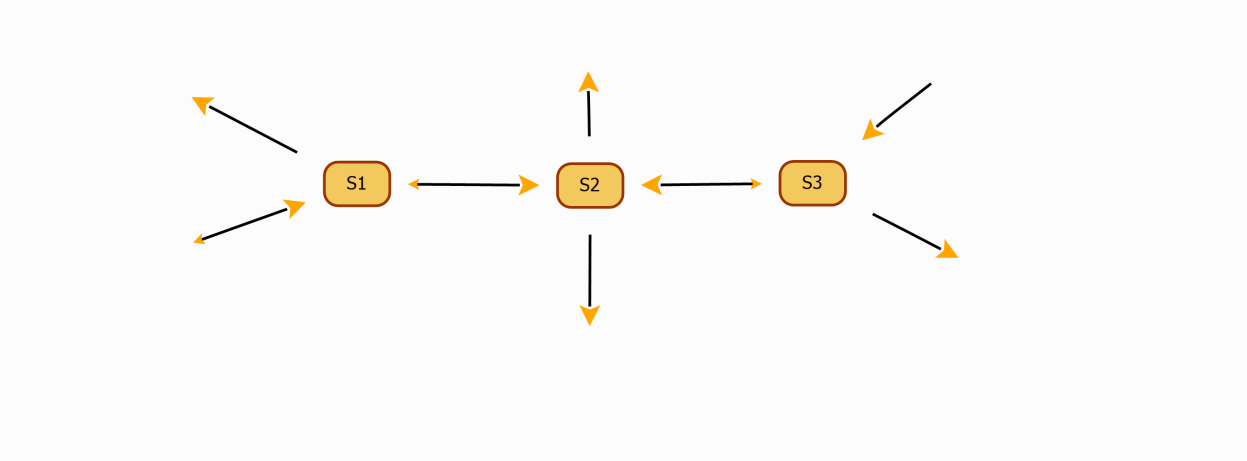
J8: S3 => $X5; v;
v = 0



Test Model 22:
J1: $Xo -> S1; v;
J2: S1 => $X2; v;
J3: S1 -> S2; v;
J4: S2 => $X3; v;
J5: S2 => $X4; v;
J6: $X1 => S3; v;
J7: S3 => S2; v;
J8: S3 => $X5; v;
v = 0



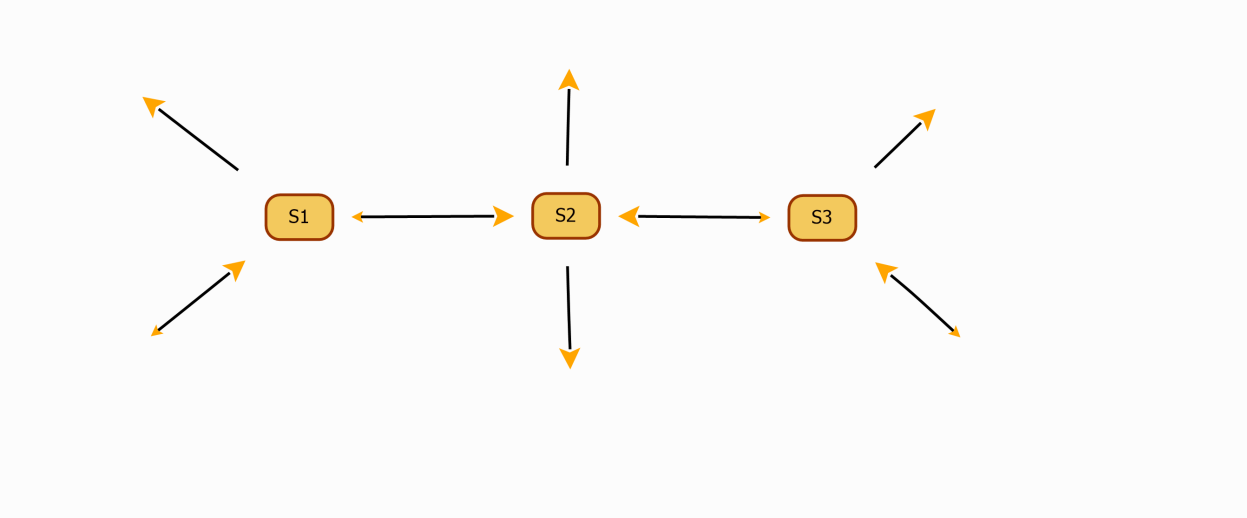
Test Model 23:
J1: $Xo -> S1; v;
J2: S1 => $X2; v;
J3: S1 -> S2; v;
J4: S2 => $X3; v;
J5: S2 => $X4; v;
J6: $X1 => S3; v;
J7: S3 -> S2; v;
J8: S3 => $X5; v;
v = 0

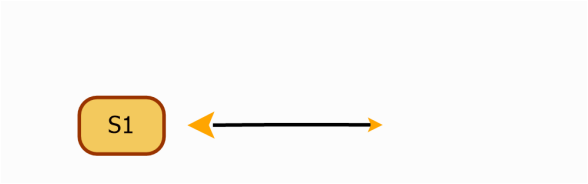


Test Model 24:
J1: $Xo -> S1; v;
J2: S1 => $X2; v;
J3: S1 -> S2; v;
J4: S2 => $X3; v;
J5: S2 => $X4; v;
J6: $X1 -> S3; v;
J7: S3 -> S2; v;
J8: S3 => $X5; v;
v = 0

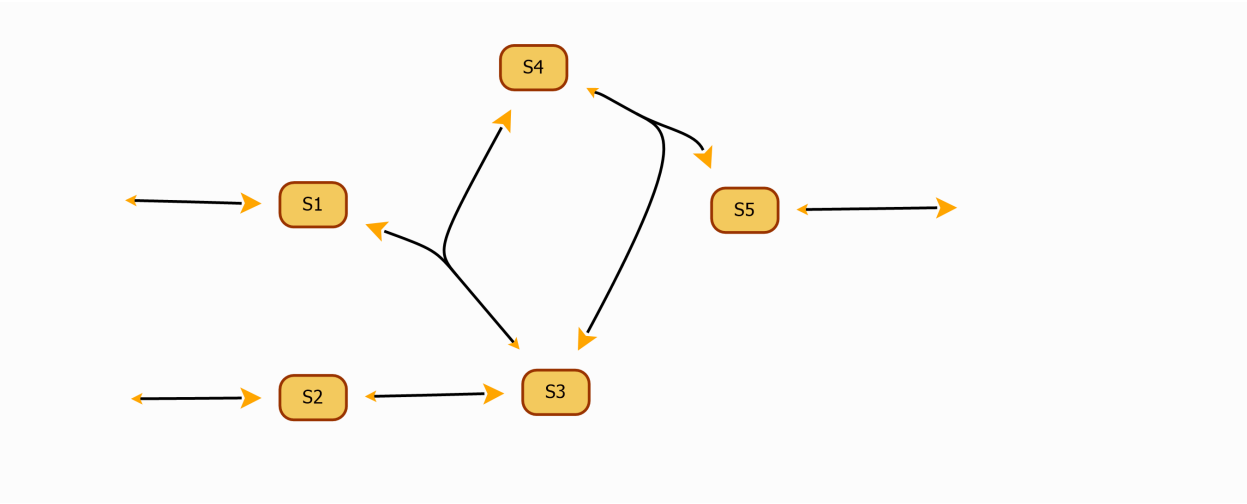
Test Model 25:
$X -> S1; v;
v = 0

Test Model 26:



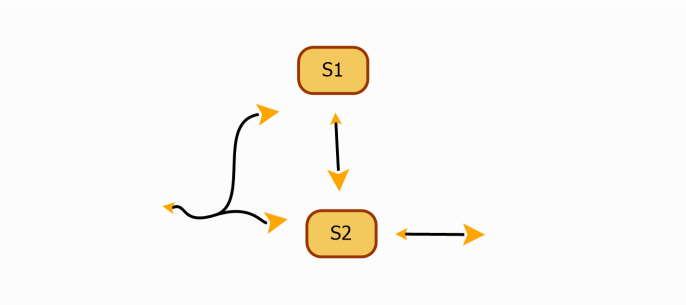


J1: $Xo -> S1; v;
J2: $X1 -> S2; v;
J3: S2 -> S3; v;
J4: S3 -> S1 + S4; v;
J5: S4 -> S3 + S5; v;
J6: S5 -> $X2; v;
v = 0

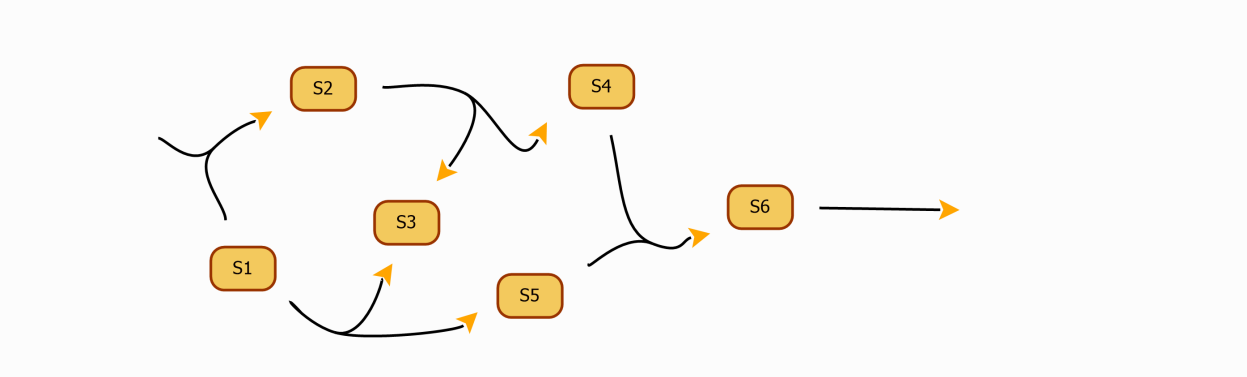


Test Model 27:

J1: $Xo -> S1 + S2; v;
J2: S1 -> S2; v;
J3: S2 -> $X1; v;
v = 0

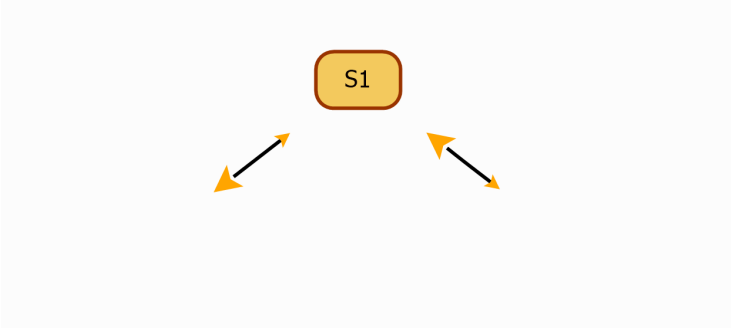


Test Model 28:
J1: $Xo + S1 => S2; v;
J2: S2 => S3 + S4; v;
J3: S1 => S5 + S3; v;
J4: S4 + S5 => S6; v;
J5: S6 => $X1; v;
v = 0

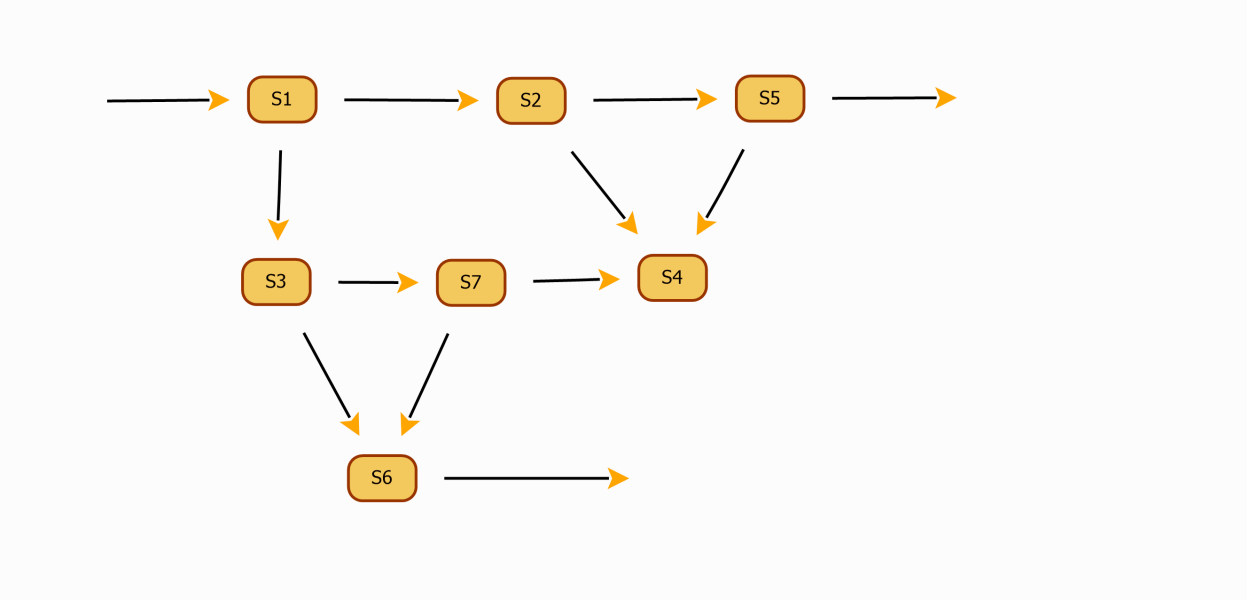


Test Model 29:
J1: $Xo -> S1; v
J2: S1 -> $X1; v;
v = 0

Test Model 30:
J1: $Xo => S1; v

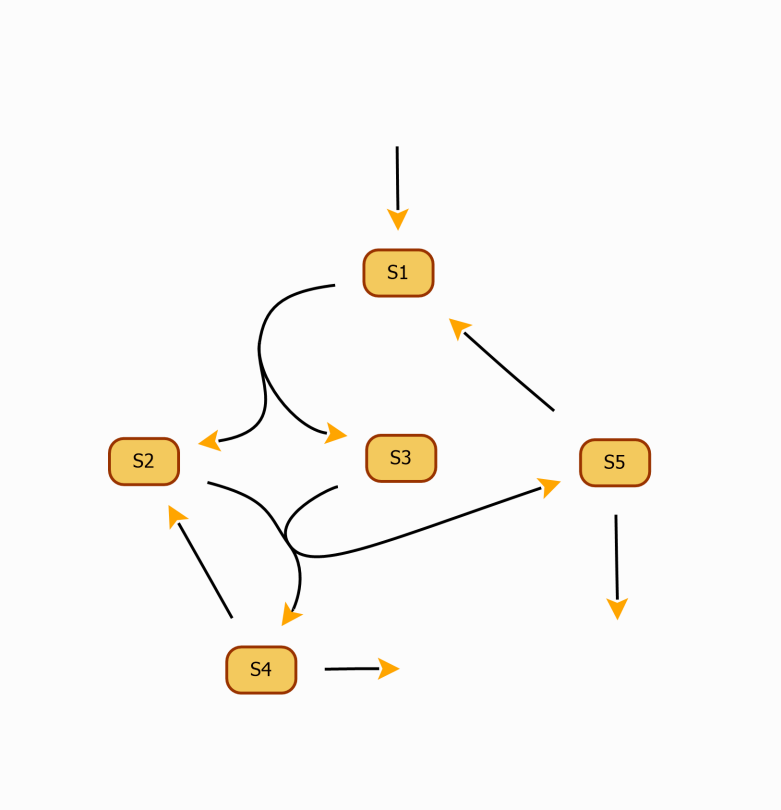


J2: S1 => S3; v;
J3: S1 => S2; v;
J4: S2 => S5; v;
J5: S2 => S4; v;
J6: S3 => S6; v;
J7: S3 => S7; v;
J8: S7 => S6; v;
J9: S7 => S4;v;
J10: S5 => S4; v;
J11: S6 => $X1; v;
J12: S5 => $X2; v;
v = 0



Test Model 31:
J1: $Xo => S1; v;
J2: S1 => S2 + S3; v;
J3: S2 + S3 => S4 + S5; v;

J4: S5 => $X1; v;
J5: S4 => S2; v;
J6: S5 => S1; v;
v = 0



CHAPTER 6

Uploading Wheels (pip)

Before generating and uploading wheels, read this information carefully:

Versioning in PyPI is very strict. You need to follow a specific guideline when bumping the version number.

There is absolutely NO do-overs. Once you upload a version, that namespace is occupied forever. Therefore, we will follow post-release version scheme for minor bug fixes. When you wish to upload a new version without changes in version number, add .post<post-release-version-number> at the end. It is not at all unusual to upload several posts for a given release in this manner. Just increment the post number in the setup.py script.

LibStructural supports Microsoft Windows and Apple Mac OS X. Check each section for detailed instructions.

6.1 Windows

To generate wheels and upload it to PyPI, you need Python, twine, and PyPI account with the right authentication. Make sure that the version number in setup.py is correct.

To build a Python wheel run:

```
python setup.py bdist_wheel --python-tag=cp27 --plat-name=win_amd64
```

in command console. Make sure you are in a correct python version corresponding with the python tag. The above line generates a Python wheel for 64-bit Python 2.7. Change flags if you want to generate wheels for different Python versions (cp35, cp36, etc.) or different architecture (win32). Once finished, a wheel will be generated under dist folder within sub-folder (Win_64/2, etc.) with name such as:

```
structural-2.3.0.post1-cp27-none-win_amd64.whl
```

Make sure to rename ABI tag 'none' to 'cp27m' (or 'cp35m', 'cp36m', etc.), e.g.:

```
structural-2.3.0.post1-cp27-cp27m-win_amd64.whl
```

To upload the wheels to PyPI, run:

```
twine upload <file-location> -u <username> -p <password>
```

It is also possible to configure a .pypirc file so that you don't have to type in your username and password every time when you upload. See this page for more info.

6.2 Mac OS X

The process for building PyPI distributions on Mac parallels the process for Windows. The following command lines are useful:

```
python setup.py bdist_wheel --python-tag=cp27 --plat-name=macosx_10_11_x86_64
```

```
python3 setup.py bdist_wheel --python-tag=cp36 --plat-name=macosx_10_11_x86_64
```

The wheel will be put in the dist directory. If one of the fields in the wheel directory is "none", change to cp27m (replace 27 with whatever Python version you used).

To upload the package, use twine :

```
twine upload structural-2.1.0-cp27-cp27m-macosx_10_11_x86_64.whl
```

CHAPTER 7

API

class LibStructural

analyzeWithFullyPivotedLU(*self*)

Uses fully pivoted LU Decomposition for Conservation analysis.This method performs the actual analysis of the stoichiometry matrix loaded either via

`LibStructural.loadStoichiometryMatrix` or `LibStructural.loadSBMLFromString`.

Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

`LibStructural.analyzeWithQR`
`LibStructural.analyzeWithLU`
`LibStructural.analyzeWithLUandRunTests`
`LibStructural.analyzeWithFullyPivotedLU` or
`LibStructural.analyzeWithFullyPivotedLUwithTests`

Returns A string with information about the analysis process

analyzeWithFullyPivotedLUwithTests(*self*)

Uses fully pivoted LU Decomposition for Conservation analysis. This method performs the actual analysis of the stoichiometry matrix loaded either via

`LibStructural.loadStoichiometryMatrix` or `LibStructural.loadSBMLFromString`.

Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

`LibStructural.analyzeWithQR`,

LibStructural.analyzeWithLU,
‘‘LibStructural.analyzeWithLU‘‘andRunTests,
LibStructural.analyzeWithFullyPivotedLU or
LibStructural.analyzeWithFullyPivotedLUwithTests

This method additionally performs the integrated test suite and returns those results.

Returns A string with information about the analysis process

analyzeWithLU(*self*)
Uses LU Decomposition for Conservation analysis. This method performs the actual analysis of the stoichiometry matrix (loaded either via

LibStructural.loadStoichiometryMatrix or LibStructural.loadSBMLFromString.

Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

LibStructural.analyzeWithQR,
LibStructural.analyzeWithLU,
LibStructural.analyzeWithLUandRunTests,
LibStructural.analyzeWithFullyPivotedLU or
LibStructural.analyzeWithFullyPivotedLUwithTests

Returns A string with information about the analysis process

analyzeWithLUandRunTests(*self*)
Uses LU Decomposition for Conservation analysis. This method performs the actual analysis of the stoichiometry matrix loaded either via

LibStructural.loadStoichiometryMatrix or LibStructural.loadSBMLFromString.

Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

LibStructural.analyzeWithQR,
LibStructural.analyzeWithLU,
LibStructural.analyzeWithLUandRunTests,
LibStructural.analyzeWithFullyPivotedLU or
‘‘LibStructural.analyzeWithFullyPivotedLUwithTests’‘

This method additionally performs the integrated test suite and returns those results.

Returns A string with information about the analysis process

analyzeWithQR(*self*)
Uses QR factorization for structural analysis. This method performs the actual analysis of the stoichiometry matrix loaded either via

LibStructural.loadStoichiometryMatrix or LibStructural.loadSBMLFromString.

Only after one of the analysis methods below has been called are the structural matrices (L0, K0...) available.

LibStructural.analyzeWithQR,
LibStructural.analyzeWithLU,
LibStructural.analyzeWithLU, RunTests,
LibStructural.analyzeWithFullyPivotedLU,
LibStructural.analyzeWithFullyPivotedLUwithTests

Remarks: This is the prefered method for structural analysis.

Returns A string with information about the analysis process

getColumnReorderedNrMatrix (*self*)

Returns The Nr Matrix repartitioned into NIC (independent columns) and NDC (dependent columns). The Nr matrix contains the independent rows of the stoichiometry matrix

getColumnReorderedNrMatrixIds (*self*)

Returns The Nr Matrix row and column labels (repartitioned into NIC and NDC).

getConditionNumber (*self*, *matrix*)

Param Takes a matrix (numpy) as an argument. Find the condition number of the matrix.

Returns The condition number

getConservedLaws (*self*)

Returns Algebraic expressions for the conserved cycles. Returns empty if none.

getConservedSums (*self*)

Returns Total mass for each conserved cycle in the model. The sum is based on the current initial conditions.

getDependentReactionIds (*self*)

Returns The list of Ids for the dependent reactions.

getDependentSpeciesIds (*self*)

Returns The list of Ids for the dependent species.

getEigenValues (*self*, *matrix*)

Param Matrix to find the eigenvalues for.

Returns An array, first column are the real values and second column are imaginary values

getEigenVectors (*self*, *matrix*)

Param Matrix to find the eigenvectors for

Returns An array where each columns is an eigenvector

getElementaryModesDouble (*self*)

Returns An array where each column is an elementary mode (Generated from MetaTool)

getElementaryModesInteger (*self*)

Returns An array where each column is an elementary mode

getElementaryModesMetaToolRxnIds (*self*)

Returns An array of reaction Ids corresponding with the columns of getElementaryModesMeta-ToolRxnIds() matrix.

getFloatingSpeciesIds (*self*)

Returns The unordered list of floating species Ids.

getFullyReorderedN0StoichiometryMatrix (*self*)
Computes the N0 matrix if possible. The N0 matrix will contain all the dependent rows of the stoichiometry matrix.

Returns The N0 Matrix.

getFullyReorderedNrMatrix (*self*)
The Nr matrix contains all the linearly independent rows of the stoichiometry matrix.

Returns The Nr Matrix.

getFullyReorderedStoichiometryMatrix (*self*)

Returns The fully reordered stoichiometry matrix. Rows and columns are reordered so all indepedent rows

of the stoichiometry matrix are brought to the top and left side of the matrix.

getFullyReorderedStoichiometryMatrixIds (*self*)

Returns The row and column labels for the reordered stoichiometry matrix (row reordered stoichiometry matrix)

getGammaMatrix (*self*)

Returns Gamma, the conservation law array.

Each row represents a single conservation law where the column indicates the participating molecular species. The number of rows is therefore equal to the number of conservation laws. Columns are ordered according to the rows in the reordered stoichiometry matrix, see `LibStructural.getReorderedSpeciesId` and `LibStructural.getReorderedStoichiometryMatrix`.

getGammaMatrixGJ (*self*, *matrix*)

Param The stoichiometry matrix that will be used to calculate gamma matrix.

Returns Gamma, the conservation law array.

Each row represents a single conservation law where the column indicate the participating molecular species. The number of rows is therefore equal to the number of conservation laws. In this case the Gamma Matrix is recalculated for the given stoichiometry matrix. The column label will be the same label as the stoichiometry matrix.

getGammaMatrixIds (*self*)

Returns The row and column labels for Gamma, the conservation law array.

getIndependentReactionIds (*self*)

Returns The list of Ids for the independent reactions.

getIndependentSpeciesIds (*self*)

Returns The list of Ids for the independent species.

getInitialConditions (*self*)

Returns Initial Conditions used in the model.

static getInstance (*self*)

static method to get an instance of LibStructural (allows use as singleton)

import structural ls = structural.LibStructural.getInstance()

getK0Matrix (*self*)

Returns The K0 Matrix.

K0 is defined such that K0 = -(NIC)$^{-1}$* NDC, or equivalently, [NDC NIC][I K0]' = 0 where [NDC NIC] = Nr

getK0MatrixIds (*self*)

Returns The K0 Matrix row and column labels.

getKMatrix (*self*)

Returns The K matrix (right nullspace of Nr)

The K matrix has the structure, [I K0]'

getKMatrixIds (*self*)

Returns the K matrix row and column labels.

getL0Matrix (*self*)

Returns The L0 Matrix.

L0 is defined such that L0*Nr = N0. L0 forms part of the link matrix, L. N0 is the set of linear dependent rows from the lower portion of the reordered stoichiometry matrix.

getL0MatrixIds (*self*)

Returns The L0 Matrix row and column labels.

getLeftNullSpace (*self*, *matrix*)

Param Matrix to find the left nullspace of.

Returns The Left Nullspace of the matrix argument.

getLinkMatrix (*self*)

Returns L, the Link Matrix, left nullspace (aka nullspace of the transpose Nr).

L will have the structure, [I L0]', such that L*Nr = N

getLinkMatrixIds (*self*)

Returns The row and column labels for the Link Matrix, L

getModelName (*self*)

Returns The name of the model if SBML model has Name-tag, otherwise it returns the SBML id. If only a stoichiometry matrix was loaded 'untitled' will be returned.

getN0Matrix (*self*)

Returns The N0 Matrix.

The N0 matrix is the set of linearly dependent rows of N where L0 Nr = N0.

getN0MatrixIds (*self*)

Returns The N0 Matrix row and column labels.

getNDCMatrix (*self*)

Returns The NDC Matrix (the set of linearly dependent columns of Nr).

getNDCMatrixIds (*self*)

Returns The NDC Matrix row and column labels.

getNICMatrix (*self*)

Returns The NIC Matrix (the set of linearly independent columns of Nr)

getNICMatrixIds (*self*)

Returns The NIC Matrix row and column labels.

getNmatrixSparsity (*self*)

Returns The number of nonzero values in Stoichiometry matrix

getNrMatrix (*self*)

Returns The Nr Matrix.

The rows of the Nr matrix will be linearly independent.

getNrMatrixIds (*self*)

Returns The Nr Matrix row and column labels.

getNumDepReactions (*self*)

Returns The number of dependent reactions

getNumDepSpecies (*self*)

Returns The number of dependent species.

getNumIndReactions (*self*)

Returns The number of independent reactions

getNumIndSpecies (*self*)

Returns The number of independent species.

getNumReactions (*self*)

Returns The total number of reactions.

getRConditionNumber (*self*, *matrix*)

Find the condition number of a matrix.

Param A matrix as an argument.

Returns The condition number

getRank (*self*, *matrix*)

Param Matrix to find the rank of.

Returns The rank as an integer.

getReactionIds (*self*)

Returns The list of reaction ids

getReorderedReactionIds (*self*)

Returns The reordered Id list of reactions.

getReorderedSpeciesIds (*self*)

Returns The reordered list of molecular species (choosing the SBML Id if possible)

getReorderedStoichiometryMatrix (*self*)

Returns The reordered stoichiometry matrix (row reordered stoichiometry matrix, columns are not reordered!)

getReorderedStoichiometryMatrixIds (*self*)

Returns The row and column labels for the reordered stoichiometry matrix (row reordered stoichiometry matrix)

getRightNullSpace (*self*, *matrix*)

Param Matrix to find the right nullspace of.

Returns The Right Nullspace of the matric argument.

getStoichiometryMatrix (*self*)

Returns Unaltered stoichiometry matrix.

getStoichiometryMatrixBoundary (*self*)

Returns Unaltered stoichiometry matrix with boundary species included at the end.

getStoichiometryMatrixIds (*self*)

Returns The row and column labels for the original and unaltered stoichiometry matrix.

getSummary (*self*, **args*)

Returns Returns the summary string of the last analysis.

getTestDetails (*self*)

Returns Details about the validation tests.

getTolerance (*self*)

Returns The currently used tolerance for deciding whether a small number is considered zero or not.

This function returns the tolerance currently used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered zero and will be neglected.

getgElementaryModes (*self*)

Returns An array where each column is an elementary mode

isReactionReversible (*self*, **args*)

Param Reaction index

Returns True if the reaction with given index is reversible

Checks whether a given reaction is reversible or not.

loadReactionIdsWithValues (*self*, *id_array*, *value_array*)

Param A vector of reaction ids to load

Param A vector of reaction rates

This function should be used whenever labeled matrices are important as these labels will be used in labeling the structural matrices. This function sets the reaction ids. It is also possible to provide reaction rate values. This will be used when calculating the conserved sums.

Remarks: This method should only be called after `LibStructural.loadStoichiometryMatrix`. `LibStructural.analyzeWithQR` should be called for the function to take effect.

loadSBMLFromFile (*self*, *SBMLfile*)

Param An SBML file name to load

Returns Information about the loaded model

Loads SBML model from the specified file.

loadSBMLFromString (*self*, *SBMLstring*)

Param SBML string to load

Returns Information about the loaded model

Loads SBML model from a string variable.

loadSBMLwithTests (*self*, **args*)

Param A file name to an SBML model to load

Returns Information about the loaded model and results of the internal test suite

Loads an SBML model into the library and carries out tests using the internal test suite.

loadSpeciesIdsWithValues (*self*, *id_array*, *value_array*)

Param A vector of species ids to load

Param A vector of initial concentrations

Loads species names and initial values. This function should be used whenever labeled matrices are important as these labels will be used in labeling the structural matrices. This function sets the species ids. It is also possible to provide an initial condition for each of the species. This will be used when calculating the conserved sums.

Remarks: This method should only be called after `LibStructural.loadStoichiometryMatrix`. `LibStructural.analyzeWithQR` should be called for the function to take effect.

loadStoichiometryMatrix (*self*, *Matrix*)

Param 2D array stoichiometry matrix

Loads a stoichiometry matrix into the library. To analyze the stoichiometry call one of the following:

```
LibStructural.analyzeWithQR,
LibStructural.analyzeWithLU,
LibStructural.analyzeWithLUandRunTests,
LibStructural.analyzeWithFullyPivotedLU,
LibStructural.analyzeWithFullyPivotedLUwithTests,
```

rref (*self*, *matrix*)

Computes the reduced row echelon of the given matrix. Tolerance is set to indicate the smallest number consider to be zero.

Param A matrix (numpy)

Param Optional: tolerance (float), default is 1E-6

Returns A reduced row echelon form of the matrix

rref_FB (*self*, *matrix*)

Param A matrix

Returns The reduced row echelon form of the matrix.

runElementaryModeTests (*self*)

Returns A valididty test result of elementary modes generated from 31 models.

runLibstructTests (*self*)

Returns A summary of test results on the integrity of structural methods.

saveElementaryModes (*self*, *csv_format=False*)

Returns A directory path for the file generated

setTolerance (*self*, **args*)

Param An integer or a float

Sets user specified tolerance. This function sets the tolerance used by the library to determine what value is considered as zero. Any value with absolute value smaller than this tolerance is considered as zero and will be neglected.

validateStructuralMatrices (*self*)

Validates structural matrices. Calling this method will run the internal test suite against the structural matrices. Those tests include:

Test 1 : Gamma*N = 0 (Zero matrix)
Test 2 : Rank(N) using SVD (5) is same as m0 (5)
Test 3 : Rank(NR) using SVD (5) is same as m0 (5)
Test 4 : Rank(NR) using QR (5) is same as m0 (5)
Test 5 : L0 obtained with QR matches Q21*inv(Q11)
Test 6 : N*K = 0 (Zero matrix)

Returns A tuple of strings indicating whether a particular test passed or failed.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

I

L

R

S

V