Appraisal of data-driven and mechanistic emulators of nonlinear hydrodynamic urban drainage simulators

Juan Pablo Carbajal[1], João Paulo Leitão[1], Carlo Albert[1], and Jörg Rieckermann[1]

[1]Swiss Federal Institute of Aquatic Science and Technology, Eawag, Überlandstrasse 133, 8600 Dübendorf, Switzerland

February 6, 2017

Abstract

Many model based scientific and engineering methodologies, such as system identification, sensitivity analysis, optimization and control, require a large number of model evaluations. In particular, model based real-time control of urban water infrastructures and online flood alarm systems require fast prediction of the network response at different actuation and/or parameter values. General purpose urban drainage simulators are too slow for this application. Fast surrogate models, so-called emulators, provide a solution to this efficiency demand. Emulators are attractive, because they sacrifice unneeded accuracy in favor of speed. However, they have to be fine-tuned to predict the system behavior satisfactorily. Also, some emulators fail to extrapolate the system behavior beyond the training set. Although, there are many strategies for developing emulators, up until now the selection of the emulation strategy remains subjective. In this paper, we therefore compare the performance of two families of emulators for open channel flows in the context of urban drainage simulators. We compare emulators that explicitly use knowledge of the simulator's equations, i.e. *mechanistic emulators* based on Gaussian Processes, with purely data-driven emulators using *matrix factorization*. Our results suggest that in many urban applications, naive data-driven emulation outperforms mechanistic emulation. Nevertheless, we discuss scenarios in which we think that mechanistic emulation might be favorable for i) extrapolation in time and ii) dealing with sparse and unevenly sampled data. We also provide many references to advances in the field of Machine Learning that have not yet permeated into the Bayesian environmental science community.

1   Introduction

For many real-world systems with a nonlinear response, model based tasks such as sensitivity analysis, learning model parameters from data (i.e. system identification or model calibration), and real-time control, are hampered by the long runtime of the employed numerical simulators. Even if runtimes

1

are short, these methods require a large number of model runs, which can take a prohibitive long time. One way of speeding up these tasks is to build fast surrogate models, so called *emulators*, to replace the computationally expensive simulators. An emulator is a numerical model that is tailored to approximate the results of a computationally expensive simulator with a huge reduction in the time needed to run a simulation [O'Hagan, 2006], i.e. it is a metamodel. These ideas also belong to the technique of Reduced-Order Models (ROM), specially for models based on Partial Differential Equations (PDE) [Baur et al., 2014, Quarteroni et al., 2016], and emulation as described below.

To ground ideas, imagine that the flow at the outlet of a drainage network is limited using a flow limiting gate or by activating water storage systems (Fig. 1). The position of the gate and the activation of storage is controlled using a model predictive controller [Xi et al., 2013]. Such a scenario is relevant in performance optimization of water treatment plants [Fu et al., 2008]. The signals used to control the flows could be the current intensity and duration of rain events from several rain gauges within the catchment. The controller needs to estimate an optimal course of action by predicting the flows induced by the rain and many possible actuations. This optimization generally requires thousands of model runs, which can take a prohibitive long time when running a physically detailed simulator of the sewer network, such as a EPA Storm Water Management Model (SWMM) model [Rossman, 2010]. However, the simulator is just used to estimate the relation between the rain, the actuation, and the flow. The full details of the simulator might not be required to obtain an accurate estimation of this relation. Feedback control might further reduce the required accuracy of the estimated relation.

Emulation and interpolation are equivalent problems. An emulator is built using the best available simulator to sample the space of actuations and/or parameters (henceforth the latter will include actuations). The training data is then used to build an interpolation function which should predict values at unseen parameters with an acceptable degree of accuracy, which is case dependent. That is, we reconstruct an unknown function $F : \mathbb{R}^{|\gamma|+1} \to \mathbb{R}$, that takes a parameter vector of size $|\gamma|$ and a time instant, and generates the value of the magnitude of interest. When this function is evaluated at the inputs used for training, the results are the same as the training outputs[1], i.e. this is the meaning of interpolation of the training data adopted herein. Stated in this way, no distinction is needed between the parameters and the time components in the input. However, knowing that the data is generated by a dynamical system, we separate time from the other parameter components. Thus, we can find one interpolant in time and one in parameter space, which might be coupled to each other. This parameter-time coupling emerges naturally in mechanistic emulation, as will be shown in Sec. 2.2.

When the simulator is based on differential equations, the link between Gaussian Processes (GP) and linear stochastic differential equations (SDE) [Poggio and Girosi, 1990, Steinke and Schölkopf, 2008, Albert, 2012, Särkkä et al., 2013, González et al., 2014, Solin, 2014], permits the creation of GP based emulators that include knowledge about the simulator dynamics; these are called *mechanistic emulators* (MEMs). Conceptually, mechanistic emulation seeks a function that interpolates the training data whithin a class of functions defined by an SDE. The importance of GPs for MEMs stems from the fact that they are the formal solution of this SDE. Hence when the simulator is linear the emulator gives exact results; while for nonlinear simula-

---

[1]Herein considered noiseless since they are generated by a deterministic simulator.

Figure 1: An imaginary drainage network with flow limiting gate and/or water storage systems. The interesting singal is the level/flow at the outlet of the network, marked with a circle. Isometric tiles from www.kenney.nl.

tors, the MEM will provide only an approximation. Increasing the accuracy of this approximation and the efficiency of the methods are two fundamental challenges in GP based emulation [O'Hagan, 2006, Rasmussen and Williams, 2006, Ch. 8].

Reichert et al. [2011] enumerated four overlapping approaches for developing emulators of dynamic simulators:

i) Gaussian Processes

ii) Basis function decompositions

iii) State space transition function approximation

iv) Stochastic linear model conditioned on data using Kalman smoothing.

In particular approaches i) and iv) are two different implementations of the same problem [Steinke and Schölkopf, 2008]. Roughly speaking the Kalman smoothing algorithm used in iv) is an iterative implementation of the conditioning of the GP in i). The iteration in iv) avoids the ill-conditioned covariance matrices [Hansen, 1998] involved in GP when sampling rates are high [Steinke and Schölkopf, 2008, Reichert et al., 2011] and it is faster than direct matrix inversion in a serial implementation. The GP approach i) is better suited for parallelization, speedups and energy saving via approximated computing [Angerer et al., 2015].

Approaches i) (or iv)) and ii) are similar with respect to their implementation. That is, approach ii) can be implemented using GP regression [Rasmussen and Williams, 2006, sec. 2.7]. Therefore, the essential difference between i) and ii) is that the former explicitly introduces mechanistic knowledge. It will be shown here that approach i) is currently constrained to linear mechanistic knowledge, while popular methods based on maximum entropy [Victor and Johannesma, 1986, Christakos, 1998, Harte, 2011] can handle nonlinear knowledge. The difference between GP based mechanistic emulation and maximum entropy methods is that in the latter, the mechanistic knowledge is added as constraints on the moments of the predictive or posterior distribution, while in the former its is added as the dynamics of the prior model. Adding constraints to predictive distributions require expert knowledge available at the level of the emerging behavior of the simulator, while adding dynamic information requires knowledge about the constitutive elements parts of the simulator. The latter is likely to be readily available from the development of the simulator itself.

Herein we compare the performance of GP emulators built using approaches i), which we call *mechanistic emulation*, and ii) which we call *data-driven emulation*. The basis function that will be used for data-driven emulation will be derived solely from the data using matrix factorization, i.e. they will not explicitly include mechanistic knowledge. In this article we use singular value decomposition (SVD) and nonnegative matrix factorization (NMF) to extract these bases (Sec. 2.1), but more general basis extraction methods like Proper Orthogonal Decomposition (POD) could be used [Hesthaven et al., 2016]. We compare results in an academic emulation problem to highlight the differences between the approaches (Sec. 3.1-3.2). We also provide emulation examples pertinent to the fields of hydrology and urban water management (Sec. 3.3-3.4). In all of these, data-driven emulation outperforms mechanistic emulation. The objective of this comparison is to provide intuition about the suitability of each approach, which is not available to date to the best of our knowledge, to highlight the need of enhancements of our emulators, and to motivate research questions in the field of emulation. This is relevant, because, as described above, many applications in the field of urban drainage

and flood predictions to date are hampered by slow models. To a lesser degree, this is one of the few NMF applications in hydrology [Alexandrov and Vesselinov, 2014].

2 Methods and Materials

In the subsequent sections we firstly describe the two emulation approaches used herein (Sec. 2.1-2.2). Aiming at a wide readership, mathematical detail and rigor are kept at a minimum required level. References are provided for the interested reader. Secondly, we describe the datasets used for training and testing the emulators (Sec. 2.3). These include models of two small catchments in Switzerland, that we use to thoroughly evaluate the performance of the emulators.

We use the word *data* to refer to the input and output pairs provided by the simulator being emulated, e.g. in the case of a hydrodynamic simulator, inputs could be time and physical parameters of a sewer network, and outputs water levels or flows.

2.1 Data-driven emulators

In this approach we make the following assumptions about the simulation data used to build the emulator:

a) The data contains the most significant dynamic features of the system response and these can be used as a *time varying basis* to reproduce the data.

b) Unseen system responses are well approximated by a linear combination of the features in a), i.e. there exist a solution manifold that can be well approximated with low dimensional sets [Quarteroni et al., 2016, Hesthaven et al., 2016].

c) There exists a "smooth" mapping between inputs and the coefficients of the linear combinations of features.

With these assumptions in mind we define the approximation strategy:

$$y(t, \boldsymbol{\gamma}) \simeq \sum_{i=1}^{q} \beta_i(\boldsymbol{\gamma}) \phi_i(t), \tag{1}$$

$$\beta_i(\boldsymbol{\gamma}) \sim \mathcal{GP}\left(\mathfrak{m}_i(\boldsymbol{\gamma}), \mathfrak{K}_i(\boldsymbol{\gamma}, \boldsymbol{\gamma}')\right) \tag{2}$$

where $y(t, \boldsymbol{\gamma})$ is the output of the simulator at time t and parameters $\boldsymbol{\gamma}$, β_i is a mapping between these parameters and the components of the output in the basis function set $\{\phi_i(t)\}$. Following Rasmussen and Williams [2006, Sec. 2.7] this could be generalized to a full GP regression problem. However, as stated here the problem is simpler and it is justified by the performance it provides in the examples showcased in Sec. 3.

The procedure to build a data-driven emulator follows:

i. Extract the first $q \leq n_{\mathrm{trn}}$ features $\{\phi_i(t)\}$ using the training set $\{y_i(t, \boldsymbol{\gamma}_i)\}_{i=1}^{n_{\mathrm{trn}}}$. This gives a $q \times n_{\mathrm{trn}}$ matrix $\boldsymbol{B}$ of coefficients (q coefficients for each simulation used for training) and the basis evaluated at the observed time points $\Phi_{ij} = \phi_j(t_i)$, $i = 1, \ldots, N$, $j = 1, \ldots, q$.

ii. Interpolate the $\boldsymbol{B}$ coefficients from step i. using a GP to obtain a function of the inputs $\boldsymbol{B} = f(\boldsymbol{\gamma})$.

iii. Evaluate $f(\boldsymbol{\gamma})$ on the parameters in test set, use eq. (1) to predict outputs, and compare them with the output test set $\{y_i(t, \boldsymbol{\gamma}_i)\}_{i=1}^{n_{\text{tst}}}$.

The features $\{\phi_i(t)\}$ are extracted from the training data itself via matrix factorization, which allows us to impose some general constraints on the features, e.g. nonnegativity. Herein we use the singular value decomposition (SVD) and nonnegative matrix factorization (NMF), which are explained in later pragraphs.

These methods provide the basis evaluated only at the observed time points, Φ, and to predict at unobserved times we linearly interpolate them over time. The implications of this interpolation will be discussed in Sec. 4. This approach decouples the interpolation in time with the interpolation in parameter space.

SVD is a robust factorization to calculate the eigenvectors of the covariance of the data matrix, i.e. the principal components. Given the matrix $\boldsymbol{Y} \in \mathbb{R}^{N \times n}$, SVD calculates the orthogonal matrices $\Phi \in \mathbb{R}^{N \times n}$ and $\boldsymbol{W} \in \mathbb{R}^{n \times n}$, and the element-wise nonnegative diagonal matrix $\Sigma \in \mathbb{R}_+^{n \times n}$ with only $\Sigma_{ii} \neq 0$. These matrices fulfill the relation

$$\boldsymbol{Y} = \Phi \Sigma \boldsymbol{W}^\top. \tag{3}$$

Using this factorization to calculate the covariance of the data matrix gives

$$\boldsymbol{Y}\boldsymbol{Y}^\top = \Phi \Sigma \Sigma^\top \Phi^\top. \tag{4}$$

Showing that Φ are eigenvectors of the covariance matrix, i.e the principal components of the data. A decomposition of $\boldsymbol{Y}$ in the form of eq. (1) is obtained by defining $\boldsymbol{B}$ as the follows:

$$\boldsymbol{B} = \Sigma \boldsymbol{W}^\top, \tag{5}$$

$$\boldsymbol{Y} = \Phi \boldsymbol{B}. \tag{6}$$

The quality of approximation of the data degrades graciously with decreasing number of principal components. Hence, only the first $q < n_{\text{trn}}$ principal components are used in general. This reduces the size of the representation of the training data. In the context of ROM and PDE, this decomposion is also know as Proper Orthogonal Decomposition (POD).

NMF provides an approximate minimum norm positive decomposition of the data [Kim and Park, 2008][2]. Formally it solves the following problem:

Problem 1 (NMF). *Given the matrix $\boldsymbol{Y} \in \mathbb{R}^{N \times n}$ and $q \in \mathbb{N}$ with $q \leq n$, find matrices $\Phi \in \mathbb{R}_+^{N \times q}$ and $\boldsymbol{B} \in \mathbb{R}_+^{q \times n}$ such that they minimize*

$$\|\boldsymbol{Y} - \Phi \boldsymbol{B}\|^2 + a\|\Phi\|^2 + b\|\boldsymbol{B}\|^2. \tag{7}$$

Where $\|\cdot\|$ is the Frobenius matrix norm and $\mathbb{R}_+$ is the set of nonnegative real numbers.

Intuitively, NMF works as SVD but constraining the principal components and the mixing coefficients to be nonnegative. The decomposition controls the norm of the basis and its coefficients using two regularization terms parametrized with weights a and b, which are manually tuned. The nonnegative basis provided by NMF might be readily interpreted in physical terms in hydrological applications, yet the method is not widespread in the community.

---

[2]function nmf_bpas of GNU Octave's linear-algebra package http://octave.sourceforge.net/linear-algebra/.

2.2 GP based mechanistic emulator (MEM)

The predictive mean of a GP in $\boldsymbol{x}$ conditioned on data observed at $\boldsymbol{x}'$, has the following structure

$$y(\boldsymbol{x}) = \mathfrak{K}(\boldsymbol{x},\boldsymbol{x}')\boldsymbol{\alpha} + \mathfrak{m}(\boldsymbol{x}), \qquad (8)$$

$$\boldsymbol{\alpha} = \left(\mathfrak{K}(\boldsymbol{x}',\boldsymbol{x}') + \kappa I\right)^{-1}\left(y(\boldsymbol{x}') - \mathfrak{m}(\boldsymbol{x}')\right). \qquad (9)$$

where $\mathfrak{K}$ and $\mathfrak{m}$ stand for the covariance and the mean function of the prior GP, respectively. The weights vector $\boldsymbol{\alpha}$ is learned from the data at the conditioning step, requiring the inversion of the matrix obtained by evaluating the covariance function at the observed inputs, shown in eq. (9). The regularization parameter κ encodes our trust on the prior knowledge and the error, if any, of the observations $y(\boldsymbol{x}')$[3].

A MEM is the GP in time and simulator parameters associated with an input-driven linear stochastic ordinary differential equation (SODE) with a multidimensional state space. Each component of the state space is defined by an observed simulator's parameter vector ($\boldsymbol{\gamma}_i$). Extra components are reserved for prediction at unseen simulator's parameter vectors[4].

The process of building a MEM requires the definition of

  i. A linear prior model.

  ii. The covariance and mean functions of the GP.

  iii. A mapping from the parameters of the simulator to the parameters of the GP.

In the following sections we obtain the mathematical expression of the covariance and mean functions of a first order linear time invariant (LTI) SODE, and explain the construction of the mapping between parameter spaces. LTI systems often arise, for example, from a finite element modeling of partial differential equations.

For a rigorous and more general development, that include time varying parameters, see Albert [2012]. The development for periodic difference equations was presented in Steinke and Schölkopf [2008] and extended to a more general setting in González et al. [2014].

A LTI SODE in the vector-valued function

$$\boldsymbol{s}(t) : \mathbb{R}^{+} \to \mathbb{R}^{M}, \qquad (10)$$

with initial condition $\boldsymbol{s}(0) = \boldsymbol{s}_0 \sim \mathcal{N}(\bar{\boldsymbol{s}}_0, \Sigma_0)$, is defined by the equation

$$\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t}(t) = \mathcal{A}\boldsymbol{s}(t) + \boldsymbol{u}(t) + \boldsymbol{\xi}(t), \qquad (11)$$

where $\mathcal{A} \in \mathbb{R}^{M \times M}$, $\boldsymbol{u}(t)$ is a deterministic exogenous actuation, and $\boldsymbol{\xi}(t) \sim \mathcal{N}\left(0, \Sigma_\xi(t,t')\right)$ is a Gaussian noise term with covariance function

$$\Sigma_\xi(t,t') = \Sigma\delta(t - t') \quad \Sigma \in \mathbb{R}^{M \times M}. \qquad (12)$$

The general solution of this equation is

$$\boldsymbol{s}(t) = e^{\mathcal{A}t}\boldsymbol{s}_0 + \int_0^t e^{\mathcal{A}(t-\tau)}(\boldsymbol{u}(\tau) + \boldsymbol{\xi}(\tau))\,\mathrm{d}\tau. \qquad (13)$$

---

[3]In the MEMs used herein, to obtain interpolation of the training data, we set the regularization parameter to the machine epsilon.

[4]typically only one component is reserved for this, but more could be used in a parallel setting.

where the first term is the solution of the homogenous ODE, i.e. with $\boldsymbol{u}(t)$ and $\boldsymbol{\xi}$ both zero. The second term is the response of the ODE to the noisy actuation.

As mentioned above, this system depends on the simulator's parameters. In general this means $\mathcal{A}(\boldsymbol{\gamma})$, $\Sigma_\xi(\boldsymbol{\gamma},\boldsymbol{\gamma}')$, $\boldsymbol{u}(t,\boldsymbol{\gamma})$, and potentially $\boldsymbol{s}_0(\boldsymbol{\gamma})$.

2.2.1 Covariance function

We calculate the covariance function of the trajectories in eq. (13) using the formula

$$\mathfrak{K}(t,r) = \mathbb{E}\left[(\boldsymbol{s}(t) - \mathbb{E}[\boldsymbol{s}(t)])(\boldsymbol{s}(r) - \mathbb{E}[\boldsymbol{s}(r)])^\top\right], \tag{14}$$

where $\boldsymbol{s}^\top$ indicates transposition and $\mathbb{E}$ is the ensemble average over initial conditions and noise realizations.

The ensemble average of the solutions, $\mathbb{E}[\boldsymbol{s}(t)]$, is solely determined by the average of the homogeneous part (depending only on the initial condition) and the deterministic actuation $\boldsymbol{u}$, because the contribution of the noise term $\boldsymbol{\xi}$ vanishes due to its zero mean value:

$$\mathbb{E}[\boldsymbol{s}(t)] = e^{\mathcal{A}t}\bar{\boldsymbol{s}}_0 + \int_0^t e^{\mathcal{A}(t-\tau)}\boldsymbol{u}(\tau)\mathrm{d}\tau. \tag{15}$$

That is, the mean function is the solution to the deterministic (noise-free) inhomogeneous ODE.

From eqs. (13) and (15) we obtain the factors in the expectation in eq. (14),

$$\boldsymbol{s}(t) - \mathbb{E}[\boldsymbol{s}(t)] = e^{\mathcal{A}t}(\boldsymbol{s} - \bar{\boldsymbol{s}}_0) + \int_0^t e^{\mathcal{A}(t-\tau)}\boldsymbol{\xi}(\tau)\mathrm{d}\tau. \tag{16}$$

Inserting this in the expression for the covariance function, eq. (14), we obtain four terms. The first term is the covariance of the initial conditions. The second term is a product of the integral of the noise term. The last two terms are products of the initial conditions and the noise term; these terms will vanish due to the independence of the initial condition and the noise term, and due to the zero mean of the latter. Finally, we obtain:

$$\begin{aligned}\mathfrak{K}(t,r) = e^{\mathcal{A}t}\Sigma_0 e^{\mathcal{A}^\top r} + \\ \int_0^t \int_0^r e^{\mathcal{A}(t-\tau)}\Sigma\delta(\tau-\rho)e^{\mathcal{A}^\top(r-\rho)}\mathrm{d}\rho\mathrm{d}\tau = \\ e^{\mathcal{A}t}\Sigma_0 e^{\mathcal{A}^\top r} + \int_0^{\min(t,r)} e^{\mathcal{A}(t-\mu)}\Sigma e^{\mathcal{A}^\top(r-\mu)}\mathrm{d}\mu.\end{aligned} \tag{17}$$

The second term can be recognized as the property of the covariance under a linear transformation:

$$\boldsymbol{s} = \mathrm{G}\boldsymbol{v} \quad \Sigma_{\boldsymbol{s}} = \mathrm{G}(\mathrm{G}\Sigma_{\boldsymbol{v}})^\top, \tag{18}$$

where the matrix product should be interpreted as the application of the integral

$$(\boldsymbol{A}\boldsymbol{v})(t) = \int_0^\infty \boldsymbol{A}(t,\mu)\boldsymbol{v}(\mu)\mathrm{d}\mu, \tag{19}$$

$$(\boldsymbol{A}\boldsymbol{B})(t,r) = \int_0^\infty \boldsymbol{A}(t,\mu)\boldsymbol{B}(\mu,r)\mathrm{d}\mu. \tag{20}$$

In our case G stands for the Green's function of the linear operator

$$\frac{\mathrm{d}}{\mathrm{d}t} - \mathcal{A}, \tag{21}$$

8

which is

$$\mathrm{G}(t,r) = \mathcal{H}(t-r)e^{\mathcal{A}(t-r)}, \tag{22}$$

with $\mathcal{H}$ the Heaviside or Step function. Its transpose (adjoint) is

$$\mathrm{G}^{\dagger}(t,r) = \mathcal{H}(r-t)e^{\mathcal{A}^{\top}(r-t)}. \tag{23}$$

giving

$$
\begin{aligned}
\left(\boldsymbol{G}\left(\boldsymbol{G}\Sigma_{\boldsymbol{\xi}}\right)^{\dagger}\right)(t,r) &= \\
\int_0^{\infty} \mathrm{G}(t,\mu)\left(\int_0^{\infty} \mathrm{G}(\mu,\tau)\Sigma\delta(\tau-r)\mathrm{d}\tau\right)^{\dagger}\mathrm{d}\mu &= \\
\int_0^{\infty} \mathrm{G}(t,\mu)\Sigma\,\mathrm{G}^{\dagger}(\mu,r)\mathrm{d}\mu &= \\
\int_0^{\infty} \mathcal{H}(t-\mu)\mathcal{H}(r-\mu)e^{\mathcal{A}(t-\mu)}\Sigma e^{\mathcal{A}^{\top}(r-\mu)}\mathrm{d}\mu &= \\
\int_0^{\min(t,r)} e^{\mathcal{A}(t-\mu)}\Sigma e^{\mathcal{A}^{\top}(r-\mu)}\mathrm{d}\mu &
\end{aligned}
\tag{24}
$$

This implies that if the differential operator has a known Green's function, then the covariance function of the GP can be calculated by transforming the covariance function Σ of the noisy input $\boldsymbol{\xi}$. A more general and formal treatment of this process is described in Kimeldorf and Wahba [1970] and the relation between differential operators and kernels is summarized in Steinke and Schölkopf [2008]. This covariance function computes the statistical interactions between the components of the LTI SODE. In other words, it provides the coupling of components of what is know as multi-output GP in the machine learning community and cokriging in geostatistics [Rasmussen and Williams, 2006, Sec. 9.1]. In Fricker et al. [2013] different structures of the coupling between output components were studied, MEMs automatically derive the coupling structure from the available prior knowledge.

2.2.2 Linear prior

To determine the elements in the system matrix $\mathcal{A}$ in (11), we select a linear model for each output in the training data corresponding to a simulator's parameter vector, which we call the *linear proxy*:

$$\mathcal{L}_i : \mathbb{R}_+ \times \mathbb{R}^q \to \mathbb{R}^d, \tag{25}$$

$$t,\boldsymbol{\theta}_i \mapsto \boldsymbol{z}(t,\boldsymbol{\theta}_i). \tag{26}$$

The q dimensional parameter vector $\boldsymbol{\theta}_i$ of the proxy depends on the simulator's parameters used for the ith simulation, i.e. $\boldsymbol{\theta}_i(\boldsymbol{\gamma}_i)$. The proxy's output dimension d is equal to dimension of each output in the training data, usually $d = 1$. The concatenated set of proxies defines the *linear prior* of the MEM.

Herein, the proxies will be m dimensional linear time invariant (LTI) ODEs, i.e. in state space form

$$\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t}(t) = \boldsymbol{A}(\boldsymbol{\theta})\boldsymbol{s}(t) + \boldsymbol{u}(t,\boldsymbol{\theta}) \quad \boldsymbol{A} \in \mathbb{R}^{m \times m}, \boldsymbol{u} \in \mathbb{R}^m \tag{27}$$

$$\boldsymbol{z}(t) = \boldsymbol{C}\boldsymbol{s}(t) \quad\quad\quad \boldsymbol{C} \in \mathbb{R}^{d \times m} \tag{28}$$

Here all proxies have the same dimension m. Although this is not required by the method, it is the simplest structure of MEMs [Albert, 2012]. Nevertheless, proxies with different dimensions might be better suited for dynamical systems with bifurcations, e.g. training data containing a mixture of

oscillating and converging time series, due to the presence of a Supercritical Andronov-Hopf bifurcation [Kuznetsov, 2006] in the simulator.

The emulator's linear prior is constructed by aggregating the proxies and by coupling them with a noise term with a covariance that depends on the simulator's parameters, i.e

$$\frac{\mathrm{d}S}{\mathrm{d}t}(t) = \mathcal{A}(\Theta)S(t) + U(t,\Theta) + \boldsymbol{\xi}(t,\Gamma), \tag{29}$$

$$Z(t) = \mathcal{C}(\Theta)S(t), \tag{30}$$

$$\mathcal{A} \in \mathbb{R}^{mn \times mn}, U, \xi \in \mathbb{R}^{mn \times 1} \mathcal{C} \in \mathbb{R}^{d \times mn}.$$

Where $\Theta = \{\boldsymbol{\theta}_i\}$ and $\Gamma = \{\boldsymbol{\gamma}_i\}$ contain the corresponding n parameter vectors. The matrix $\mathcal{A}$ is block diagonal, $\mathcal{C}$ is a horizontal concatenation, and U a vertical concatenation:

$$\mathcal{A}(\Theta) = \begin{bmatrix} \boldsymbol{A}(\boldsymbol{\theta}_1) & & \\ & \ddots & \\ & & \boldsymbol{A}(\boldsymbol{\theta}_n) \end{bmatrix},$$

$$U(t,\Theta) = \begin{bmatrix} \boldsymbol{u}(t,\boldsymbol{\theta}_1) \\ \vdots \\ \boldsymbol{u}(t,\boldsymbol{\theta}_n) \end{bmatrix}, \tag{31}$$

$$\mathcal{C}(\Theta) = \begin{bmatrix} \boldsymbol{C}(\boldsymbol{\theta}_1) & \dots & \boldsymbol{C}(\boldsymbol{\theta}_n) \end{bmatrix}.$$

2.2.3 Parameter mapping

To evaluate the matrices in eq. (31) we need a mapping from simulator's parameters $\boldsymbol{\gamma}$ to proxy's parameters $\boldsymbol{\theta}$, , i.e. $\Gamma \mapsto \Theta$. It can be an ad-hoc function derived from knowledge about the simulator, as was done in [Albert, 2012, Machac et al., 2016a,b] or it can be learned directly from the data. The latter is especially useful if proxies with different dimensions are combined to form the linear prior of the emulator.

A proxy's parameter $\boldsymbol{\theta}_i$ can be learned from the data via optimization, e.g. least squares fit of the data $y_i(T,\boldsymbol{\gamma}_i)$ using $\boldsymbol{z}(T,\boldsymbol{\theta}_i)$. Alternatively, Θ can be left as hyperparameters in the GP and estimated by maximizing the likelihood. In any case, the obtained pairs $(\boldsymbol{\gamma}_i, \boldsymbol{\theta}_i)$ are then used to learn a mapping between the simulator's and emulator's parameter spaces.

2.2.4 Warped MEM

It is possible to add some nonlinear knowledge to a mechanistic emulator when the simulator's equations can be approximated with a Wiener model, i.e. a time independent nonlinear function applied to the states of a linear dynamical system. To do this we use Warped GPs [Snelson et al., 2003]. This modification does not affect the structure of the emulator, only the data on which it is trained. In addition to the linear dynamical system parameters defining the prior, the parameters of the nonlinear function need to be learned as well unless this mapping is explicitly given, e.g. from the linearization of a nonlinear ODE via a change of variables as in the Bernoulli ODE [Hairer et al., 1993].

2.3 Datasets description

Simulated data used to build emulators are provided in a set of scalar signals $y_i(t_j, \gamma_i)$ with $i = 1, \ldots, n$, all of them sampled at the same time points, i.e. $T = \{t_j\}$, $j = 1, \ldots, N$. For all emulators reported here, the time series in the datasets were subsampled as much as possible, without deteriorating their relevant dynamic features, e.g. oscillations and/or peaks. In all cases, the datasets are randomly separated in a training set of size n_{trn} and a test set of size n_{tst}, with $n_{\mathrm{trn}} + n_{\mathrm{tst}} = n$. Table 1 summarizes the properties of the data used.

Dataset	Nonlinear DS I & II	Wartegg	Adliswil
# parameters, $|\gamma|$	1	2	8
Time samples, N(used/total)	6/40	52/2880	193/601
Training set, n_{trn}	10	200	128
Test set, n_{tst}	190	700	128

Table 1: Summary of datasets used herein.

2.3.1 Nonlinear dynamical system dataset I & II

To illustrate the virtues of MEMs, we first consider a didactical example using data generated by the model

$$\frac{\mathrm{d}x}{\mathrm{d}t} = a(x_0)x + b(x_0), \quad x(0) = x_0, \tag{32}$$

$$a(x) = -a_0 e^{-a_1(x - \mathrm{sign}(x))^2}, \tag{33}$$

$$b(x) = b_0 \tanh(x). \tag{34}$$

The parameters values are $a_0 = 12.616, a_1 = 5, b_0 = 2$.

The time evolution of this contrived system is linear. The parameters defining the evolution depend nonlinearly on the initial condition, i.e. the parameters of the linear system are nonlinear functions of the initial condition. Therefore an emulator of this system takes time and the initial condition ($|\gamma| = 1$) as inputs. The behavior of this system meets the above assumptions underlying a MEM with a linear time-invariant prior. Thus an emulator without coupling noise solves this system exactly.

The first dataset consists of simulated time series with 40 output observations for 200 initial conditions $x_0 \in [-1, 1]$.

The second dataset is built by mixing the trajectories of the dynamical system in eq. (32), specifically:

$$\hat{x}(t, x_0) = \int_\alpha x(t, x_0 + \alpha) \mathcal{N}_\alpha(x_0, \sigma^2) \mathrm{d}\alpha, \tag{35}$$

where $\sigma = 0.5$. Although the nonlinearities in the dynamical system still depend only on the initial condition, the smoothing couples neighboring trajectories.

For the mechanistic emulation we will use a 1 dimensional linear proxy:

$$\frac{\mathrm{d}s}{\mathrm{d}t}(t) = \theta_1(s_0)s(t) + \theta_2(s_0), \tag{36}$$

and use the knowledge from the first simulator to set $s_0 = x_0$, $\theta_1(s_0) = a(s_0)$, and $\theta_2(s_0) = b(s_0)$. Since each proxy is the solution of the system in eq. (32), no coupling of the components of the MEM is needed in the first case, i.e. $\Sigma = I$. For the second simulator we will couple the components using a 1 dimensional Matérn covariance function and the parameters mapping will be learned from the data.

2.3.2 Wartegg catchment dataset

This dataset was generated from a SWMM model of a $2.64\,\text{km}^2$ urban catchment located in the city of Lucerne in the canton of Lucerne, Switzerland. This model has been calibrated satisfactorily to observed rainfall-runoff and used for hydrological studies of the site [Tokarczyk et al., 2015, detailed model description provided therein].

The dataset consists of 900 time series with 2880 data points, simulating 24 h of water levels in an open outlet during different rain events. To drive the system into a highly nonlinear behavior we synthetically generated rain events covering a wide range of return periods. These events were generated following a block rain model [Gujer, 2007, Ch. 13.2] parametrized by intensity (I) and duration (d), i.e. $\gamma = (I, d)$ and $|\gamma| = 2$. The intensity of the event spans 30 values in the range $I \in [10, 100]\,\text{mm}\,\text{h}^{-1}$, with 30 different durations ($d \in [10, 240]\,\text{min}$) for each intensity.

For the mechanistic emulation we will use a 1 dimensional linear proxy for the water level:

$$\frac{\mathrm{d}h}{\mathrm{d}t}(t) = \theta_1(\gamma)h(t) + \theta_2(\gamma)R(t, \gamma) \tag{37}$$

where $R(t, \gamma)$ is the rain event used in the simulation. The values of $\theta_1(\gamma)$ and $\theta_2(\gamma)$ are obtained by a least squares fit of the data.

Due to the low performance achieved by standard MEMs in this dataset, the actual MEM presented in Sec. 3.3 uses a Wiener model as proxy. This is done using Warped GPs (Sec. 2.2.4), with a nonlinearity given by $\hat{h}(t) = a(\gamma)\tanh(b(\gamma)h(t) + c(\gamma))$, where the parameters a, b and c are learned from the data.

2.3.3 Adliswil catchment dataset

This dataset was generated from a SWMM model of a $1.63\,\text{km}^2$ urban catchment located in the city of Adliswil in the canton of Zurich, Switzerland. The dataset was used in Machac et al. [2016a](detailed model description provided therein) to create an emulator which was then used to speed-up the calibration (identification) of the parameters in the simulator.

The dataset consist of 256 time series with 601 data points, all of them corresponding to a single rain event, but with different 8 dimensional input parameter vectors ($|\gamma| = 8$). The time series are simulations of inflow to the local WWTP, and the parameters describe the physical properties of the sewer network.

For the mechanistic emulation we will use the same linear proxy as in Machac et al. [2016a], a 1 dimensional ODE for the discharge:

$$\frac{\mathrm{d}Q}{\mathrm{d}t}(t) = \theta_1(\gamma)Q(t) + \theta_2(\gamma)R(t) \tag{38}$$

where $R(t)$ is the measured rain event.

Two MEMs will be built, the first uses the values of $\theta_1(\gamma)$ and $\theta_2(\gamma)$ using the relation from Machac et al. [2016a], and the second will obtain them from a least squares fit of the data.

2.4 Performance assessment

To asses the quality of an emulator we compute the following emulation errors on the data reserved for testing:

i. Maximum absolute error (MAE)

$$e_{\mathrm{MAE}}(\boldsymbol{\gamma}_j) = \max_i \left| \hat{y}(t_i, \boldsymbol{\gamma}_j) - y(t_i, \boldsymbol{\gamma}_j) \right| \tag{39}$$

where $\hat{y}(t, \boldsymbol{\gamma})$ and $y(t, \boldsymbol{\gamma})$ are the emulated and simulated responses, respectively.

ii. Root mean square error (RMSE)

$$e_{\mathrm{RMSE}}(\boldsymbol{\gamma}_j) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}(t_i, \boldsymbol{\gamma}_j) - y(t_i, \boldsymbol{\gamma}_j) \right)^2} \tag{40}$$

Error e_{MAE} measures the error in reproducing extreme values and/or peaks in the simulated signals, while error e_{RMSE} measures an overall quality of the emulation.

3 Results

3.1 Nonlinear system dataset

This dataset is used to highlight the value of the mechanistic over data-driven emulation. It is suited for illustration purposes, since the surface to be reconstructed can be plotted ($|\boldsymbol{\gamma}| = 1$).

Fig. 2 shows the response of the system described in eqs. (32)-(34) as a surface $\mathbb{R}^2 \to \mathbb{R}$. Fig. 2a illustrates the weakness of the SVD emulator, which does not exploit the simulator's parameter dependence. Moreover predictions at unseen time points are provided by linear interpolation of the SVD basis. To build the MEM we used the exact simulator's parameter dependence. Although training data is sparse in the time direction, since the MEM encodes the right time evolution, the reconstruction quality is high, as can be seen in Fig. 2b. Although the SVD emulator could be improved by using an exponential basis for time interpolation [Franz and Gehler, 2006], i.e. the one provided by the covariance function of the MEM, the recovered parameter dependence will still be poor due to the low sampling of the parameter space.

This example shows that mechanistic emulation is ideal for situations where there is good prior knowledge and the training data is sparse. This is even more striking when data corresponding to different simulator parameters is sampled at different times, e.g. with adaptive stepsize simulators. In this case mechanistic emulation can be applied directly, while SVD emulation becomes complicated as a matrix completion problem needs to be solved before factorization can be applied [see Oh, 2010, for an SVD relevant analysis]. In a similar fashion the Kalman smoothing algorithm described in Reichert et al. [2011] also requires a first step in which all the data is interpolated into the same temporal grid, e.g. via interpolation.

3.2 Nonlinear system dataset II

Since the structure of the proxy of a MEM does not change once its dimension m is set, we can optimize the proxy's parameters to reduce epistemic biases. The example shown in Fig. 3 illustrates the effect of mismatches between

13
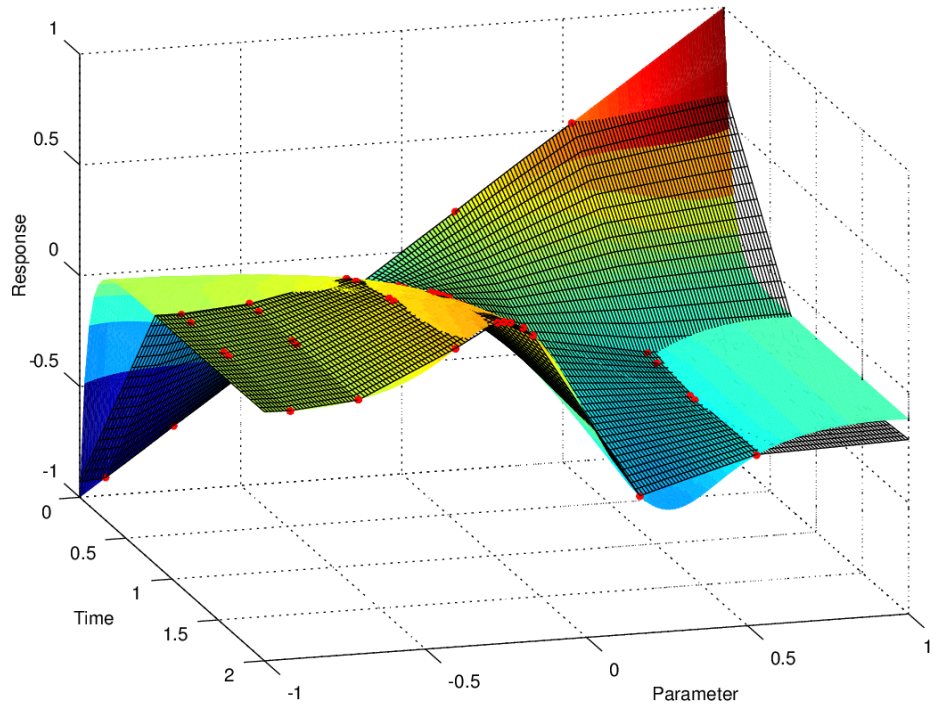
(a) SVD
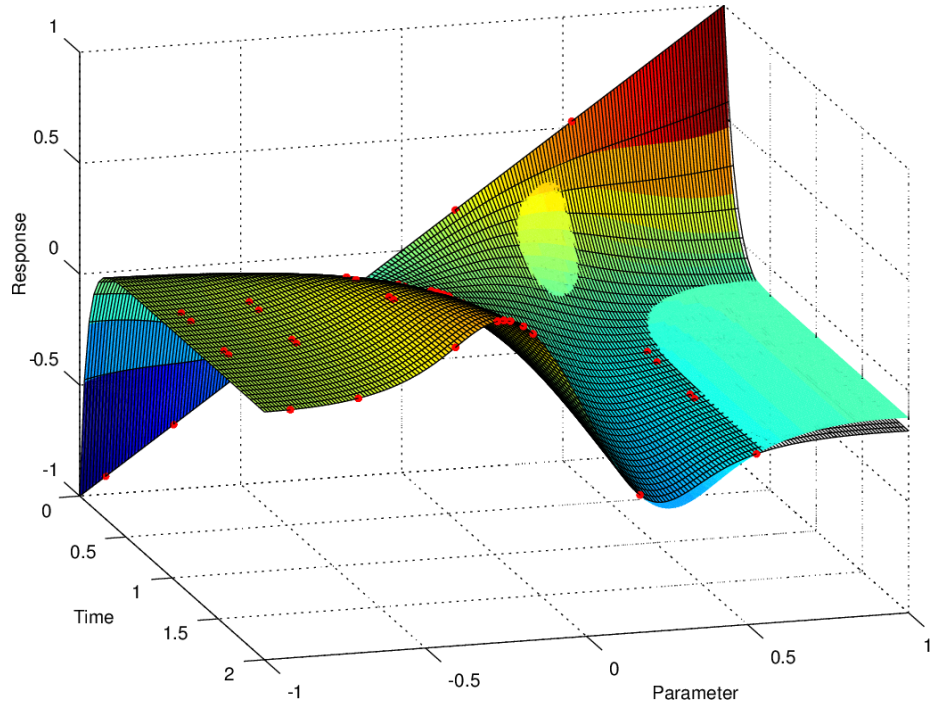
(b) MEM

Figure 2: Nonlinear system emulation of didactical model. The colored surface shows the behaviour of the simulator output. Red dots show the training data. The emulated surface is shown with a black wireframe. In this contrived scenario a MEM (b) outperforms an SVD emulator (a). The failure of SVD is mainly due to the linear interpolation in the time direction.

14

the dynamical system and the proxy, i.e. epistemic bias, and how it can be mitigated. In Fig. 3a we show the performance of a MEM built with the same proxy as in Fig. 2b, but used to emulate the nonlinear dynamical system described at the end of Sec. 2.3.1. Comparing with Fig. 2b, we see that in the regions where there is no observed data, the emulation is biased. A MEM with proxies fitted to the data is shown in Fig. 3b. In this case the epistemic bias is considerably reduced although we did not used mechanistic knowledge to improve the parameters mapping.

Hence, if the mapping between the simulator parameters and the linear proxy's parameters cannot be exactly determined from the mechanistic knowledge, it is better to fit the proxy's parameters to the data instead of using and ad-hoc calculation based on on one's best guess or expert opinion. The latter can be improved a posteriori by studying the parameters mapping emerging from the fit.

3.3 Wartegg catchment dataset

Results of these simulations can be seen in Fig. 4. For rains shorter than a certain duration (about 20 min for the intensity used in the plot, $41\,\mathrm{mm\,h^{-1}}$) the water level response shows the typical wave of runoff in an open channel. After some critical duration, which corresponds to the catchment's time of concentration, the water level becomes constant and remains fixed for the duration of the rain, defining the triangular region marked in the plot. After the rain, the water level goes back to a lower fixed level and remains there for a period of time which is a nonlinear function of the rain duration, e.g. between 4-6 h for a 4 h event. This shows the nonlinear nature of the storage involved in the effective discharge of the network. Finally there is a slow decay in the level fueled by the residual water in the network, the rate of this decay also depends nonlinearly on the rain's duration.

Fig. 5 shows the distribution of the test error of a MEM[5] and a NMF emulator with 7 components. Table 2 summarizes the mean of the error distributions. The NMF emulator outperforms the mechanistic emulator. In previous trials, an SVD emulator was also built and provided results comparable with NMF (not shown here), however it produced negative predictions just before the steep increase of the water level at the beginning of the rain event.

| Emulator	MAE (m, %)	RMSE (m, %)
NMF(1×10^4)	3.2×10^{-2}, 7.7	0.94×10^{-2}, 4.1
MEM(1×10^2)	22.6×10^{-2}, 53.7	4.4×10^{-2}, 17.4

Table 2: Mean emulation errors corresponding to the Wartegg catchment dataset. The value in parenthesis is the simulation speed-up factor obtained with respect to the original simulator, e.g. if SWMM simulation takes 3 s, MNF emulator takes 0.3 ms.

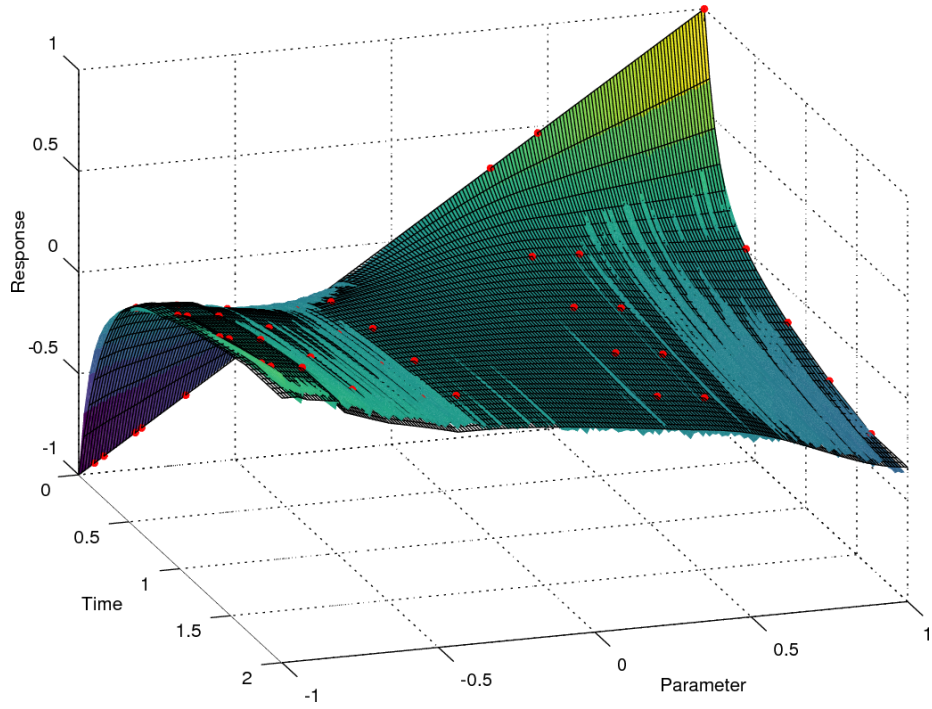Fig. 6 shows the quality of the NMF emulation for three different rain intensities.

3.4 Adliswil catchment dataset

In [Machac et al., 2016a] a SWMM model of the Adliswil catchment was emulated using a MEM with a prior derived from a simplified version of the

---

[5]Warped MEM, see Sec. 2.3.2. MEMs with linear proxies were unable to reduce average RMSE below 25%.
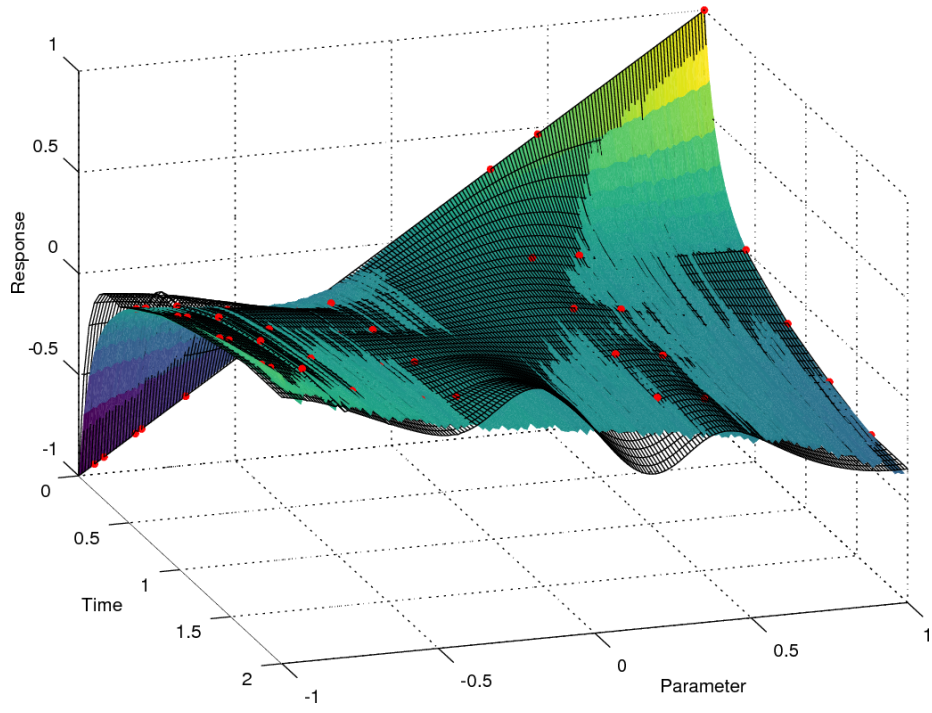
(a) MEM

(b) MEM fitted proxy

Figure 3: Nonlinear system emulation II. The colored surface shows the surface to reconstruct. In this second contrived scenario the MEM (b) performs the same as the SVD emulator (a). Red dots show the training data. The emulated surface is shown with a black wireframe.
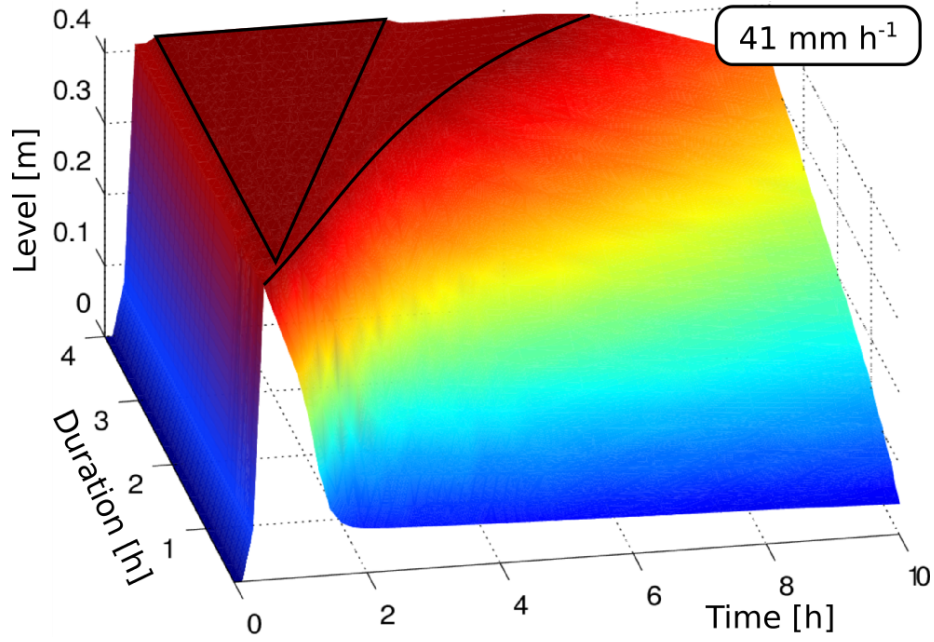
Figure 4: Simulator output for a fixed intensity ($41\,\mathrm{mm\,h^{-1}}$) and varying duration. The black bold lines label different runoff regimes: free-surface flows, runoff with activated storage and emptying of storage in the catchment.
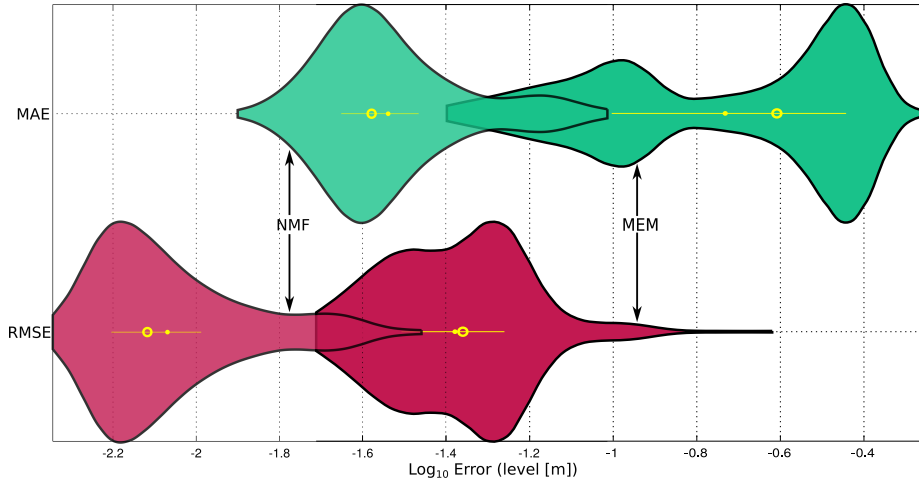
Figure 5: Test error distribution of the NMF emulator. The violin plots show the distribution of the logarithm of the error for the maximum absolute error (MAE) and the root mean square error (RMSE). Light colored plots corresponds to the NMF emulator, solid colored plots to the MEM. The mean(median) error is indicated with filled(empty) circles. See Table 2 for their numerical values.
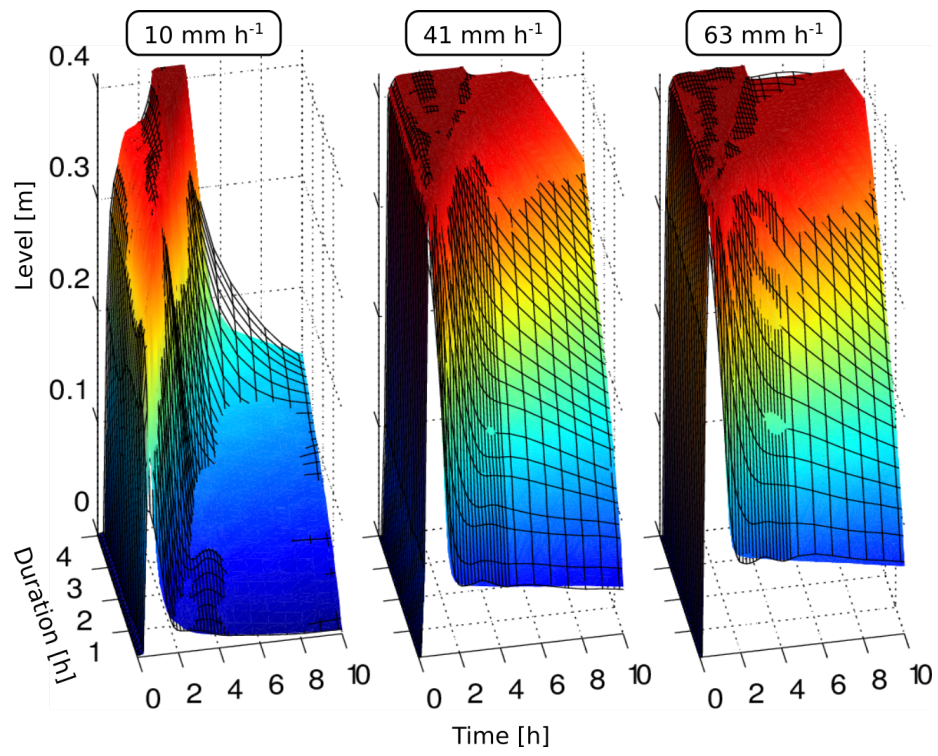
Figure 6: NMF emulation quality. The colored surface shows the simulation outputs for three different rain intensities. The NMF emulated surface is shown with a black wire-frame.

simulator's equations, with the aim of running a system identification task from a single rain event.

Figure 7 shows the distribution of the test error of a MEM and a SVD emulator with 6 components. Both errors are calculated on the same test set.
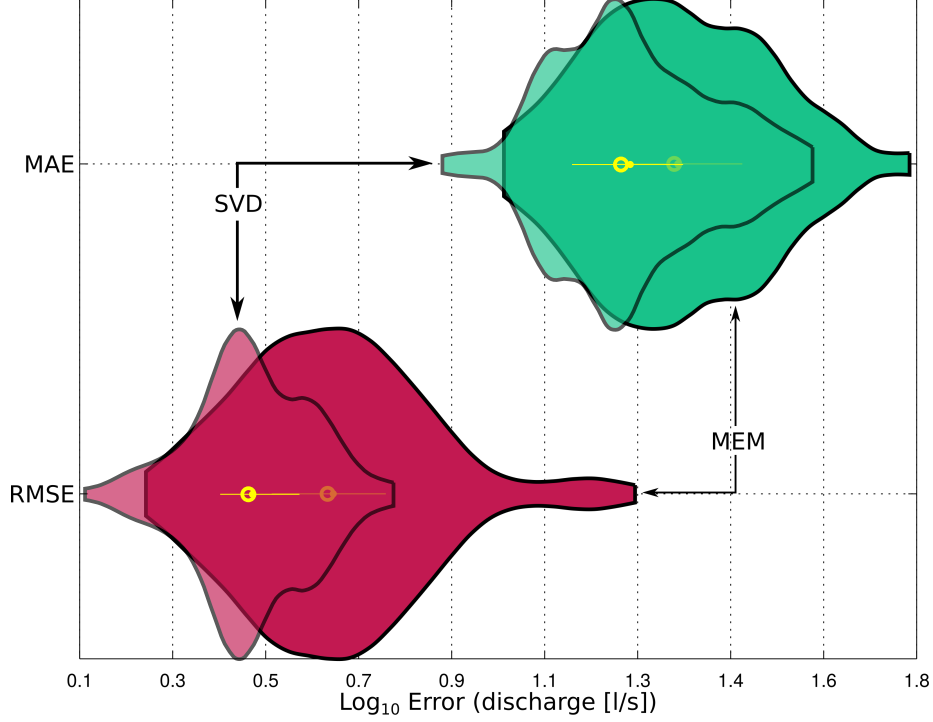

Figure 7: Test error distribution of the MEM from Machac et al. [2016a] and a SVD emulator. The violin plots show the error distribution for the mean maximum absolute error (MAE) and the root mean square error (RMSE). Light colored plots corresponds to the SVD emulator, solid colored plots to the MEM. The mean(median) error is indicated with filled(empty) circles. See Table 3 for their numerical values.

Table 3 summarizes the mean of the error distributions. In this example, the SVD emulator also performs better than its mechanistic counter part.

Although the time interpolation of SVD does not respect the dynamics of the system, the density of data is enough to provide a good estimation. The simplicity of the SVD emulator, when compared with the mechanistic one, makes this approach much more compelling for this application.

4 Discussion

The results presented in the previous sections seem to suggest that MEMs are not useful for emulating complicated hydrological or hydrodynamic simulators and that they remain as an academic curiosity. However MEMs still have many properties that can be exploited for better and faster emulation, which should not be ignored only due to the early state of the method. These known advantages include the suitability for parallelization and, speedups and energy saving via approximated computing [Angerer et al., 2015].

Emulator	MAE (ls^{-1}, %)	RMSE (ls^{-1}, %)
SVD(5×10^4)	18.3, 6.2	3.19, 2.6
MEM(1×10^3)	23.2, 7.9	4.94, 4.0
MEM-fit (1×10^3)	20.2, 6.9	3.42, 2.8

Table 3: Mean emulation errors corresponding to the Adliswil catchment dataset. MEM refers to an emulator in Machac et al. [2016a], while MEM-fit to an emulator with the same proxy structure but parameters fitted to the data. The value in parenthesis is the simulation speed-up factor obtained with respect to the original simulator, e.g. if SWMM simulation takes 3 s, SVD emulator takes 60 µs.

Table 4 summarizes the steps involved in the two approaches presented here. There we indicate the relation between the method and the characteristics of the dataset. On one side, although data-driven emulators are computationally and conceptually simpler than MEMs, they are more sensitive to the sparseness of the data. On the other side, MEM's performance is limited by the linearity of the prior which might fail to express our knowledge about the nonlinear simulator. If good prior knowledge is available, the mechanistic emulation can incorporate correct dependencies, which could be exploited to reduce the amount of data needed to achieve a desired performance.

Steps 1 and 2 of mechanistic emulation are the most sensitive to the mismatch between prior and simulator. To mitigate this, we keep the model structure provided by the prior and learn the parameter values from the data, thus providing the best linear proxy for the dataset (Sec. 3.2). This generally provides sharper error distributions than using the parameter values obtained directly from the prior [Albert, 2012]. Note that mitigating this simulator-prior mismatch is not a question of data quantity, since more data will override the prior and all mechanistic insights it provided, thus rendering mechanistic knowledge unnecessary [Steinke and Schölkopf, 2008]. More data would also increase the memory and computational resources needed by the emulator. Increasing the density of time samples will also reduce the condition number of the covariance matrices and the inversion problem at the training phase will be ill-posed, unless iterative condition methods are used [Reichert et al., 2011].

In step 2 of a data-driven emulator we need to factorize the data. If the data is very sparse the generalization quality of the factorization is expected to be poor. For data that is sampled at different temporal grids the situation becomes even more delicate since data preprocessing is required to build up a grid, e.g. via interpolation or matrix completion. Matrix factorization also provide features only at the observed inputs, therefore an interpolation method is required when emulating unseen time points at step 5. The Kalman smoothing implementation of the mechanistic emulation used in Reichert et al. [2011] will be similarly affected. Optimizing this interpolation can be as hard as using a MEM directly (Sec. 3.1).

Equation (15) shows that the mean function of the prior GP is given by the solution to the noise-free linear ODE obtained by removing the noisy term of the SDE (11). This suggests a simple improvement of the emulator in which the mean function is replaced with a better approximator of the data. This is the underlying idea behind the work of González et al. [2014], in which the actuation affecting the mean of the GP is replaced by a signal generated by the nonlinear part of the model applied to a surrogate trajectory. In that work however, the surrogate trajectory, e.g. built with matrix factorization, does not encode mechanistic knowledge explicitly. This knowl-

Step	Mechanistic emulation	Data-driven emulation
1	Define prior	–
2	Obtain emulator parameters	Factorize the data
3	Define/Build parameters mapping	Build parameters mapping
4	Conditioning	–
5 (Emulation)	Matrix · vector	Matrix · vector + time interpolation

Table 4: Steps involved in the two emulation approaches described in this article. Data-driven approaches are simpler than mechanistic ones. Step 2 and 5 of data-driven approaches suffer if data is sparse or unevenly sampled. Steps 1 and 2 of the mechanistic approach suffer from the lack of expressiveness of linear priors.

edge could be introduced via the analytical solution of a nonlinear differential equation, such as the nonlinear Bernoulli ODE, or analytical approximations of more general nonlinear differential equations [Adomian, 1991]. These enhancements would only affect the mean function of the predictive GP, improving extrapolation quality and thus allowing for more sparse training sets. However, the estimation of uncertainties remains limited by the covariance function associated with the linear prior. This can be improved by including the mean function parameters in the conditioning step [Rasmussen and Williams, 2006, sec. 2.7].

All these observations are derived from more general results on the reducibility and emulation readinness of general simulators and not only valid for the especific simulators we used here. This is specially true for the data-driven approach, see for example Ch. 5 of Quarteroni et al. [2016].

5   Conclusions

We provided a comparison of mechanistic and data-driven emulation in several examples pertinent to the field of hydrology and urban water management. In all of these, data-driven emulation outperforms mechanistic emulation. The current state of MEMs makes them advantageous to fully data-driven emulators, when the training data is sparse and unevenly sampled. This is the case when many simulation runs with high temporal resolution are prohibitively expensive, or when adaptive stepsize simulators are used. If the only objective of emulation is to obtain a fast tool to replace a simulator, there seem to be no advantage in using mechanistic knowledge besides the case of sparse and unevenly sampled data mentioned before. The gain obtained from enhancements of MEMs discussed here, such as Wiener model proxies, Nonlinear mean functions, and hybrid mechanistic/data-driven emulators should be quantified in relation to the test error of an inexpensive data-driven emulator.

Acknowledgements

References

Anthony O'Hagan. Bayesian analysis of computer code outputs: A tutorial. *Reliability Engineering & System Safety*, 91(10-11):1290–1300, 2006. doi: 10.1016/j.ress.2005.11.025.

Ulrike Baur, Peter Benner, and Lihong Feng. Model order reduction for linear and nonlinear systems: a system-theoretic perspective. *Archives of Computational Methods in Engineering*, 21(4):331–358, 2014.

Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced Basis Methods for Partial Differential Equations*, volume 92 of *UNITEXT*. Springer International Publishing, Cham, 2016. ISBN 978-3-319-15430-5. doi: 10.1007/978-3-319-15431-2.

Yu-Geng Xi, De-Wei Li, and Shu Lin. Model Predictive Control — Status and Challenges. *Acta Automatica Sinica*, 39(3):222–236, mar 2013. ISSN 18741029. doi: 10.1016/S1874-1029(13)60024-5.

Guangtao Fu, Soon-Thiam Khu, and David Butler. Multiobjective optimisation of urban wastewater systems using parego: a comparison with nsga ii. In *11th International Conference on Urban Drainage*, 11 ICUD, Edinburgh, Scotland, 2008.

L.A. Rossman. Storm Water Management Model User's Manual Version 5.0. Technical Report EPA/600/R-05/040, National Risk Management Research Laboratory, Office of Research and Development, U.S. Environmental Protection Agency, 2010.

Tomaso Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990. ISSN 00189219. doi: 10.1109/5.58326.

Florian Steinke and Bernhard Schölkopf. Kernels, regularization and differential equations. *Pattern Recognition*, 41(11):3271–3286, 2008. ISSN 00313203. doi: 10.1016/j.patcog.2008.06.011.

Carlo Albert. A mechanistic dynamic emulator. *Nonlinear Analysis: Real World Applications*, 13(6):2747–2754, 2012.

Simo Särkkä, Arno Solin, and Jouni Hartikainen. Spatiotemporal Learning via Infinite-Dimensional Bayesian Filtering and Smoothing: A Look at Gaussian Process Regression Through Kalman Filtering. *IEEE Signal Processing Magazine*, 30(4):51–61, jul 2013. ISSN 1053-5888. doi: 10.1109/MSP.2013.2246292.

Javier González, Ivan Vujačić, and Ernst Wit. Reproducing kernel Hilbert space based estimation of systems of ordinary differential equations. *Pattern Recognition Letters*, 45:26–32, 2014.

Arno Solin. Explicit Link Between Periodic Covariance Functions and State Space Models. *Proc. of the Seventeenth Int. Conf. on Artificial Intelligence and Statistics*, 33:904–912, 2014.

C E Rasmussen and C K I Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation And Machine Learning. MIT Press, 2006.

P. Reichert, G. White, M.J. Bayarri, and E.B. Pitman. Mechanism-based emulation of dynamic simulation models: Concept and application in hydrology. *Computational Statistics & Data Analysis*, 55(4):1638–1655, 2011. doi: 10.1016/j.csda.2010.10.011.

Per Christian Hansen. *Rank-Deficient and Discrete Ill-Posed Problems*. Society for Industrial and Applied Mathematics, jan 1998. ISBN 978-0-89871-403-6. doi: 10.1137/1.9780898719697.

C.M. Angerer, R. Polig, D. Zegarac, H. Giefers, C. Hagleitner, C. Bekas, and A. Curioni. A fast, hybrid, power-efficient high-precision solver for large linear systems based on low-precision hardware. *Sustainable Computing: Informatics and Systems*, 2015. ISSN 2210-5379. doi: 10.1016/j.suscom.2015.10.001.

J D Victor and P Johannesma. Maximum-entropy approximations of stochastic nonlinear transductions: an extension of the Wiener theory. *Biological cybernetics*, 54(4-5):289–300, 1986. ISSN 0340-1200.

George Christakos. Spatiotemporal information systems in soil and environmental sciences. *Geoderma*, 85(2-3):141–179, 1998. ISSN 00167061. doi: 10.1016/S0016-7061(98)00018-4.

J. Harte. *Maximum Entropy and Ecology: A Theory of Abundance, Distribution, and Energetics*. Oxford Series in Ecology and Evolution. OUP Oxford, 2011. ISBN 9780191621161.

Jan S. Hesthaven, Gianluigi Rozza, and Benjamin Stamm. *Reduced Basis Methods*, pages 27–43. Springer International Publishing, Cham, 2016. ISBN 978-3-319-22470-1. doi: 10.1007/978-3-319-22470-1_3.

Boian S. Alexandrov and Velimir V. Vesselinov. Blind source separation for groundwater pressure analysis based on nonnegative matrix factorization. *Water Resources Research*, 50(9):7332–7347, 2014. ISSN 1944-7973. doi: 10.1002/2013WR015037.

Jingu Kim and Haesun Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 353–362, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3502-9. doi: 10.1109/ICDM.2008.149.

George S. Kimeldorf and Grace Wahba. A Correspondence Between Bayesian Estimation on Stochastic Processes and Smoothing by Splines. *The Annals of Mathematical Statistics*, 41(2):495–502, apr 1970. ISSN 0003-4851. doi: 10.1214/aoms/1177697089.

Thomas E Fricker, Jeremy E Oakley, and Nathan M Urban. Multivariate Gaussian Process Emulators With Nonseparable Covariance Structures. *Technometrics*, 55(1):47–56, feb 2013.

Y. A. Kuznetsov. Andronov-Hopf bifurcation. *Scholarpedia*, 1(10):1858, 2006. revision #90964.

David Machac, Peter Reichert, Jörg Rieckermann, and Carlo Albert. Fast mechanism-based emulator of a slow urban hydrodynamic drainage simulator. *Environmental Modelling & Software*, 78:54–67, 2016a. doi: 10.1016/j.envsoft.2015.12.007.

David Machac, Peter Reichert, and Carlo Albert. Emulation of dynamic simulators with application to hydrology. *Journal of Computational Physics*, 313:352–366, May 2016b.

Edward Snelson, Carl Edward Rasmussen, and Zoubin Ghahramani. Warped gaussian processes. In *In Advances in Neural Information Processing Systems (NIPS*, page 2003. MIT Press, 2003.

E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I (2Nd Revised. Ed.): Nonstiff Problems*. Springer-Verlag New York, Inc., New York, NY, USA, 1993. ISBN 0-387-56670-8.

P. Tokarczyk, J. P. Leitao, J. Rieckermann, K. Schindler, and F. Blumensaat. High-quality observation of surface imperviousness for urban runoff modelling using uav imagery. *Hydrology and Earth System Sciences*, 19(10): 4215–4228, 2015. doi: 10.5194/hess-19-4215-2015.

Willi Gujer. *Siedlungswasserwirtschaft*. Springer, Berlin, 3. edition, 2007.

Matthias O Franz and Peter V Gehler. How to choose the covariance for gaussian process regression independently of the basis. In *Proc. Gaussian Processes in Practice Workshop*, 2006.

Sewoong Oh. *Matrix completion: Fundamental limits and efficient algorithms*. PhD thesis, Stanford University, 2010.

G Adomian. A review of the decomposition method and some recent results for nonlinear equations. *Computers & Mathematics with Applications*, 21 (5):101–127, 1991. ISSN 08981221. doi: 10.1016/0898-1221(91)90220-X.

Octave community. GNU Octave 4.0.2, 2016. URL www.gnu.org/software/octave/. Last accessed Sept. 1, 2016.

The Sage Development Team. Sage Mathematics Software (Version 7.2), 2016. URL http://www.sagemath.org. Last accessed Sept. 1, 2016.

Inkscape community. Inkscape 0.91, 2016. URL http://www.inkscape.org/. Last accessed Sept. 1, 2016.