Documentation of LNA++

Justin Feigelman[1,2,3], Daniel Weindl[2], Fabian J. Theis[1,2], Jan Hasenauer[1,2], and Carsten Marr[1]

[1]Helmholtz Zentrum München - German Research Center for Environmental Health, Institute of Computational Biology, Ingolstädter Landstraße 1, 85764 Neuherberg Germany
[2]Technische Universität München, Center for Mathematics, Chair of Mathematical Modeling of Biological Systems, Boltzmannstraße 3, 85748 Garching, Germany
[3]ETH Zürich, Institute of Molecular Systems Biology, Auguste-Piccard-Hof 1, 8093 Zürich, Switzerland

Contents

1 Toolboxes for Linear Noise Approximation

LNA++ is a tool designed to efficiently compute the linear noise approximation to a chemical reaction network, along with the first and second order sensitivities of the mean, covariance and temporal cross-covariance matrix. Furthermore, LNA++ facilitates parameter inference on time series data by providing first and second order sensitivities of the log-likelihood with respect to model parameters. To ensure computational efficiency, the numerical simulation is implemented in C/C++. 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 provides the mathematical background and describes the library requirements, the installation and the usage of LNA++.

2 Mathematical background

2.1 Notation

Throughout this document we will indicate matrices using double struck notation, and vectors using boldface; time derivatives are indicated using the conventional dot notation, i.e. $\dot{x} := \frac{d}{dt}x(t)$.

2.2 Chemical reaction networks

We define a chemical reaction network (CRN) to be a set of chemical reactions defined for a set of chemical species. Each reaction consumes a set of educts and yields a set of products. Consider a CRN with R reactions, denoted $\mathcal{R}_1, \ldots, \mathcal{R}_R$, between a set of N chemical species. The j^{th} reaction, $\mathcal{R}_j$, can be summarized succinctly as

$$\mathcal{R}_j : \sum_{i=1}^{N} \mathbb{S}_{ij}^{-} X_i \longrightarrow \sum_{i=1}^{N} \mathbb{S}_{ij}^{+} X_i. \tag{1}$$

The matrices $\mathbb{S}^{-} = \{\mathbb{S}_{ij}^{-}\}$ and $\mathbb{S}^{+} = \{\mathbb{S}_{ij}^{+}\}$, with $i = 1, \ldots, N$ and $j = 1, \ldots, R$ denote the stoichiometric matrix of educts and products, i.e. the number of molecules of each species consumed or produced in each reaction, respectively. The net stoichiometric matrix is defined as $\mathbb{S} := \mathbb{S}^{+} - \mathbb{S}^{-}$.

2.3 Linear Noise Approximation

In the linear noise approximation, the stochastic system dynamics of the CRN are approximated by a set of ordinary differential equations which capture the evolution of the mean and temporal cross-covariance of all species of the CRN. The molecular copy number X_i of the i^{th} species can be converted to a molecular concentration, denoted x_i, by dividing by the size of the reaction volume Ω, such that $x_i = X_i/\Omega, i = 1, \ldots, N$. In the following we summarize the LNA for CRNs, for more details see, e.g. Van Kampen [1] and Komorowski *et al.* [2, 3]. In the LNA, the molecular concentration vector of the system $\mathbf{x}(t) = (x_1(t), \ldots, x_N(t))^T$ is approximated by the sum of a deterministic mean term $\boldsymbol{\phi}(t)$ and a stochastic fluctuation term $\boldsymbol{\eta}(t)$:

$$\mathbf{x}(t) = \boldsymbol{\phi}(t) + \boldsymbol{\eta}(t). \tag{2}$$

The deterministic component $\boldsymbol{\phi}(t)$ satisfies the macroscopic rate equations (MRE):

$$\dot{\boldsymbol{\phi}}(t) = \mathbb{S}\mathbf{F}(\boldsymbol{\phi}, \boldsymbol{\Theta}, t) \tag{3}$$

where $\mathbf{F} = (F_1, ..., F_R)^T$ is the vector of reaction fluxes for each of the R reactions at time t parametrized by model parameters $\boldsymbol{\Theta} = (\theta_1, \ldots, \theta_P)$. The stochastic fluctuation term $\boldsymbol{\eta}(t)$ obeys the stochastic differential equation:

$$d\boldsymbol{\eta} = \mathbb{A}(\boldsymbol{\phi}, \boldsymbol{\Theta}, t)\boldsymbol{\eta}\, dt + \mathbb{E}(\boldsymbol{\phi}, \boldsymbol{\Theta}, t) d\mathbf{W} \tag{4}$$

with

$$\mathbb{A}(\boldsymbol{\phi}, \boldsymbol{\Theta}, t) = \mathbb{S}\frac{\partial}{\partial \boldsymbol{\phi}}\mathbf{F}(\boldsymbol{\phi}, \boldsymbol{\Theta}, t) \tag{5}$$

$$\mathbb{E}(\boldsymbol{\phi}, \boldsymbol{\Theta}, t) = \mathbb{S}\sqrt{\mathrm{diag}(\mathbf{F}(\boldsymbol{\phi}, \boldsymbol{\Theta}, t))} \tag{6}$$

where $\mathbf{W}$ is a N-dimensional Wiener process. The covariance of the system at time t, $\mathbb{V}(t) = \mathrm{cov}(\mathbf{x}(t))$, i.e. $\mathbb{V}$ is the matrix with entries $\mathbb{V}_{ij}(t) = \mathrm{cov}(\mathbf{x}_i(t), \mathbf{x}_j(t))$, is given by the solution of

$$\dot{\mathbb{V}}(t) = \mathbb{A}\mathbb{V} + \mathbb{V}\mathbb{A}^T + \mathbb{E}\mathbb{E}^T. \tag{7}$$

Temporal cross-covariances $\mathrm{cov}(\mathbf{x}(s), \mathbf{x}(t))$ between the system at times s and $t, t \geq s$, are derived by applying the fundamental solution matrix $\mathbb{G}(s, t)$, defined by (9), to the system variance at time s:

$$\mathrm{cov}\left(\mathbf{x}(s), \mathbf{x}(t)\right) = \mathbb{V}(s)\mathbb{G}(s, t)^T \tag{8}$$

$$\dot{\mathbb{G}}(s, t) = \mathbb{A}\mathbb{G}(s, t) \tag{9}$$

$$\mathbb{G}(s, s) = \mathbb{I}_N. \tag{10}$$

The variance $\mathbb{V}$ and the temporal cross-covariance $\mathrm{cov}(\mathbf{x}(s), \mathbf{x}(t))$ can be employed to evaluate the correlation of the system across multiple time points. We denote the vector of stacked states at different time points $t_1, \ldots, t_M$ by

$$\tilde{\mathbf{x}} = (x_1(t_1), ..., x_N(t_1), x_1(t_2), ..., x_N(t_2), ..., x_1(t_M), ..., x_N(t_M))^T. \tag{11}$$

The covariance of $\tilde{\mathbf{x}}$ is given by the temporal cross-covariance block matrix $\Sigma = \{\Sigma^{kl}\}_{k,l=1}^{M}$. In matrix notation we obtain

$$\Sigma = \mathrm{cov}(\tilde{\mathbf{x}}) = \begin{pmatrix} \mathbb{V}(t_1) & \mathrm{cov}(\mathbf{x}(t_1), \mathbf{x}(t_2)) & ... & \mathrm{cov}(\mathbf{x}(t_1), \mathbf{x}(t_M)) \\ \mathrm{cov}(\mathbf{x}(t_2), \mathbf{x}(t_1)) & \mathbb{V}(t_2) & ... & \mathrm{cov}(\mathbf{x}(t_2), \mathbf{x}(t_M)) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{cov}(\mathbf{x}(t_M), \mathbf{x}(t_1)) & \mathrm{cov}(\mathbf{x}(t_N), \mathbf{x}(t_2)) & ... & \mathbb{V}(t_M) \end{pmatrix}.$$

The temporal cross-covariance block matrix is constructed by setting $\Sigma^{kk} = \mathbb{V}(t_k), k = 1, \ldots, M$, along the diagonal, and

$$\begin{aligned} \Sigma^{kl} &= \mathrm{cov}(\mathbf{x}(t_k), \mathbf{x}(t_l)) \\ &= \mathbb{V}(t_k)\mathbb{G}(t_k, t_l)^T, k < l \leq M \end{aligned} \tag{12}$$

for the off-diagonal elements.

2.4 Computing the likelihood of a trajectory

The likelihood of a trajectory $\tilde{\mathbf{x}}$ as defined in (11) consisting of observations at M discrete time points, conditional on the model parameters $\boldsymbol{\Theta}$, is given by a multivariate normal distribution with stacked mean $\tilde{\boldsymbol{\phi}} = (\boldsymbol{\phi}_1(t_1), \ldots, \boldsymbol{\phi}_N(t_1), \ldots, \boldsymbol{\phi}_1(t_M), \ldots, \boldsymbol{\phi}_N(t_M))^T$ obtained by solving (3), and cross-covariance matrix Σ computed via (7), and (8). The likelihood of $\boldsymbol{\Theta}$ is given by:

$$L(\boldsymbol{\Theta}) = P(\tilde{\mathbf{x}}|\boldsymbol{\Theta}) = (2\pi)^{-M/2} |\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}})^T \Sigma^{-1} (\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}})\right\} \tag{13}$$

where $|\Sigma|$ denotes the determinant of Σ, and ϕ, Σ are functions of $\boldsymbol{\Theta}$. Similarly, the log-likelihood of $\boldsymbol{\Theta}$, $\ell(\boldsymbol{\Theta}) := \log L(\boldsymbol{\Theta})$, is given by:

$$\ell(\boldsymbol{\Theta}) = -\frac{N}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}})^T \Sigma^{-1} (\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}}) \tag{14}$$

2.5 First order sensitivities

2.5.1 Analytical

The sensitivity of the log-likelihood with respect to the k^{th} model parameter, θ_k, is given by $\frac{d\ell}{d\theta_k}$. For the following we introduce the shorthand notation $\phi'_i := \frac{d}{d\theta_i}\phi$, $\phi''_{ij} := \frac{d^2}{d\theta_i d\theta_j}\phi$, and similarly for all components of the LNA.

The first order sensitivity of the log-likelihood with respect to θ_i is computed as:

$$\begin{aligned}\frac{d\ell}{d\theta_i} &= -\frac{1}{2}\frac{d\log|\Sigma|}{d\theta_i} - \frac{1}{2}\left((-\tilde{\boldsymbol{\phi}}'_i)^T\Sigma^{-1}(\tilde{\mathbf{x}}-\tilde{\boldsymbol{\phi}}) + (\tilde{\mathbf{x}}-\tilde{\boldsymbol{\phi}})^T\frac{d\Sigma^{-1}}{d\theta_i}(\tilde{\mathbf{x}}-\tilde{\boldsymbol{\phi}}) + (\tilde{\mathbf{x}}-\tilde{\boldsymbol{\phi}})^T\Sigma^{-1}(-\tilde{\boldsymbol{\phi}}'_i)\right)\\ &= -\frac{1}{2}\mathrm{Tr}\left(\Sigma^{-1}\Sigma'_i\right) + (\tilde{\boldsymbol{\phi}}'_i)^T\Sigma^{-1}(\tilde{\mathbf{x}}-\tilde{\boldsymbol{\phi}}) + \frac{1}{2}(\tilde{\mathbf{x}}-\tilde{\boldsymbol{\phi}})^T\left(\Sigma^{-1}\Sigma'_i\Sigma^{-1}\right)(\tilde{\mathbf{x}}-\tilde{\boldsymbol{\phi}})\end{aligned} \tag{15}$$

where Σ'_k and $\boldsymbol{\phi}'_k$ are computed using the forward sensitivity equations below in conjunction with (3) and (8):

$$\dot{\boldsymbol{\phi}}'_k(t) = \frac{d}{d\theta_k}\dot{\boldsymbol{\phi}}(t) = \mathbb{S}\frac{d}{d\theta_k}\mathbf{F}(\boldsymbol{\phi},\boldsymbol{\Theta},t) \tag{16}$$

$$= \mathbb{S}\left(\frac{\partial\mathbf{F}}{\partial\boldsymbol{\phi}}\boldsymbol{\phi}'_k + \frac{\partial\mathbf{F}}{\partial\theta_k}\right) \tag{17}$$

$$= \mathbb{A}\phi'_k + \mathbb{S}\frac{\partial\mathbf{F}}{\partial\theta_k} \tag{18}$$

$$\frac{d}{d\theta_k}\mathrm{cov}\left(\mathbf{x}(s),\mathbf{x}(t)\right) = \mathbb{V}'_k\mathbb{G}(s,t)^T + \mathbb{V}(s)\mathbb{G}'_k(s,t)^T \tag{19}$$

$$\dot{\mathbb{V}}'_k = \frac{d}{d\theta_k}\dot{\mathbb{V}}(t) = \mathbb{A}'_k\mathbb{V} + \mathbb{A}\mathbb{V}'_k + \mathbb{V}'_k\mathbb{A}^T + \mathbb{V}{\mathbb{A}'_k}^T + \mathbb{E}'_k\mathbb{E}^T + \mathbb{E}{\mathbb{E}'_k}^T \tag{20}$$

$$= \frac{d}{d\theta_k}\dot{\mathbb{V}}(t) = \mathbb{A}'_k\mathbb{V} + \mathbb{A}\mathbb{V}'_k + \mathbb{V}'_k\mathbb{A}^T + \mathbb{V}{\mathbb{A}'_k}^T + (\mathbb{E}\mathbb{E}^T)'_k \tag{21}$$

with

$$(\mathbb{E}\mathbb{E}^T)'_k := \frac{d(\mathbb{E}\mathbb{E}^T)}{d\theta_k} = \frac{\partial(\mathbb{E}\mathbb{E}^T)}{\partial\phi}\phi'_k + \frac{\partial(\mathbb{E}\mathbb{E}^T)}{\partial\theta_k}.$$

Note that (20) and (21) are alternative formulations. (21) is more suited for numerical implementations as it avoids fractions with state variables (which might be zero) in the denominator.

2.5.2 Finite difference approximation

The forward finite difference approximation of a function $f(\theta)$ is obtained by Taylor expanding the function about the point θ:

$$f(\theta + \Delta\theta) = f(\theta) + f'(\theta)\Delta\theta + \mathcal{O}(\Delta\theta^2). \tag{22}$$

Ignoring terms with power of $\Delta\theta$ greater than one, $f'(\theta) \approx \frac{f(\theta+\Delta\theta)-f(\theta)}{\Delta\theta}$. Thus the computation of the finite difference approximation requires one additional evaluation of f at $\theta + \Delta\theta$. Likewise, for a function $f(\boldsymbol{\Theta}), \boldsymbol{\Theta} \in \mathbb{R}^d$, the finite difference approximation to the gradient $\nabla_{\boldsymbol{\Theta}} f$ requires an additional d evaluations of f.

2.6 Second order sensitivities

2.6.1 Analytical

The matrix of second order sensitivities, i.e. the Hessian, of the log-likelihood is computed as $\frac{\partial^2 \ell}{\partial \boldsymbol{\Theta}^2} = \{\frac{\partial^2 \ell}{\partial \theta_i \partial \theta_j}\}_{i,j=1}^d$, where the $(i,j)^{\text{th}}$ term is given by

$$\begin{aligned}
\frac{d^2\ell}{d\theta_i d\theta_j} = &-\frac{1}{2}\mathrm{Tr}\left(-\Sigma^{-1}\Sigma_j'\Sigma^{-1}\Sigma_i' + \Sigma^{-1}\Sigma_{ij}''\right) + (\tilde{\boldsymbol{\phi}}_{ij}'')^T\Sigma^{-1}\left(\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}}\right) \\
&- (\tilde{\boldsymbol{\phi}}_i')^T\left(\Sigma^{-1}\Sigma_j'\Sigma^{-1}\right)(\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}}) - \tilde{\boldsymbol{\phi}}_i'^T\Sigma^{-1}\tilde{\boldsymbol{\phi}}_j' - (\tilde{\boldsymbol{\phi}}_j')^T(\Sigma^{-1}\Sigma_i'\Sigma^{-1})(\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}}) \\
&+ \frac{1}{2}(\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}})^T\Sigma^{-1}(-2\Sigma_j'\Sigma^{-1}\Sigma_i' + \Sigma_{ij}'')\Sigma^{-1}(\tilde{\mathbf{x}} - \tilde{\boldsymbol{\phi}}).
\end{aligned} \tag{23}$$

It depends on the higher order sensitivities of the macroscopic rate equation ($\boldsymbol{\phi}_{ij}''$) and of the temporal cross-covariance matrix (Σ_{ij}''). Recalling the equations for the first order sensitivities (17), (19) and (5), we find that the second order sensitivity of the MRE ($\boldsymbol{\phi}_{ij}''$) is given by:

$$\begin{aligned}
\dot{\boldsymbol{\phi}}_{ij}'' &= \frac{d}{d\theta_j}\dot{\boldsymbol{\phi}}_i'(t) \\
&= \mathbb{A}_j'\boldsymbol{\phi}_i' + \mathbb{A}\boldsymbol{\phi}_{ij}'' + \mathbb{S}\frac{d}{d\theta_j}\frac{\partial \mathbf{F}}{\partial \theta_i} \\
&= \left(\frac{\partial \mathbb{A}}{\partial \boldsymbol{\phi}}\boldsymbol{\phi}_j' + \frac{\partial \mathbb{A}}{\partial \theta_j}\right)\boldsymbol{\phi}_i' + \mathbb{A}\boldsymbol{\phi}_{ij}'' + \mathbb{S}\left(\frac{\partial}{\partial \boldsymbol{\phi}}\frac{\partial \mathbf{F}}{\partial \theta_i}\frac{\partial \boldsymbol{\phi}}{\partial \theta_j} + \frac{\partial}{\partial \theta_j}\frac{\partial \mathbf{F}}{\partial \theta_i}\right) \\
&= \left(\frac{\partial \mathbb{A}}{\partial \boldsymbol{\phi}}\boldsymbol{\phi}_j' + \frac{\partial \mathbb{A}}{\partial \theta_j}\right)\boldsymbol{\phi}_i' + \mathbb{A}\boldsymbol{\phi}_{ij}'' + \frac{\partial \mathbb{A}}{\partial \theta_i}\boldsymbol{\phi}_j' + \mathbb{S}\frac{\partial^2 \mathbf{F}}{\partial \theta_j \partial \theta_i}.
\end{aligned} \tag{24}$$

The computation of the second order sensitivities of the temporal cross-covariance matrix is slightly more involved, and requires also the higher order sensitivities of the fundamental matrix ($\mathbb{G}''_{ij}$). From (8)-(10) we find

$$\frac{d}{d\theta_i}\text{cov}(\mathbf{x}(s),\mathbf{x}(t)) = \mathbb{V}'_i(s)\mathbb{G}^T(s,t) + \mathbb{V}(s)\mathbb{G}'^T_i(s,t) \tag{25}$$

and

$$\frac{d^2}{d\theta_i d\theta_j}\text{cov}(\mathbf{x}(s),\mathbf{x}(t)) = \mathbb{V}''_{ij}(s)\mathbb{G}^T(s,t) + \mathbb{V}'_i(s)\mathbb{G}'^T_j(s,t) + \mathbb{V}'_j(s)\mathbb{G}'^T_i(s,t) + \mathbb{V}(s)\mathbb{G}''^T_{ij}(s,t). \tag{26}$$

From (20) we find

$$\begin{aligned}
\dot{\mathbb{V}}''_{ij} &= \frac{d}{d\theta_j}\dot{\mathbb{V}}'_i \\
&= \mathbb{A}''_{ij}\mathbb{V} + \mathbb{A}'_i\mathbb{V}'_j + \mathbb{A}'_j\mathbb{V}'_i + \mathbb{A}\mathbb{V}''_{ij} + \mathbb{V}''_{ij}\mathbb{A}^T + \mathbb{V}'_i\mathbb{A}'_j \\
&\quad + \mathbb{V}'_j\mathbb{A}'^T_i + \mathbb{V}(\mathbb{A}''_{ij})^T + \mathbb{E}''_{ij}(\mathbb{E})^T + \mathbb{E}'_i(\mathbb{E}'_j)^T + \mathbb{E}'_j(\mathbb{E}'_i)^T + \mathbb{E}(\mathbb{E}''_{ij})^T \qquad (27) \\
\mathbb{A}''_{ij} &= \frac{d}{d\theta_j}\left(\frac{\partial\mathbb{A}}{\partial\boldsymbol{\phi}}\boldsymbol{\phi}'_i + \frac{\partial\mathbb{A}}{\partial\theta_i}\right) \\
&= \left(\frac{\partial^2\mathbb{A}}{\partial\boldsymbol{\phi}^2}\boldsymbol{\phi}'_j + \frac{\partial^2\mathbb{A}}{\partial\theta_j\partial\boldsymbol{\phi}}\right)\boldsymbol{\phi}'_i + \frac{\partial\mathbb{A}}{\partial\boldsymbol{\phi}}\phi''_{ij} + \frac{\partial}{\partial\boldsymbol{\phi}}\frac{\partial\mathbb{A}}{\partial\theta_i}\frac{\partial\boldsymbol{\phi}}{\partial\theta_j} + \frac{\partial^2\mathbb{A}}{\partial\theta_j\partial\theta_i}. \qquad (28)
\end{aligned}$$

The sensitivities of the fundamental matrix $\mathbb{G}$ are computed using (8):

$$\begin{aligned}
\dot{\mathbb{G}}'_i(s,t) &= \mathbb{A}'_i(t)\mathbb{G}(s,t) + \mathbb{A}(t)\mathbb{G}'_i(s,t) \qquad (29) \\
\dot{\mathbb{G}}''_{ij}(s,t) &= \mathbb{A}''_{ij}(t)\mathbb{G}(s,t) + \mathbb{A}'_i(t)\mathbb{G}'_j(s,t) + \mathbb{A}'_j(t)\mathbb{G}'_i(s,t) + \mathbb{A}\mathbb{G}''_{ij}(s,t). \qquad (30)
\end{aligned}$$

2.6.2 Finite difference approximation

Consider a function $f(\boldsymbol{\Theta})$, where $\boldsymbol{\Theta} = (\theta_1,\ldots,\theta_d)^T$. The finite difference approximation of the second order sensitivity of $f(\boldsymbol{\Theta})$ with respect to the $(i,j)^{\text{th}}$ model parameters, $\frac{\partial^2 f}{\partial\theta_i\partial\theta_j}$, can be numerically computed using the following Taylor expansion about $\boldsymbol{\Theta}$ with small perturbations $\Delta\theta_i$ and $\Delta\theta_j$:

$$\begin{aligned}
\frac{\partial^2 f(\boldsymbol{\Theta})}{\partial\theta_i\partial\theta_j} \approx \frac{1}{2\Delta\theta_i\Delta\theta_j}\Big[&f(\boldsymbol{\Theta} + \mathbf{e_i}\Delta\theta_i + \mathbf{e_j}\Delta\theta_j) - f(\boldsymbol{\Theta} + \mathbf{e_i}\Delta\theta_i) - f(\boldsymbol{\Theta} + \mathbf{e_j}\Delta\theta_j) \\
&+2f(\boldsymbol{\Theta}) - f(\boldsymbol{\Theta} - \mathbf{e_i}\Delta\theta_i) - f(\boldsymbol{\Theta} - \mathbf{e_j}\Delta\theta_j) + f(\boldsymbol{\Theta} - \mathbf{e_i}\Delta\theta_i - \mathbf{e_j}\Delta\theta_j)\Big]
\end{aligned} \tag{31}$$

where $\mathbf{e_i}$ indicates the vector of zeros with a one in the i^{th} position. Thus the computation of the finite difference approximation (31) requires an additional 6 evaluations of the function $f(\boldsymbol{\Theta})$ (two of which are shared with the first order finite difference approximation).

2.6.3 Confidence intervals of parameter estimates

Confidence intervals can be computed using the maximum likelihood estimator (MLE) and the Hessian of the log-likelihood computed in (23). In particular, using the Hessian, it is possible to make the Laplace approximation to the probability density function (pdf) of the model parameters, i.e. approximating the pdf as a multivariate Gaussian locally around the MLE. The mean is given by the MLE $\hat{\boldsymbol{\Theta}}$, and the covariance matrix by the negative inverse Hessian, evaluated at $\hat{\boldsymbol{\Theta}}$:

$$\boldsymbol{\Theta} \sim \mathcal{N}\left(\hat{\boldsymbol{\Theta}}, -\left(\frac{\partial^2 \ell(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}^2}\right)^{-1}\Bigg|_{\boldsymbol{\Theta}=\hat{\boldsymbol{\Theta}}}\right) \tag{32}$$

3 LNA++ installation and requirements

3.1 Installation

Unpack the LNA++ archive or clone the LNA++ repository which contains all the necessary scripts and source code for building LNA++ Python modules or Matlab executable files. No further building or installation of LNA++ itself is necessary until the creation of the model executables, as described below.

3.2 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/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++` or `clang`.

Blitz++

Blitz++ version 1.0.1 is required. This version is included in the `./libraries` folder. Alternatively, Blitz++ can be downloaded from https://github.com/blitzpp/blitz. It is also available via macports on mac OS.

When using the provided source archive, extract it to `../libraries/blitz-1.0.1`, change to this folder and run:
`./configure --with-pic --prefix=‘pwd‘/../install/blitz-1.0.1 && make install`

For other versions of Blitz++ in non-default locations you need to adapt `include_dirs` and `library_dirs` in `python/setupTemplate.py` or the respective lines in `matlab/compileLNA.m`.

CVODES

Numerical integration is computed using the CVODES Sundials package. Source code is included in `./libraries`, and is also available from http://computation.llnl.gov/casc/sundials. To setup CVODES, unzip the source archive, change to the source directory and run:

```
./configure --with-pic --prefix=‘pwd‘/../install/cvodes-2.7.0
make && make install
```

Note: CVODES should be compiled using the `-fPIC` flag to ensure cross-platform compatibility. This can be enabled by specifying `--with-pic` to the configure script.

Python

In order to use LNA++ with Python, ensure you have installed the necessary scientific computing modules as listed below. Python libraries can be installed using package managers such as apt on Ubuntu, or macports/homebrew on Mac OS, or using the `pip3` Python package manager.

LNA++ is compatible with Python 3.0 or later. All libraries must thus be the Python3-compatible version.

Note for Ubuntu Users: The necessary Python packages can be installed using the command
`sudo apt install python3-numpy python3-sympy libsbml5-python python3-ipdb`

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.

libsbml Libsbml is required for translating SBML models for use with LNA++.

ipdb IPython debugger.

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

Matlab

In order to use LNA++ with Matlab, the Matlab *Symbolic Math Toolbox* is required for the analytic derivation of the LNA and its sensitivity equations. Matlab is not required when using only the Python version LNA++.

4 Usage

In this section we describe how to use LNA++ to create Matlab executable models and Python modules. More application examples are available in the `examples/` directory.

4.1 Overview

LNA++ converts a user-specified model, provided as an SBML model configuration file, into a set of symbolic functions for the time derivatives of means, temporal cross-covariances, and their sensitivities. These functions are converted to C code which is then compiled together with the core computation engine to generate either Matlab or Python executable files/modules. The system can then be simulated using these executables and user-specified initial conditions and model parameters.

The stoichiometric matrix and reaction propensity functions are extracted from a user-supplied SBML model file (see http://sbml.org/) or can be provided directly.

4.2 Matlab

To generate the Matlab executable simply call the function `generateLNA` specifying the path to the SBML input file, the model name and an optional argument indicating whether

to attempt to compute the steady state of the system as described below. See also `examples/runBirthDeath.m` for a detailed example of how this is done.

Building the model

To build the model call `generateLNA(sbmlFileName, modelName, computeSS)`

1. `sbmlFileName`: the path of the SBML model specification file
2. `modelName`: the desired name of the model
3. `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.

Remark: Analytical solution of the steady state equations can only be derived for some models. If the user selects `'Y0'`, `'V0'` or `'BOTH'` and the calculations fails, LNA++ reports an error.

Note for custom library paths and custom linker or compiler flags: It is also possible to specify custom flags to the linker or compiler, for example if CVODES or Blitz are not installed in a standard location. Linker and compiler flags can be specified using Name, Value argument pairs where Name is either `Linker_Flags`, or `Cpp_Flags`, e.g.,

- `generateLNA(sbmlFileName, modelName, computeSS, 'Linker_Flags', FLAGS)` or
- `generateLNA(sbmlFileName, modelName, computeSS, 'Cpp_flags', FLAGS)`.

Linker flags and compiler flags can both be specified, by providing both pairs of arguments.

Matlab executable

`generateLNA` 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 `models/` subdirectory with the name of the model as specified in the `generateLNA` call. To execute them, copy them to the desired final folder, or add them to the Matlab path using `addpath`. Usage of the generated function is specified in the help for that function.

See `examples/runBirthDeath.m`, and the generated `models/BirthDeath/BirthDeath_LNA.m` for an example of the generated wrapper function and its invocation.

Using the generated mex wrapper

The mex wrapper 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. `Timepoints`: a vector of times at which the result of the model simulation is to be outputted. The first time is always zero.
3. `V0` (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.
4. `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
5. `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.

6. `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.

$$\begin{bmatrix} \texttt{V\_11}, \texttt{V\_12}, \texttt{V\_13} \\ \texttt{V\_21}, \texttt{V\_22}, \texttt{V\_23} \\ \texttt{V\_31}, \texttt{V\_32}, \texttt{V\_33} \end{bmatrix}$$

with `V_12 = V_21`, `V_13 = V_31`, and `V_23 = V_32` 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. In the following N is the number of observed variables (`obsVar`), T is the number of time points specified by `Time`, and d is the number of model parameters + the number of species . Note that LNA++ computes sensitivities with respect to model parameters and the initial conditions of the `MRE`, but not currently with respect to the initial variances.

check meaning of d

1. `MRE`: a $N \times T$ matrix corresponding to the solution of the macroscopic rate equation for all observed variables.

2. `Sigma`: a $N \times N \times T$ array corresponding to the temporal cross-covariance of all observed species at all time points.

3. `dMRE` (optional): a $N \times T \times d$ array corresponding to the first order sensitivities of the `MRE` to each of the model parameters over all time points. 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): a $N \times N \times T \times d$ array corresponding to the first order sensitivities of the temporal cross-covariance matrix at all timepoints for the observed 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): a $N \times T \times T \times d \times d$ array corresponding to the second order sensitivities of the `MRE` at all timepoints for the observed 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): a $N \times N \times T \times T \times d \times d$ array corresponding to the second order sensitivities of the temporal cross-covariance matrix at all timepoints for the observed 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.

4.3 Python

To use LNA++ with Python, call the function `generateLNA` of the LNA module, with the following arguments:

1. `sbmlFileName`: the path to the SBML model specification file

2. `modelName`: the desired name of the model

3. `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.

Remark: Analytical solution of the steady state equations can only be derived for some models. If the user selects `'Y0'`, `'V0'` or `'BOTH'` and the calculations fails, LNA++ reports an error.

Creating the Python module

After running `generateLNA`, a Python module is created and placed in the `models/modules/` subdirectory. The module is named `modelLNA`, where 'model' is replaced by the name of the model specified. 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` environment variable.

Note for non-standard library or include paths: If CVODES or Blitz++ are not installed in a standard location (e.g. due to lack of administrator permissions), then custom include and/or library paths can be provided by specifying two additional arguments to `generateLNA`:

- `include_dirs`: a Python list of directories to be searched for header files, specified as a list of strings.
- `lib_dirs`: a Python list of directories to be searched for libraries, specified as a list of strings.

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
```

Inputs: 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

$$\begin{bmatrix} \texttt{V\_11}, \texttt{V\_12}, \texttt{V\_13} \\ \texttt{V\_21}, \texttt{V\_22}, \texttt{V\_23} \\ \texttt{V\_31}, \texttt{V\_32}, \texttt{V\_33} \end{bmatrix}$$

   with `V_12 = V_21`, `V_13 = V_31`, and `V_23 = V_32` 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 automatically 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` (optional): a Boolean variable indicating whether the second order sensitivities should be computed. Implies `computeSens=True`. By default no second order sensitivities are computed.

Outputs: The LNA function returns between 2 and 6 output arguments. In the following, N is the number of observed variables (`obsVar`), T is the number of time points specified by `Time`, and d is the number of model parameters + the number of species. Note that LNA++ computes sensitivities with respect to model parameters and the initial conditions of the MRE, but not currently with respect to the initial variances.

1. `MRE`: an $N \times T$ matrix corresponding to the solution of the macroscopic rate equation for all observed variables.

2. `Sigma`: an $N \times N \times T$ array corresponding to the temporal cross-covariance of all observed species at all time points.

3. `dMRE` (if `computeSens`): an $N \times T \times d$ array corresponding to the first order sensitivities of the `MRE` to each of the model parameters over all time points. 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 $N \times N \times T \times T \times d$ array corresponding to the first order sensitivities of the temporal cross-covariance matrix at all timepoints for the observed 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 $N \times T \times d \times d$ array corresponding to the second order sensitivities of the `MRE` at all timepoints for the observed 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 $N \times N \times T \times T \times d \times d$ array corresponding to the second order sensitivities of the temporal cross-covariance matrix at all timepoints for the observed 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 Advanced Notes

5.1 Specifying custom propensity functions

It is possible to manually specify the components necessary for the computation of the linear noise approximation instead of deriving them from a supplied SBML file (see also `examples/runLinearChain.(m|py)`. For both Matlab and Python, the following components need to be specified:

State variables

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

Stoichiometric matrix

This is the N by R matrix of stoichiometric coefficients describing the chemical reaction network, where N is the number of species and R 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 R, 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. By entering the reaction fluxes manually (i.e. not from SBML), custom propensity functions can be specified.

Model parameters

The model parameters are the set of model-specific constants that influence the reaction rates. Sensitivities of the mean and temporal cross-covariance matrices are computed with respect to the model parameters.

Matlab

```
% Define the stoichiometric matrix (S) for the model:
S = [1,-1,0,0; % change to mRNA
     0,0,1,-1]; % change to protein

% Define the state variables and model parameters as symbols:
syms k_m k_p g_m g_p real
syms m p real

modelName = 'BirthDeath';
phi = [m,p]; % state vector
Theta = [k_m, k_p, g_m, g_p]; % parameter vector
```

```
npar = length(Theta); % number of parameters

% Define the reaction fluxes as a function of phi, t and Theta:
fluxFunction = @(phi,t,Theta) [Theta(1),Theta(3)*phi(1),...
                    Theta(2)*phi(1),Theta(4)*phi(2)];

% Generate the symbolic C code:
generateLNAComponents(modelName, S, fluxFunction, phi, Theta, 'BOTH')

% Compile the code and generate the Mex file:
compileLNA(modelName);
```

Python

```
# Setup
from sympy import Matrix
from LNA import *

# Define the stoichiometric matrix (S) for the model:
S = Matrix([[1,-1,0,0],[0,0,-1,1]])

# Define the state variables and model parameters as symbols:
phi = symbols('m,p',real=True)

# Reaction constants
Theta = symbols('k_m,k_p,g_m,g_p',real=True)

# Define the reaction fluxes as a function of phi, t and Theta:
fluxFunction = lambda phi,t,Theta: \
[Theta[0],Theta[2]*phi[0],Theta[1]*phi[0],Theta[3]*phi[1]]

modelName = 'BirthDeath'

# Generate the symbolic C code:
tups = generateLNAComponents(modelName,S,fluxFunction,\
                             species,parameters,computeSS)
npar = len(parameters) # number of parameters

```

Table 1: Reactions for a birth-death process

$\mathcal{R}_1$:	$\emptyset$	$\xrightarrow{k_m}$	mRNA
$\mathcal{R}_2$:	mRNA	$\xrightarrow{\gamma_m}$	$\emptyset$
$\mathcal{R}_3$:	mRNA	$\xrightarrow{k_p}$	mRNA + Protein
$\mathcal{R}_4$:	Protein	$\xrightarrow{\gamma_p}$	$\emptyset$

```
25 # Compile the code and generate the Python module:
26 compileLNA(modelName,S,tups,npar)
```

6 Applications

6.1 Birth-death Process

To test the performance of LNA++, we implemented a simple birth-death process involving a constitutively active DNA, and stochastic production and degradation of mRNA and protein (see Figure 1A, Table 1, `examples/runBirthDeath.(m|py)`). The mRNA production reaction has a constant propensity k_m. The propensity of producing a protein molecule depends on the number of mRNAs, and on the rate constant k_p. Lastly, mRNA is degraded with rate constant γ_m and protein with rate constant γ_p.

Stochastic simulations of the chemical reaction network were generated using the stochastic simulation algorithm [4] implemented in the StockKit 2.0 simulation framework [5]. For simulation, the parameters were set to $k_m = 20$, $k_p = 25$, $\gamma_m = 10$ and γ_p=1.

We performed exact simulation of the stochastic process and used LNA++ to compute the mean and temporal autocovariance of the proteins using the same model parameters. For simplicity, and to replicate a typical fluorescence microscopy setup, only protein abundance is considered to be known. The system size is set to $\Omega = 1$ so that we can identify molecule numbers with molecular concentrations. For this simple system, the LNA agrees well with the exact stochastic simulations in terms of mean and variance (Figure 1B) and the autocovariance (i.e. the cross-covariance of the protein only; Figures 1C and 1D). The autocovariance is shown at three sampled time points ($t = 1.0, t = 2.0, t = 5.0$), showing good qualitative agreement between the empirical covariance derived from 500 simulated SSA trajectories (Figure 1E circles), and from the LNA (Figure 1E lines). In each case, the autocovariance is peaked at zero time lag, and decreases to zero in both directions, i.e. the correlation of fluctuations at a particular time point with future or past fluctuations decreases with increasing absolute time lag.

Check missing subfigures D,E

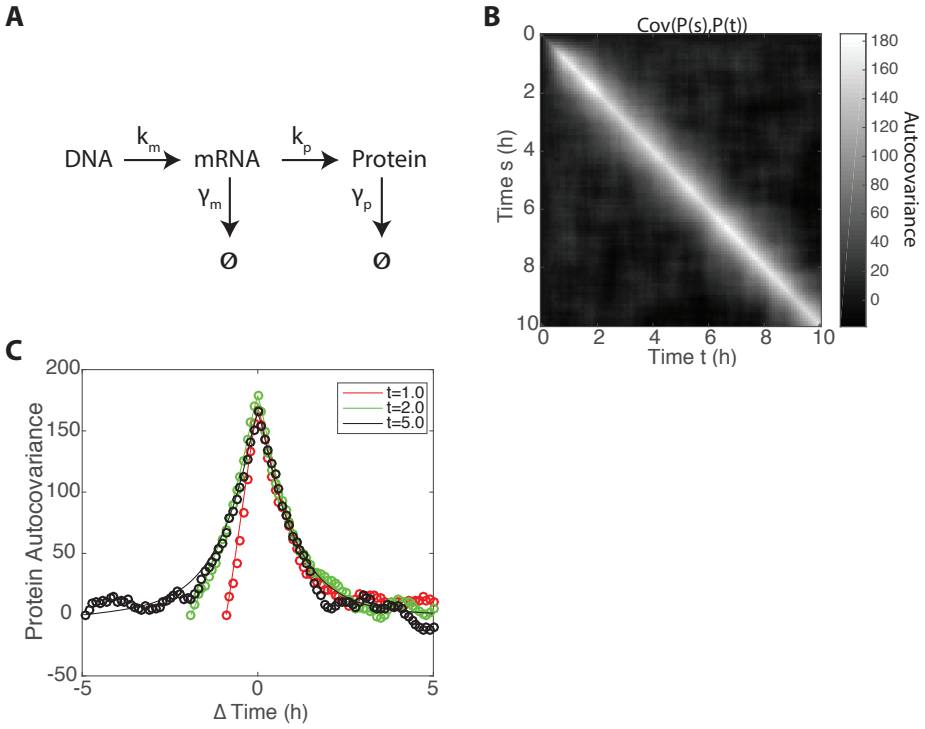

Figure 1: **LNA accurately approximates a simple Birth-death model.** (A) mRNA and proteins are produced and degraded in a simple birth-death model. (B) The protein autocovariance matrix, $\mathrm{cov}(P(s), P(t))$, estimated from empirical data. (C) Comparison of the LNA estimate and the empirical protein autocovariance at three time points. Estimates derived from SSA shown as circles, LNA as lines.

Next, we compute sensitivities of the mean protein copy number with respect to the model parameters using LNA++, and compare the computed results to a finite difference approximation. We find that the sensitivity of the mean agrees exactly with a finite difference approximation for both first and second order sensitivities, see Figures 2 and 3, respectively. The first order sensitivity of the autocovariance also shows excellent agreement between the finite difference approximation and the analytical result from LNA++ (Figure 4). Lastly, the second order sensitivities are difficult to compute robustly using finite differences, for example due to limited accuracy of the ODE solver. While this is generally the case, for this simple model the finite difference approximation still agrees reasonably well with the analytically computed sensitivities from LNA++ (Figure 5).

We performed multi-start optimization in order to compare parameter inference with and

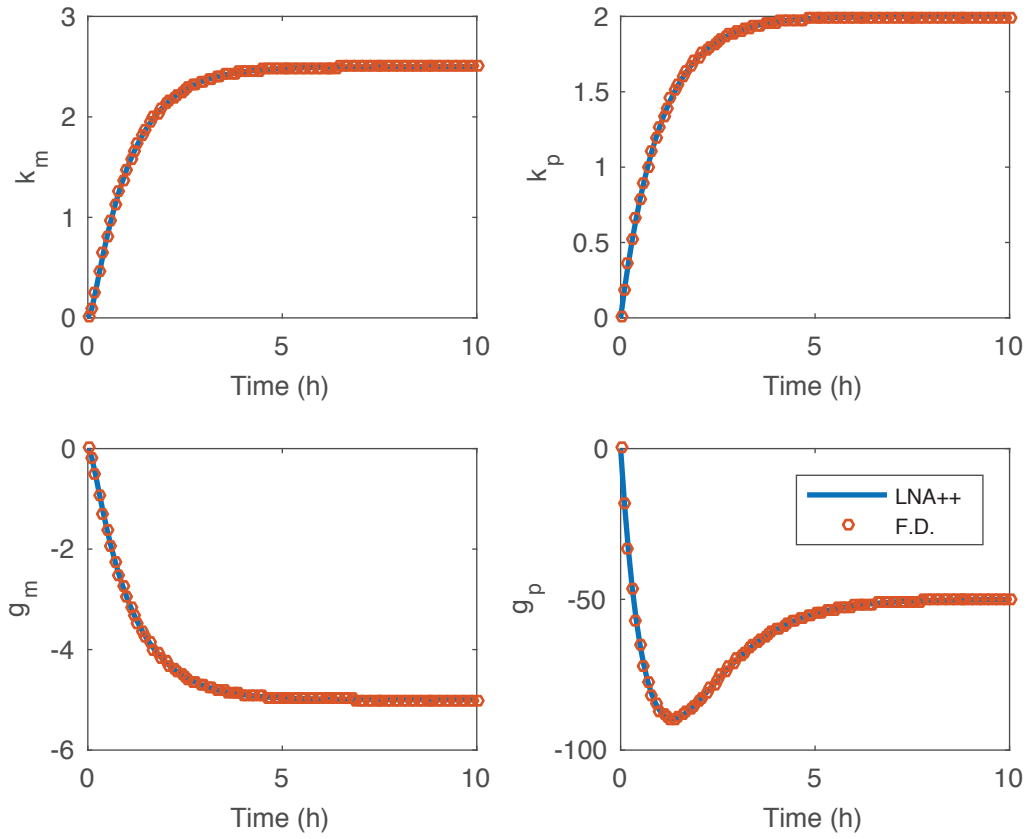

Figure 2: **Comparison of the analytical first order sensitivity of mean protein concentration with respect to parameters ($\frac{\partial \phi}{\partial \theta_i}$) and the finite difference approximation.**

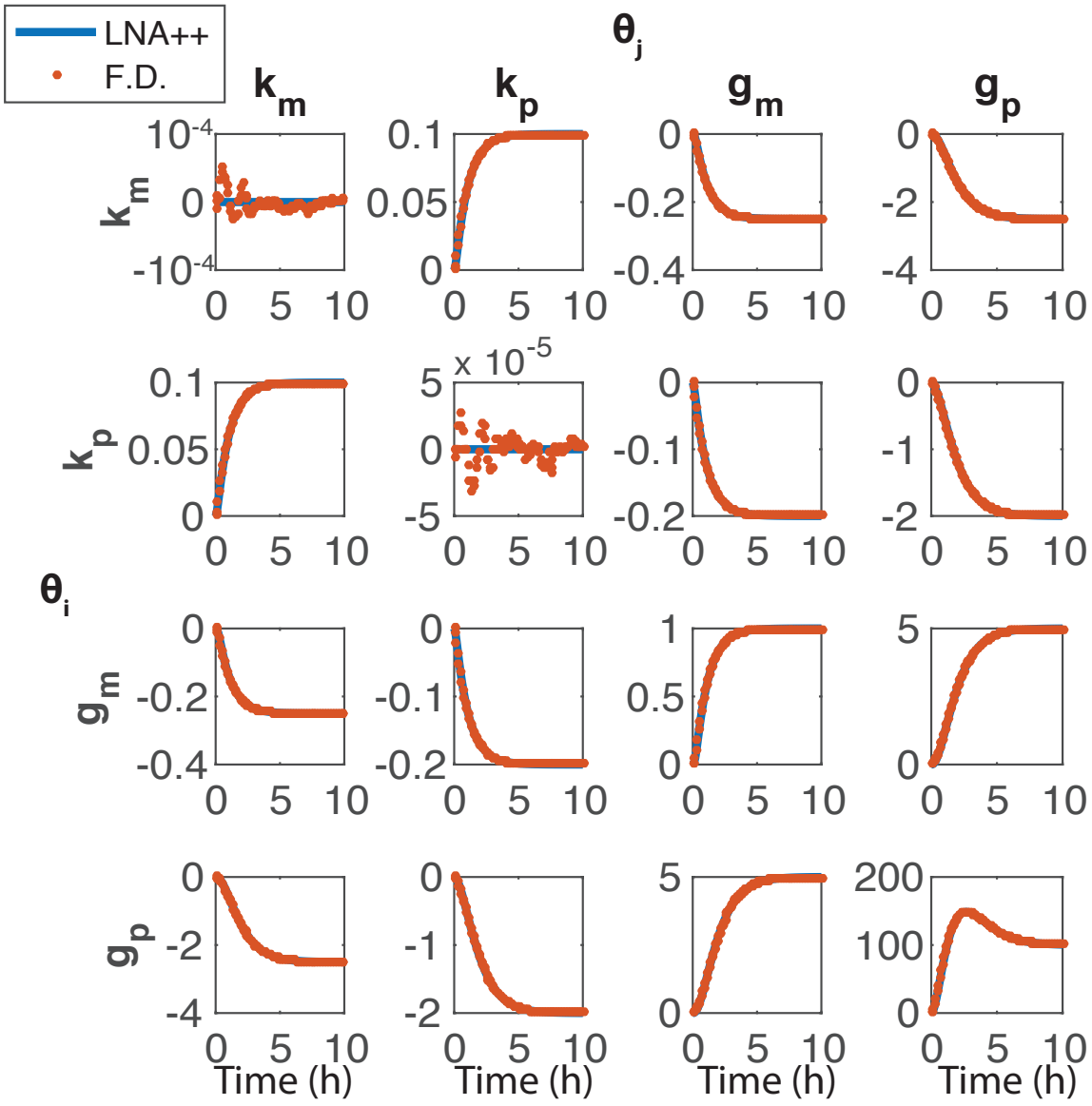

Figure 3: **Comparison of the analytical second order sensitivity of mean protein concentration with respect to parameters ($\frac{\partial^2 \phi}{\partial \theta_i \partial \theta_j}$) and the the finite difference approximation.**

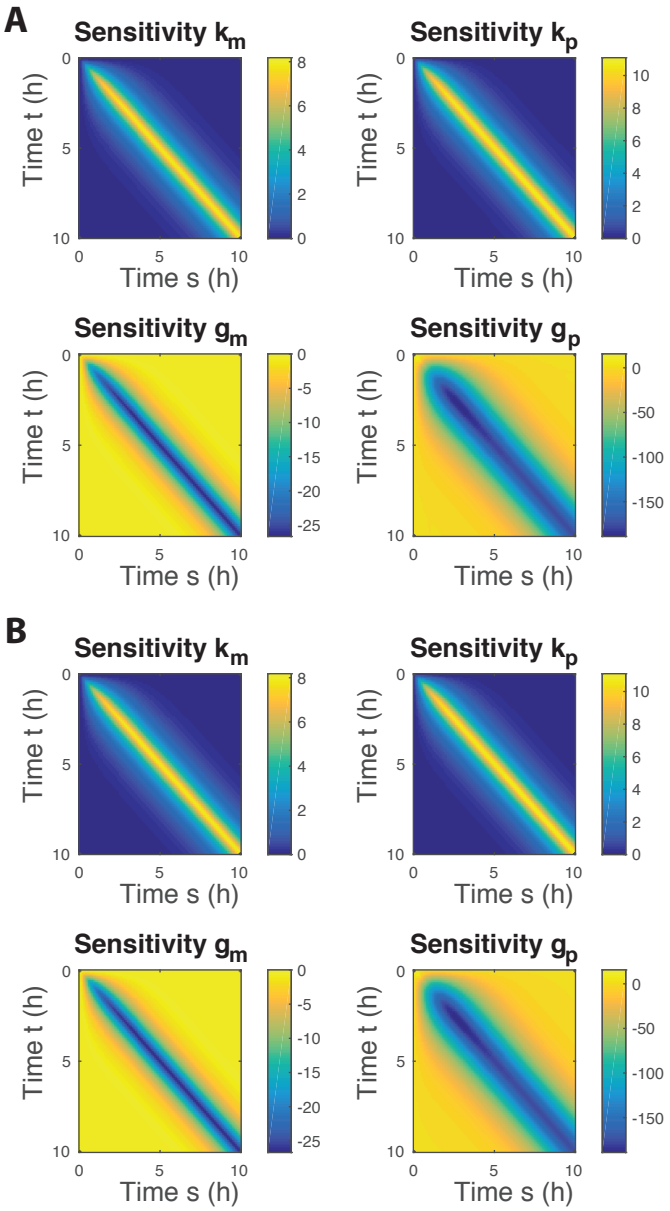

Figure 4: **Comparison of the analytical first order sensitivity of the protein autocovariance with respect to model parameters ($\frac{\partial \Sigma}{\partial \theta_i}$) and the finite difference approximation.** (A) Finite difference approximation. (B) Sensitivity analysis.

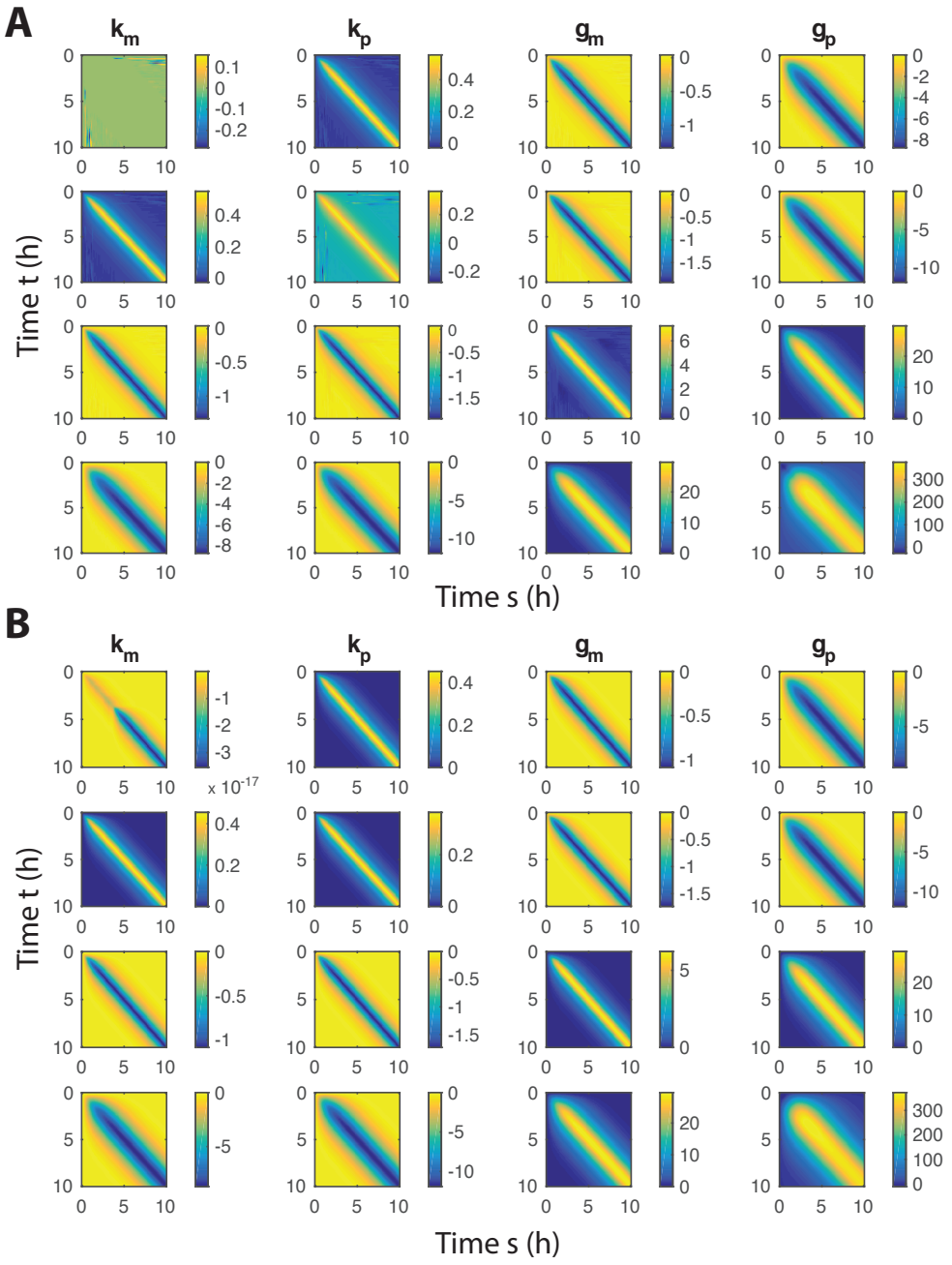

Figure 5: **Comparison of the analytical second order sensitivities of the protein autocovariance with respect to model parameters ($\frac{\partial^2 \Sigma}{\partial \theta_i \partial \theta_j}$) and the finite difference approximation.** (A) Finite difference approximation. (B) Sensitivity analysis.

without the analytical sensitivities. Performing 10 optimizations with randomly (Latin hypercube) chosen initial values for (γ_m, γ_p) in the range $[1, 100] \times [0.01, 10]$ (the correct values of k_m and k_p are assumed to be known), we see that the numerical optimization finds a continuum of local optima before terminating (Figure 6A). These optima have sometimes drastically different log-likelihood values (Figure 6B) and differing values for the locally-optimal model parameters. By contrast, optimization performed using the analytical gradients computed using LNA++ shows excellent convergence to the same global optimum in each of the ten numerical optimizations (Figure 6B). The model parameters identified coincide with the true model parameters used to simulate the data (Figure 6A), unlike the numerical optimization without sensitivities which shows substantial variation. We thus conclude that the first order sensitivities have provided a clear benefit for robust parameter optimization.

Finally, we used LNA++ to evaluate the likelihood landscape for the simple birth-death model. We fix the model parameters k_m and k_p to their correct values, and vary the parameters γ_m and γ_p over a large range encompassing the true parameters used for the stochastic simulations. We find that the for this simple model, the mRNA and protein degradation rates are identifiable, showing a clear optimum at the true model parameters (Figure 6C). Using the second order sensitivities to compute the 95% confidence interval via the Laplace approximation (32), we confirm that the parameters are indeed highly identifiable.

6.2 Linear Chain

To test the scaling performance of the LNA++ code, we implemented a series of models of increasing complexity. Each model consists of a series of linear interconversions between species. The simplest model consists of $n = 3$ species X_1, X_2, and X_3, and the $2n$ reactions:

$$\begin{aligned} \mathcal{R}_1 &: \emptyset \xrightarrow{\alpha_1} X_1 & \mathcal{R}_2 &: X_1 \xrightarrow{\beta_1} \emptyset \\ \mathcal{R}_3 &: X_1 \xrightarrow{\alpha_2} X_2 & \mathcal{R}_4 &: X_2 \xrightarrow{\beta_2} X_1 \\ \mathcal{R}_5 &: X_2 \xrightarrow{\alpha_3} X_3 & R_6 &: X_3 \xrightarrow{\beta_3} X_2 \end{aligned} \tag{33}$$

In the same way we constructed models for $n = 4 \ldots 9$. In each case we assessed the "offline cost", i.e. the time necessary to compute the components of the LNA symbolically (a one time cost), and the "online cost", i.e. the time to evaluate the solution of the LNA (evaluated at each step of the optimization). We find that the offline cost grows with the model size, as many more equations needs to be solved (Figure 7A); offline costs include

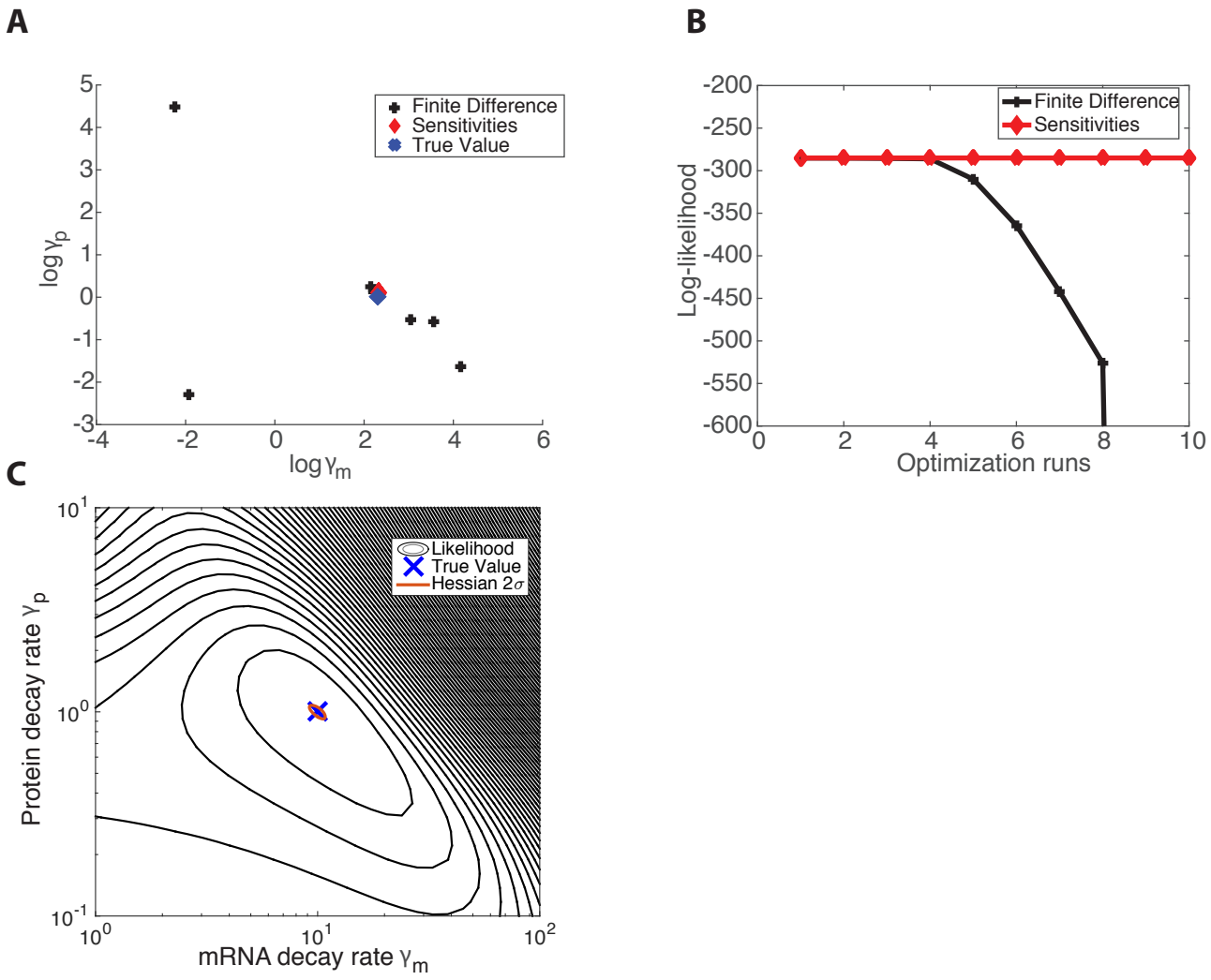

Figure 6: **Analytic sensitivities greatly improve robustness of parameter inference.** (A) The MLE estimated using sensitivities. (B) The log-likelihood of the MLE for multiple optimization runs using sensitivities and a finite difference approximation. Optimization runs are sorted by their log-likelihoods. (C) The log-likelihood landscape of the degradation rates for mRNA and protein, estimated using the LNA, shows a clear optimum for the correct model parameters (contours indicated for $\log[-\ell(\mathbf{\Theta})]$). Using the second order sensitivities, it is possible to estimate the 95% confidence interval for the model parameters (red).

solving for components necessary for first and second order sensitivities. We also estimate the online cost for the LNA solution both with and without sensitivities. The simulations were repeated 10 times each for different parameter sets with each individual parameter chosen randomly from the interval $[0, 100]$, and initial conditions for each species chosen randomly from the interval $[0, 1000]$, see Figure 7B. In each simulation, the same number of time points are used. Simulations were performed on a MacBook Pro laptop with a four core, 2.5GHz Intel i7 processor, and 16GB of RAM. The run time grows quickly, especially when computing the second order sensitivities. However, even with 9 species the run time without sensitivities is still a fraction of a second, and the run time including first order sensitivities approximately 1-2 seconds on average, thus suitable for parameter inference.

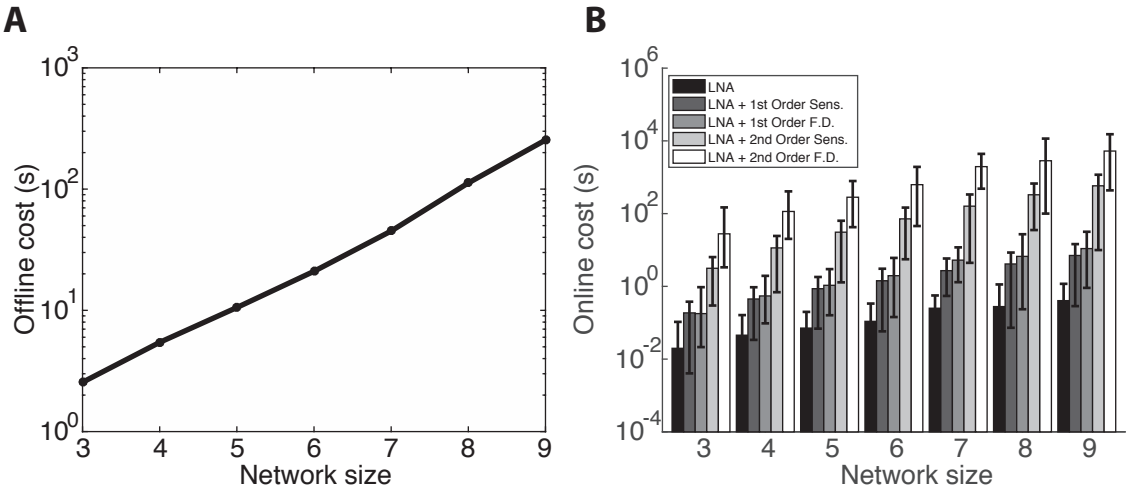

Figure 7: **LNA++ offline and online computation cost versus model size.** (A) The time to generate the Matlab executable ("offline cost"). (B) The median (1st and 3rd quantile) simulation run time ("online cost") with randomly chosen model parameters ($n = 10$). First and second order sensitivities computed using LNA++, and estimated for finite differences using a constant factor of $3n$ and $6n(3n - 1)$ times the online cost of the LNA for network size N, for first order and second order finite differences, respectively.

We compare the run time of the first and second order sensitivities to the estimated run time using finite differences. Approximating the sensitivities using a forward difference scheme requires the evaluation of the function one additional time for each parameter (see Section 2.5.2). In the case of the linear chain model with n species, there are $2n$ reactions rates and n initial conditions (one for each species), leading to an additional $3n$ simulations necessary to approximate all first order sensitivities using finite differences. To

approximate the second order sensitivities, an additional 4 evaluations of the function are necessary for each pair of model parameter/initial conditions (see Section 2.6.2), leading to a total of $4\frac{N(N-1)}{2}$ additional evaluations of f where N is the number of parameters/initial conditions with respect to which to compute the second order sensitivities. As for the first order case $N = 3n$, such that the second order finite difference requires a total of $2N(N-1) = 6n(3n-1)$ evaluations of f in addition to those required by the first order finite difference approximation. Compared to the finite difference computation, the computation of the first order sensitivity computed using LNA++ was between 1.5 and 2.9 times faster, and the computation of the second order sensitivity between 11.4 and 20.5 times faster. Lastly, the sensitivities computed using the analytical expressions do not require *a priori* knowledge of the length scale necessary for a robust finite difference approximation of the gradient, and are thus generally more robust and accurate. Run times for the finite difference approximation are estimated by multiplying the measured run times for the LNA without sensitivities by the factors $3n$ and $6n(3n-1)$ for the first and second order sensitivities, respectively.

In summary, LNA++ facilitates the automatic construction of LNAs and the corresponding forward sensitivity equations for small and medium-sized systems. The forward sensitivity equations provide a numerically more robust and computationally efficient means to calculate sensitivities than the finite difference approximation thus facilitating parameter inference and approximations of their corresponding confidence intervals.

References

[1] N G Van Kampen. Stochastic Processes in Physics and Chemistry, (North-Holland Personal Library). 2007.

[2] Michał Komorowski, Bärbel Finkenstädt, Claire V Harper, and David A Rand. Bayesian inference of biochemical kinetic parameters using the linear noise approximation. *BMC Bioinformatics*, 10(1):343, 2009.

[3] Michał Komorowski, Maria J Costa, David A Rand, and Michael P H Stumpf. Sensitivity, robustness, and identifiability in stochastic chemical kinetics models. *Proceedings of the National Academy of Sciences*, 108(21):8645–8650, 2011.

[4] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25), 1977.

[5] Kevin R Sanft, Sheng Wu, Min Roh, Jin Fu, Rone Kwei Lim, and Linda R Petzold. StochKit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics (Oxford, England)*, 27(17):2457–2458, 2011.