LNA++

[publication details…]

Contents

Introduction

LNA++ is a tool designed to efficiently compute the linear noise approximation to a chemical reaction network, along with the first two sensitivities of the mean and temporal cross-covariance matrix, using a fast C++ implementation. LNA++ can be either be integrated into a C++ project, or used in conjunction with Matlab or Python for parameter inference in chemical reaction networks.

This document describes the library requirements, installation and usage of LNA++.

Installation

The contents of the LNA++ distribution zip file should be placed into a separate folder, such as in a folder within the home directory. The archive contains all the necessary scripts and source code for building LNA++ Python modules or Matlab executable files. No further building or installation is necessary until the creation of the model executables, as described below.

Requirements

LNA++ uses the standard C++ libraries, the CVODES Sundials library for numerical integration of ODEs and forward sensitivities, and the Blitz++ library for fast, templatized tensor operations.

Compiler

Compilation of C++ source code for use as a standalone executable, or with Matlab or Python requires the use of a suitable compiler such as gcc/g++ on Linux-based systems, or clang++ on mac OS.

Blitz++

Blitz++ version 0.9 or greater is required. It can be obtained via the included source code in the 'libraries' folder, or from http://sourceforge.net. It is also available via macports on mac OS, and the apt repository on Ubuntu.

CVODES

Numerical integration is computed using the CVODES Sundials package. It is included in the 'libraries' folder, and is also available for download from http://computation.llnl.gov/casc/sundials/main.html.

Note: CVODES should be compiled using the -fPIC flag to ensure cross-platform compatibility.

Python

In order to use LNA++ with Python, you must first install the necessary scientific computing modules, listed below. Python libraries can be installed using

package managers such as apt on Ubuntu, or macports on mac os, or using the Pip3 Python package manager.

Note: LNA++ is currently only compatible with Python 3.0 or greater. All libraries must thus be the Python3-compatible version.

Numpy
Numpy is a widely used mathematics and numerical library.

Sympy
Sympy provides a framework for symbolic/algebraic computations, and is necessary for computing the linear noise approximation for arbitrary models.

(optional) pylab
Pylab provides functions for plotting vectors and matrices, and might thus be useful.

Matlab

Usage
In this section we describe how to use LNA++ to create Matlab executable models and Python modules.

Overview
LNA++ combines a user-defined model specification, written as a short Matlab or Python script, with a C++ library in order to generate executables that can be quickly run without any further need for compilation or module creation.

The LNA requires the definition of a few model-specific components in order to be computed:

State variables
These are the species in the chemical reaction network, e.g. mRNA or proteins.

Stoichiometric matrix
This is the N by M matrix of stoichiometric coefficients describing the chemical reaction network, where N is the number of species and M is the number of reactions. Thus entry (**i**,**j**) corresponds to the net change in the number of molecules of species **i**, due to the firing of reaction **j**.

Reaction fluxes
This is a vector function of length M, describing the rate of each reaction as a function of the current state of the system, the kinetic constants, and potentially with explicit time dependence.

Model parameters
The model parameters are the set of model-specific constants that influence the rates of possible reactions. Sensitivities of the mean and

temporal cross-covariance matrices are computed with respect to the model parameters.

Matlab

When using LNA++ with Matlab, the quantities described above must be defined by performing these steps:

- Define the stoichiometric matrix S. This matrix should have N rows and M columns, where N is the number of species and M is the number of reactions in the chemical reaction network.
- Define symbolic variables such as the state variables and the model parameters using the *syms* statement (requires the Symbolic Toolbox)
- Define the reaction flux vector function *F*, e.g. using an anonymous function or a handle to a function defined in an m-file. The reaction flux vector should have the signature *F(phi, t, Theta),* where *phi* is the vector of state variables, *t* the current time, and *Theta* the vector of model parameters.
- Define *phi* as a vector of all of the state variables, *Theta* as the vector of model parameters, and *npar* is the (integer) number of parameters.

Note: It is recommended to save the model as a script in the 'models' subdirectory of the LNA++ root directory. This ensures the correct relative path for the subsequent build steps.

See **BirthDeath.m** in 'models' for an example of how this is done.

Paths

After running the functions for generating the mex file, the resulting code, executable, and wrapper function are placed in a subdirectory with the name of the model as specified. Make sure that the wrapper function and executable are on the matlab path, e.g. by changing to the directory of the model or by adding them to the path using *addpath.*

Place the script containing the model defined in the steps above into the 'models' directory, and use this directory as the working directory for the following steps.

Creating the Matlab executable (Mex) file

Once the model has been defined using the steps above, it must first be converted into m-files by solving for the necessary LNA components, and then converted into C code. Finally it is compiled together with the LNA++ source, using the Matlab compiler *mex,* to generate the final executable model.

generateLNAComponents

The first step is to generate the symbolic components used for the computation of the LNA and the sensitivities. This is achieved by calling *generateLNAComponents*. This Matlab function has the following arguments:

1. *modelName*: the name of the model to be used for the executable. Do not use spaces.
2. *S:* the stoichiometric matrix, as defined above

3. *reactionFlux:* the reaction flux vector function, defined above
4. *phi:* the vector of symbolic state variables, defined above
5. *Theta:* the vector of symbolic model parameters, defined above
6. *computeSS:* this argument is optional and can have only one of the values "Y0", "V0", "BOTH", or "NONE".
    a. *Y0*: LNA++ will attempt to calculate the steady state value of the MRE of the stochastic system specified. If Y0 is not explicitly provided when invoking the model, the computed value will be automatically substituted.
    b. *V0*: LNA++ will attempt to calculate the steady state value of the variance of the stochastic system specified. If V0 is not explicitly provided when invoking the model, the computed value will be automatically substituted.
    c. *BOTH*: LNA++ will compute both V0 and Y0 if possible. If Y0 or V0 is not explicitly provided when invoking the model, the computed values will be automatically substituted.
    d. *NONE*: LNA++ will not compute any steady state values. Invocation of the model without explicitly specifying the initial conditions for Y0 and V0 will result in an error message being generated.

generateLNAComponents computes the LNA components, solves for steady states and steady state sensitivities, and generates the C code for use in the subsequent compilation step.

compileLNA

The next step is to combine the C code with the LNA++ source to create the final executable.

compileLNA takes the following arguments:

1. *modelName*: the name of the model to be used for the executable. Do not use spaces.
2. *S:* the stoichiometric matrix, as defined above
3. *Npar:* the number of model parameters

Matlab executable

compileLNA generates a mex file and a matlab wrapper, which provides help messages and removes singleton dimensions for more convenient output. These are placed in the model subdirectory corresponding to the *modelName* specified. To execute them, copy them to the desired final folder, or add them to the Matlab path. Usage of the generated function is specified in the help for that function.

See BirthDeath in 'models', and the generated *BirthDeath_LNA.m* for an example of the generated wrapper function and its invocation.

Using the generated mex file

The mex file corresponding to a particular model can be invoked with between 2 and 6 arguments, which must occur in the correct order:

Inputs

1. **Theta:** the vector of model parameters, with the same dimension as specified during the model construction
2. **Time**: a vector of times at which the result of the model simulation is to be outputted. The first time is always zero.
3. **obsVar (optional)**: a vector of indices corresponding to the species which should be outputted. The numbering of species matches the numbering of species in the stoichiometric matrix. If omitted, all species are outputted.
4. **Merr (optional):** a vector of numbers (possibly of length 1) corresponding to estimated measurement error of each species, specified as a variance. The measurement error is added to diagonal of the temporal cross-covariance matrix. If only one number is specified, this number is used for all species. If multiple numbers are specified, the number specified must equal the number of observed variables. If omitted, measurement error is assumed to be zero for all species.
5. **Y0 (optional):** a vector of initial conditions for each of the species. The length must be the same as the number of species. If omitted, the steady-state initial conditions are assumed, if they were computed when the model was created; otherwise an error is generated requesting the initial conditions to be specified.
6. **V0 (optional):** a vector of initial conditions for the (co-)variances between all species. They are specified in column-major order, i.e. for a 3x3 matrix [[V_11, V_12, V_13]; [V_21, V_22, V_23]; [V_31, V_32, V_33]], the initial conditions V0 would be specified in the order [V_11, V_12, V_22, V_13, V_23, V_33]. Only the upper triangular portion of the initial covariance matrix is specified due to symmetry. If omitted, the steady-state values are substituted, if they were computed when the model was created; otherwise an error is generated.

   For convenience, the function *toLinear* is provided as a Matlab script, which converts a symmetric covariance matrix into a column-major-ordered vector.

Outputs

If only 2 output arguments are specified, the computation engine does not compute sensitivities, which can be substantially faster. If 3 or 4 outputs are specified, first order sensitivities are computed. If 5 or 6 outputs are specified, first and second order sensitivities are computed which can take longer.

1. **MRE:** an NxT matrix corresponding to the solution of the macroscopic rate equation for all observed variables. N is the number of observed variables (**obsVar**), and T the number of time points specified by **Time.**
2. **Sigma:** an NxNxT array corresponding to the temporal cross-covariance of all observed species at all time points. N is the number of observed variables (**obsVar**), and T the number of time points specified by **Time.**

3. **dMRE (optional):** an NxTxd array corresponding to the first order sensitivities of the MRE to each of the model parameters over all time points. N is the number of observed variables (**obsVar**), T the number of time points specified by **Time,** and d is the number of model parameters + the number of species. The sensitivities are ordered such that the first sensitivities are of the MRE with respect to the model parameters, followed by sensitivities with respect to the initial conditions of each species.
4. **dSigma (optional):** an NxNxTxd array corresponding to the first order sensitivities of the temporal cross-covariance matrix at all timepoints for the observed species. N is the number of observed variables (**obsVar**), T the number of time points specified by **Time,** and d is the number of model parameters + the number of species. The sensitivities are ordered such that the first sensitivities are of the MRE with respect to the model parameters, followed by sensitivities with respect to the initial conditions of each species.
5. **d2MRE (optional):** an NxTxdxd array corresponding to the second order sensitivities of the MRE at all timepoints for the observed species. N is the number of observed variables (**obsVar**), T the number of time points specified by **Time,** and d is the number of model parameters + the number of species. The sensitivities are ordered such that the first sensitivities are of the MRE with respect to the model parameters, followed by sensitivities with respect to the initial conditions of each species.
6. **d2Sigma (optional):** an NxNxTxdxd array corresponding to the second order sensitivities of the temporal cross-covariance matrix at all timepoints for the observed species. N is the number of observed variables (**obsVar**), T the number of time points specified by **Time,** and d is the number of model parameters + the number of species. The sensitivities are ordered such that the first sensitivities are of the MRE with respect to the model parameters, followed by sensitivities with respect to the initial conditions of each species.

Python

When using LNA++ with Python, the quantities described above must be defined by performing these steps:

- Define the stoichiometric matrix S, by creating an object of type **sympy.Matrix**. This matrix should have N rows and M columns, where N is the number of species and M is the number of reactions in the chemical reaction network.
- Define symbolic variables such as the state variables and the model parameters using the *symbols* statement from the **sympy** Python package.
- Define the reaction flux vector function *F*, e.g. using an anonymous (lambda) function or a function defined in a Python module. The reaction flux vector should have the signature *F(phi, t, Theta),* where *phi* is the vector of state variables, *t* the current time, and *Theta* the vector of model parameters.

- Define *phi* as a vector of all of the state variables, and *Theta* as the vector of model parameters.

Note: It is recommended to save the model as a script in the 'models' subdirectory of the LNA++ root directory. This ensures the correct relative path for the subsequent build steps.

Creating the Python module

The Python implementation of LNA++ is similar to the MATLAB implementation described above. Functions for generating the Python module are provided in the *LNA* module.

The model components need to be defined as described above. Then, call *generateLNAComponents* from the *LNA* module with the following arguments:

1. **Model**: the name of the model, as a string without spaces.
2. **S**: the stoichiometric matrix
3. **F:** the reaction flux vector function
4. **Phi:** the vector of state variables / species
5. **Theta:** the vector of model parameters
6. **computeSS (optional)**: one of the either 'BOTH', 'NONE', 'Y0', or 'V0'. If 'BOTH', LNA++ will attempt to compute initial conditions for both variance and the MRE. If Y0 it will attempt to compute only the MRE, and if V0 only the variance. If 'NONE' both variance and MRE initial conditions must be explicitly specified when executing the simulation. If omitted, the default is 'NONE'.

The function *generateLNAComponents* returns one argument containing the generated objects that need to be converted into C code for the module generation.

Assuming the objects are stored in a variable *objs*, the next step is to call the function *compileLNA* with the following arguments:

1. **Model:** same as above
2. **S:** same as above
3. **objs:** the objects generated by *generateLNAComponents*
4. **npar:** the number of model parameters, i.e. *len(Theta)*

After these two steps, a Python module is created in the *modules* subdirectory. The module is named *modelLNA,* where model is replaced by the name of the model specified in the two functions above. The module can be imported using *import*, as long as the module is on the Python module path. To add this directory to the module path either modify the value of *sys.path* or add the modules directory to the PYTHONPATH environmental variable.

Using the created Python module

The Python module for a given model must first be imported as described above. The model is simulated using the *LNA* function in the module. For example if the

model is called 'myModel' then the generated module is called 'myModelLNA' and the *LNA* function can be imported using the statement

from myModelLNA import LNA

Input

The *LNA* function requires 2-8 input arguments, as follows. Arguments 3-8 are optional, and can be specified either by position or as formal (named) arguments.

1. **Theta:** a list of model parameters
2. **Time:** a list of time points for outputting the simulation results
3. **obsVar (optional):** a list of observed variables to be output. The default is to output all species.
4. **merr (optional):** either a single number specifying the measurement error (variance) to be added to the covariance of all observed species, or a list of numbers of the same length as the number of observed variables, specifying each measurement error individually. The default is zero observation error.
5. **Y0 (optional):** the initial conditions for the MRE. If initial conditions are not specified, but were computed using the *computeSS* flag at the time of the module creation, then the steady state values are implicitly used. If the initial conditions were not computed when generating the module, then an error occurs if no initial conditions are specified.
6. **V0 (optional):** a vector of initial conditions for the (co-)variances between all species. They are specified in column-major order, i.e. for a 3x3 matrix [[V_11, V_12, V_13]; [V_21, V_22, V_23]; [V_31, V_32, V_33]], the initial conditions V0 would be specified in the order [V_11, V_12, V_22, V_13, V_23, V_33]. Only the upper triangular portion of the initial covariance matrix is specified due to symmetry. For convenience, the function *toLinear* is provided in the LNA module, which converts a symmetric covariance matrix into a column-major-ordered vector.

   If initial conditions are not specified, but were computed using the *computeSS* flag at the time of the module creation, then the steady state values are implicitly used. If the initial conditions were not computed when generating the module, then an error occurs if no initial conditions are specified.

7. **computeSens (optional):** a Boolean variable indicating whether the first order sensitivities should be computed. By default no first order sensitivities are computed.
8. **computeSens2**: a Boolean variable indicating whether the second order sensitivities should be computed. By default no second order sensitivities are computed.

Output

The *LNA* function returns between 2 and 6 output arguments.

1. **MRE:** an NxT matrix corresponding to the solution of the macroscopic rate equation for all observed variables. N is the number of observed variables (**obsVar**), and T the number of time points specified by **Time.**
2. **Sigma:** an NxNxT array corresponding to the temporal cross-covariance of all observed species at all time points. N is the number of observed variables (**obsVar**), and T the number of time points specified by **Time.**
3. **dMRE (if computeSens):** an NxTxd array corresponding to the first order sensitivities of the MRE to each of the model parameters over all time points. N is the number of observed variables (**obsVar**), T the number of time points specified by **Time,** and d is the number of model parameters + the number of species. The sensitivities are ordered such that the first sensitivities are of the MRE with respect to the model parameters, followed by sensitivities with respect to the initial conditions of each species.
4. **dSigma (if computeSens):** an NxNxTxd array corresponding to the first order sensitivities of the temporal cross-covariance matrix at all timepoints for the observed species. N is the number of observed variables (**obsVar**), T the number of time points specified by **Time,** and d is the number of model parameters + the number of species. The sensitivities are ordered such that the first sensitivities are of the MRE with respect to the model parameters, followed by sensitivities with respect to the initial conditions of each species.
5. **d2MRE (if computeSens2):** an NxTxdxd array corresponding to the second order sensitivities of the MRE at all timepoints for the observed species. N is the number of observed variables (**obsVar**), T the number of time points specified by **Time,** and d is the number of model parameters + the number of species. The sensitivities are ordered such that the first sensitivities are of the MRE with respect to the model parameters, followed by sensitivities with respect to the initial conditions of each species.
6. **d2Sigma (if computeSens2):** an NxNxTxdxd array corresponding to the second order sensitivities of the temporal cross-covariance matrix at all timepoints for the observed species. N is the number of observed variables (**obsVar**), T the number of time points specified by **Time,** and d is the number of model parameters + the number of species. The sensitivities are ordered such that the first sensitivities are of the MRE with respect to the model parameters, followed by sensitivities with respect to the initial conditions of each species.

   Note: if computeSens2 is True, the first order sensitivities are always also computed and outputted.

Examples

Birth-beath model

One example is provided with LNA++. It is the case of a simple birth death model with two species: mRNA and protein. The model is described in detail in the supplementary materials of the LNA++ paper (see references).

The stoichiometric matrix is given by:

S = [[1 -1 0 0]; [0 0 1 -1]]

The reaction flux vector is given by:
F = [k_m, g_m*m, k_p*m, g_p*p]

with model parameters
k_m: production rate of mRNA
k_p: production rate of protein
g_m: degradation rate of mRNA
g_p: degradation rate of protein

A Matlab implementation of this model, along with code to perform simulations and generate plots for the output is included in the script 'models/BirthDeath.m', and a Python implementation of the same in 'models/BirthDeath.py'.

References

- Manuscript
- http://sourceforge.net/projects/blitz/.
- Komorowski papers