# 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 run integrated into either 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 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 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 libraries, 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 algebraic manipulation, and is necessary for computing the linear noise approximation for arbitrary models.

### (optional) Matplotlib

Matplotlib provides functions for plotting vectors and matrices, and might thus be useful.

## Matlab

In order to use LNA++ with Matlab, a few toolboxes are required.

### Symbolic Toolbox

The symbolic toolbox is needed in order to compute the analytical components of the LNA. If the symbolic toolbox is not available, it is still possible to use Python instead to first generate the LNA C functions, and then to subsequently compile the LNA Matlab executable as normal as described below.

### Coder

In order to convert auto-generated Matlab functions into C code, the Matlab Coder is used. If it is not available, Python can be used instead, as described below.

# Usage

In this section we describe how to use LNA++ to create Matlab executable models, or 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 ij 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.

## Use cases

### Matlab

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

* Define the stoichiometric matrix S
* 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, e.g. using an anonymous function or a handle to a function defined in an m-file. The reaction flux vector should have the form *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

The Matlab scripts for generating the Mex file must be on the Matlab path. This can be achieved by simply executing *addpath(‘matlab’).*

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 m-files by calling *generateLNAComponents*. This Matlab script 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

*generateLNAComponents* computes the LNA components, solves for steady states and steady state sensitivities, generates m-files, and places them in the Matlab subfolder of the model folder for the current model.

##### compileLNA

The next step is to convert the generated Matlab functions into C code, and combine this with 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

After running *generateLNAComponents* and *compileLNA*, the generated mex file and a wrapper providing help functions, and removing singleton dimensions of the output structures (which occur if specifying observed variables), 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.

## Python

### Creating the Python module

### Input

### Output

# Examples

## Birth Death model

* Examples directory
* BirthDeath.m
* BirthDeath.py

# References

<http://sourceforge.net/projects/blitz/>.