SPE-212204-MS

Guided Deep Learning Manifold Linearization of Porous Media Flow Equations

Marcelo J. Dall'Aqua, Texas A&M University; Emilio J. R. Coutinho and Eduardo Gildin, Texas A&M University; Zhenyu Guo, Hardik Zalavadia, and Sathish Sankaran, Xecta Digital Labs

Abstract

Integrated reservoir studies for performance prediction and decision-making processes are computationally expensive. In this paper, we develop a novel linearization approach to reduce the computational burden of intensive reservoir simulation execution. We achieve this by introducing two novel components: (1) augment the state-space to yield a bi-linear system, and (2) an autoencoder based on a deep neural network to linearize physics reservoir equations in a reduced manifold employing a Koopman operator. Recognizing that reservoir simulators execute expensive Newton-Raphson iterations after each timestep to solve the nonlinearities of the physical model, we propose "lifting" the physics to a more amenable manifold where the model behaves close to a linear system, similar to the Koopman theory, thus avoiding the iteration step. We use autoencoder deep neural networks with specific loss functions and structure to transform the nonlinear equation and frame it as a bilinear system with constant matrices over time. In such a way, it forces the states (pressures and saturations) to evolve in time by simple matrix multiplications in the lifted manifold. We also adopt a "guided" training approach: our training process is performed in three steps: we initially train the autoencoder, then we use a "conventional" MOR (Dynamic Mode Decomposition) as an initializer for the final full training when we use reservoir knowledge to improve and to lead the results to physically meaningful output.

Many simulation studies exhibit extremely nonlinear and multi-scale behavior, which can be difficult to model and control. Koopman operators can be shown to represent any dynamical system through linear dynamics. We applied this new framework to a two-dimensional two-phase (oil and water) reservoir subject to a waterflooding plan with three wells (one injector and two producers) with speed ups around 100 times faster and accuracy in the order of 1-3 percent on the pressure and saturations predictions. It is worthwhile noting that this method is a non-intrusive data-driven method since it does not need access to the reservoir simulation internal structure; thus, it is easily applied to commercial reservoir simulators and is also extendable to other studies. In addition, an extra benefit of this framework is to enable the plethora of well-developed tools for MOR of linear systems. This is the first work that utilizes the Koopman operator for linearizing the system with controls to the author's knowledge. As with any ROM method, this can be

directly applied to a well-control optimization problem and well-placement studies with low computational cost in the prediction step and good accuracy.

Introduction

Meeting the net-zero emission paradigm will require a realignment of hydrocarbon production strategies with other forms of energy production. Carbon capture storage and sequestration (CCS) and compressed hydrogen (H2) energy storage and geothermal energy production will contribute tremendously to achieving a sustainable form of cleaner energy production. Profiting from these energy sources is only possible if accurate and timely prediction of the injection-production behavior of fluids in the subsurface can be attained. This involves the development of subsurface and surface flow simulators that can generate accurate answers in a timely manner (real-time) using commodity-based platforms (e.g., laptops and tablets).

Integrated reservoir studies are computationally expensive, especially for performance prediction and decision-making processes. Reduced-order modeling (ROM) alleviates the computational burden of such large-scale simulations by running cheaper and faster models while preserving high accuracy. Model reduction appears in a wide range of large-scale nonlinear dynamical systems whose modeling needs unmanageable computational resources. Numerous solutions have been proposed to mitigate the computational effort in simulating such systems. For instance, physics-based model reduction algorithms, which rely on the model equations of the system, have been proven to work well for linear and mild nonlinear systems. Recently, machine learning (ML) algorithms have been used to enhance the capabilities of model reduction frameworks for highly nonlinear systems, as in the case of coupled flow and geomechanics and gas distributions and networks. In many cases, two- to three-fold speed-ups (100-1000 times faster) are expected to be attained in several applications.

However, despite the speed-up gains, ML techniques present inaccurate results when facing values outside the range from where they were trained. We aim, thus, to increase the prediction capabilities by adding physical insights and embedding the physical behavior of reduced-order models into the ML context.

In this work, we tackle the reduced complexity modeling paradigm for reservoir management and optimization applications as a basis for decision support under uncertainty. To this end, we apply concepts from model simplifications, especially reduced-order modeling, together with smart systems frameworks, such as control system theory, machine learning theory, real-time dynamical modeling, and high-performance computing to enhance current and newly developed algorithms for reservoir simulation.

Motivation

Reservoir management consists of several actions one must perform during the life cycle of a project to attain specific targets for several performance indicators, such as recovery factors or any economic metric (e.g., NPV). It has been a long quest in the industry and academia to try to find ways to improve the financial gains of oil exploration through optimization. Examples of such studies are the determination of the well location, well type (producer or injector, vertical or horizontal), drilling schedule, and, for existing wells, optimization of well controls, e.g., settings of bottom hole pressure or water injection rate (Hedengren et al. 2017; Hou et al. 2015).

Such problems can be mathematically formulated, in general, as a nonlinear programming (NLP):

$$\underset{D}{\text{minimize}} \quad \phi(y) \tag{1a}$$

$$\text{subject to} \quad \dot{x}(t) = f_1(x(t)) + f_2(x(t))u(t), \tag{1b}$$

$$y(t) = g_1(x(t)) + g_2(x(t))\, u(t), \tag{1c}$$

$$h(x(t), u(t)) \le 0, \tag{1d}$$

$$x(0) = x_0 \tag{1e}$$

where $x(t) \in \mathbb{R}^n$ is a state vector or solution trajectory of the system, $y(t) \# \mathbb{R}^p$ is an output vector of the dynamical system, $u(t) \in \mathbb{R}^m$ are the control variables, D represents the decision variables ($u(t)$) of the NLP solver, $\phi : \mathbb{R}^p \to \mathbb{R}$ is the objective function which must be minimized, eq. (1b) ($f_1 : \mathbb{R}^n \to \mathbb{R}^n$ and $f_2 : \mathbb{R}^n \to \mathbb{R}^{n\times m}$) are nonlinear equations that describe the performance of the system (e.g., material balances, production rates) and the inequality constraints eq. (1d) ($g_1 : \mathbb{R}^p \to \mathbb{R}^p$ and $g_2 : \mathbb{R}^p \to \mathbb{R}^{p\times m}$) can define process specifications or constraints for feasible plans and schedule.

To solve eq. (1b), engineers rely on model-based optimization, which depends on large-scale reservoir simulators to solve eq. (1b), which is generally a high fidelity model, i.e., the dimension of state vector $x(t)$, n, has the magnitude order of approximate $O(10^5)$ — $O(10^6)$. The issue is that such a computational task demands considerable time, hours, or even days for a single run. Therefore, optimization studies that require many forward runs are restricted to companies with excellent infrastructure with High-Performance Computers.

What is a Proxy/Surrogate Model

An alternative to the expensive numerical simulation is the so- called proxy-modeling, also known as surrogate modeling or metamodeling. A formal definition of proxies is given by Zubarev (2009) as a mathematically, statistically, or data-driven model defined function that replicates the simulation model output for selected input parameters.

There are countless ways to construct a proxy, such as polynomial regression, multivariable kriging, splines, and artificial neural network. Nevertheless, one fact is valid for all the methods; it approximates an existing numerical reservoir model and therefore contains errors. The modeler's task is to minimize this error to an acceptable value so that the proxy is considerably faster than the full numerical simulator without sacrificing the accuracy of the input-to-output behavior of the original simulator model.

This paper focuses on Model Order Reduction (MOR) technique, defined by Benner and FaSSbender (2013) as a "computational technique to reduce the order of a dynamical system described by a set of ordinary or differential-algebraic equations (ODEs or DAEs) to facilitate or enable its simulation, the design of a controller, or optimization and design of the physical system modeled". Therefore, in our view, MOR is a type of proxy modeling. We will explore some of the MOR in the following two sections.

Model Order Reduction – MOR

Several attempts to alleviate the computational cost of a forward run of eq. (1b) have been proposed over the years, such as surrogate or proxy models, upscaling, reduced-order models, and streamlines, among others (Jansen and Durlofsky 2017). Such proxy models should run much faster than the full-order representation while providing a sufficiently accurate approximation of the model outputs required for the optimization. We are interested in global model order reduction (MOR) techniques in this research; that is, we want a reduced-order model that covers the entire parameter domain (Benner, Gugercin, and Willcox 2015).

Physics-Based Method

Physics-based model reduction has been commonly built based on the projection- based framework (Benner, Gugercin, and Willcox 2015; Swischuk et al. 2019; Jansen and Durlofsky 2017) by choosing an r-dimensional ($r \# n$) test (V) and trial subspace (W), defined by the basis matrix $\mathbf{V} \in \mathbb{R}^{n\times r}$ and $\mathbf{W} \in \mathbb{R}^{n\times r}$, respectively, where V = $Range(\mathbf{V})$ and $W = Range(\mathbf{W})$. Then one "project" the states $x(t)$ into this new subspace, and one defines a new variable $z(t) \in \mathbb{R}^r$ such as $x(t) \approx \mathbf{V}z(t)$ and substitute it into eq. (1b). Finally one can define the residual $\mathbf{V}\dot{z}(t) - f_1(\mathbf{V}z(t)) - f_2(\mathbf{V}x(t))u(t)$, and enforce the Petrov-Galerkin condition:

$$\mathbf{W}^T\left(\mathbf{V}\dot{z}(t) - f_1(\mathbf{V}z(t)) - f_2(\mathbf{V}x(t))u(t)\right) = 0$$

which leads to the reduced-order model:

$$\mathbf{W}^T\mathbf{V}\dot{z}(t) = \mathbf{W}^T f_1(\mathbf{V}z(t)) + W^T f_2(Vx(t))u(t) \tag{2a}$$

$$y(t) = g_1(\mathbf{V}z(t)) + g_2(\mathbf{V}z(t))u(t) \tag{2b}$$

A significant question is how to select this basis matrix. The usual choice is Proper Orthogonal Decomposition - POD (Jansen and Durlofsky 2017; Antoulas 2005; Gildin et al. 2013; Yang et al. 2016; Ghommem, Gildin, and Ghasemi 2016; Tan et al. 2018; Florez and Gildin 2019), due to its broad applicability to linear and nonlinear systems. In a nutshell, POD assumes an orthogonal projection, **V** = **W**, and solves the following optimization problem in a set of "snapshots" (solution computed with the high-fidelity model) to identify the basis of such low dimensional space (Sirovich 1987)

$$\min_{\Phi \in R^{n\times r}} \| \mathbf{X} - \Phi\Phi^{*}\mathbf{X} \|_F = \sum_{k=r+1}^{n_s} \sigma_k, \tag{3}$$

which is solved by performing a Singular Value Decomposition *(SVD)* based on the Schmidt-Eckart-Young-Misky theorem (theorem 3.6 of Antoulas 2005).

However, a well-known bottleneck in the POD approach is in evaluating nonlinear terms whereby one needs to project back to the original large-scale state (for instance, $f_1(Vz(t))$ in eq. (2a)). This can increase the computational time and make the simulation of the reduced-order systems more expensive. A remedy for this is a second layer of approximation referred to as "hyper-reduction," and one of the first implementations was done by the discrete empirical interpolation method (DEIM) (Chaturantabut and Sorensen 2010), which reevaluates the nonlinear term only at subselection sampling points. Yang et al. (2016) has applied this technique in the context of porous media flow simulation. However, the approach taken in this paper may need a large number of bases to converge. It can yield instability in the reduced system, especially in highly nonlinear cases, such as the saturation equation.

Another remedy is the trajectory piecewise linear (TPWL) (Rewienski and White 2003) which approximates the nonlinearity by a weighted combination of linear systems around each trajectory. Along each trajectory section, the system is linearized and reduced. These reduced models are joined together to form the overall nonlinear reduced system. This approach was applied successfully to the subsurface flow simulation by Cardoso and Durlofsky (2010). However, this technique can be expensive in terms of the required space to save all the information, as it depends on the number of local approximations and the polynomial order. Moreover, the number of training samples needed for the reduced system may increase significantly for large-scale models.

A pioneer work developed by Ghasemi and Gildin (2015) has attempted to transform the two-phase water and oil reservoir equation into a QB form. They have utilized the concept of "lifting" (Kramer and Willcox 2020, 2019) by inserting new auxiliary variables into the nonlinear system. However, the system they worked have two sets of coupled nonlinear partial differential equations, the so-called pressures and the saturation equations, and they were only able to work in the saturation equation. Which left the pressure dependency of the saturation equations still acting throughout the simulation; as a result, the matrices have no constant values.

The Koopman theory is a similar concept of "lifting" (Brunton et al. 2021). Koopman's theory suggests that a broader set of "observables" may be more helpful in characterizing the dynamics. The suggestion is that by potentially picking a more comprehensive set of observables, information may be gained on the underlying nonlinear manifold on which the nonlinear dynamical system evolves (a sort of "Manifold Learning" from the machine learning community (Izenman 2008)).

This technique's challenge is finding these "observable functions" that can map the original nonlinear manifold into a possible linear or, more realistic, quasi-linear manifold. Several published papers (to cite a few: Williams, Kevrekidis, and Rowley 2015; Kaiser, Kutz, and Brunton 2021; Takeishi et al. 2017; Otto and Rowley 2019; Lusch, Kutz, and Brunton 2018; Gin et al. 2021) suggest that data-driven methods are the most promising solution to this issue, especially utilizing frameworks based on deep learning (LeCun, Bengio, and Hinton 2015; Goodfellow, Bengio, and Courville 2016).

Data-driven Based Method

The methods discussed so far are called "physics-based" methods as they heavily depend on the underlying mathematical formulation .e.g, PDE of the flow in porous media; another approach is the so-called data-driven methods. These methods assume no knowledge of the physics of the system to build a model based on the measurements. In the control literature, these techniques are known as "system identification", and a plethora of algorithms, such as ERA, Subspace Identification, and N4SID (Brunton and Kutz 2019), exist for determining an optimal identified model based on data.

In the last decade, data-driven methods received greater attention due to the industry transformation to the so-called industry 4.0 (Carvajal, Maucec, and Cullick 2017). In this new paradigm, automation has become a critical factor in adding value, especially with the deployment of data acquisition infrastructure (sensors) that can gather real-time measurements. The combination of the deluge of data streams (i.e., big data), very fast computers equipped with graphics processing units (GPU), and the rebirth of neural networks turned data-driven methods into the main topic of investigation in reservoir simulation.

These "identification" methods collect the inputs and the outputs (and/or the states) and try to identify the input-output relationship solely based on data. The most prominent of such methods is the Dynamic Mode Decomposition (DMD) (Schmid 2010; Tu et al. 2014). DMD is an equation-free, data-driven method that can be thought of as "an ideal combination of spatial dimensionality-reduction techniques, such as proper order decomposition (POD), with Fourier, transform in time". Hence, it provides dimensionality reduction in terms of a reduced set of modes and a model for how these modes evolve in time (Kutz et al. 2016). Zalavadia et al. (2019) successfully implemented this technique in reservoir modeling. However, their model was relatively simple, with very few nonlinearities. Souza et al. (2020) attempted on a more complex reservoir (more nonlinearities), but the technique failed because of the underlying assumption of linearity. A possible remedy would be to build the Koopman operator (Koopman 1931) on top of the DMD, which would first linearize the dynamics solving, in theory, this issue.

Navrátil et al. (2019) has proposed an entirely data-driven proxy model for reservoir simulation. Despite its impressive results (2000X faster than an industry-strength physics-based simulator Open Porous Media (OPM)), issues related to interpretability and robustness hinder its application in real case models. In addition, as pointed out by Willcox, Ghattas, and Heimbach (2021), collecting data from real engineering problems are costly and invariably comes with uncertainty and/or noise, which causes serious question of under-sampling issues and the ability to predict rare events. These are some of the main issues in deep learning-related techniques we address in this paper.

Hybrid Methods

One of our hypotheses of this paper is that combining physical and data-driven methods yields robust models. For instance, implementing physical losses in a neural net can help to guide data-driven strategies to obtain results that make engineering sense. This has been successfully implemented in the reservoir community (Jin, Liu, and Durlofsky 2020; Coutinho, Dall'Aqua, and Gildin 2021), in which they constructed a piece-wise linear model using deep learning techniques based on the method developed by Watter et al. 2015.

Background

In this section, we described the basic domain knowledge of the physics behind the work of this paper. We briefly review the concept of dynamical systems and tie it with the reservoir equations, which gives the bridge to the Koopman operator.

Dynamical System

A dynamical system is a system in which a function describes the time dependence of a point in a trajectory. Such a concept is broad enough to cover all the physical phenomena of the world, including electrical circuits, mechanical systems, climate, finance, neuroscience, and fluid flow. A more formal definition says that a future state depends on the previous inputs and states (Chen 1998).

A state $x(t_0)$ is the information, along with the input $u(t)$, that uniquely determines the output $y(t)$ for all $t \leq t_0$. In other words, it is the accumulation of all the previous inputs applied to the system since the beginning of time ($t = -\infty$) (Chen 1998).

It is common to represent such systems in what is called "state-space" representation. So, a general mathematical formulation of a continuous time dynamic system would be:

$$\dot{x}(t) = f(x(t), u(t)) \tag{4a}$$

$$y(t) = g(x(t), u(t)) \tag{4b}$$

where $\mathbf{f} : (\mathbb{R}^n \times \mathbb{R}^m) \rightarrow \mathbb{R}^n$ is a generic nonlinear vector function of $x \in \mathbb{R}^n$, with n being the number of states, and of $\mathbf{u} \in \mathbb{R}^m$, with m being the number of inputs. These variables completely define the dynamical evolution of the states in the state equation (eq. (4a)). Also, there is the output equation (eq. (4b)), which has the static map $g : (\mathbb{R}^n \times \mathbb{R}^m) \rightarrow \mathbb{R}^p$ that defines the output of the dynamic system, $\mathbf{y} \in \mathbb{R}^p$, with p being the number of outputs.

In the special case where $\mathbf{f}$ is nonlinear in the states but linear with respect to the controls $u(t)$, we can rewrite eq. (4a) as "control-affine" nonlinear equation:

$$\dot{x}(t) = f_1(x(t)) + f_2(x(t))u(t) \tag{5}$$

where $f_1 : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $f_2 : \mathbb{R}^n \rightarrow \mathbb{R}^{n\times m}$.

This formulation will play a central role in this work as porous media equations (i.e., reservoir simulation) are usually control-affine (Jansen 2013), and the Koopman transformation of such a system can lead to a bilinear equation (Goswami and Paley 2021).

Another special case is when eq. (4) are completely linear so that these equations can be rewritten in the following standard form:

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t) \tag{6a}$$

$$y(t) = \mathbf{C}x(t) + \mathbf{D}u(t) \tag{6b}$$

where the the $\mathbf{A} \in \mathbb{R}^{n\times m}$, $\mathbf{B} \in \mathbb{R}^{n\times m}$, $\mathbf{C} \in \mathbb{R}^{p\times m}$ and $\mathbf{D} \in \mathbb{R}^{p\times m}$ are respectively constant matrices. Heijn, Markovinovic, and Jansen 2004 have already explored such formulation in the porous media equation and showed its limitations. Nevertheless, eq. (6a) can be further manipulated to yield to piece-wise linearized systems, which are at the heart of the TPWL method (Rewienski and White 2003; Cardoso and Durlofsky 2010).

Numerical Petroleum Reservoir Simulation

Reservoir models combine conservation of mass, momentum, and energy equations to describe the multiphase flow in a porous media (Aziz and Settari 1979; Ertekin, Abou-kassem, and King 2001; Chen, Huan, and Ma 2006; Lie 2019). Our first assumptions to reach tractable mathematical equations are: the reservoir is isothermal and the system is immiscible. In particular, we will use the black oil models equations where the multi-components are lumped into pseudo-components.

Since the objective is to study control and optimization strategies for mature reservoirs, we will further restrict our model to a two-phase reservoir, water, and oil, which reasonably accurately describes typical waterflooding problems with no gas and well above the bubble point (Lie 2019). Further, we neglect capillary pressures because our interest relies on a field scale (Lie 2019).

The resulting Partial Differential Equation (PDE) can be rewritten after spatial discretization and some manipulations in a simplified matrix form (Jansen 2013):

$$\mathbf{V}(x)\frac{\mathrm{d}x}{\mathrm{d}t} + \mathbf{T}(x)X + \mathbf{F}(x)u = R(\dot{x}, x, u), \tag{7}$$

where $X = [p, s]^T$ is the state vector composed of pressures and saturations in the grid-block centers, V is the accumulation matrix (with entries that are functions of the porosity ϕ, and the oil, water, and rock compressibilities c_o, c_w and c_r), **T** is the transmissibility matrix (with entries that are functions of the rock permeabilities k, the oil and water relative permeabilities k_{ro} and k_{rw} and the oil and water viscosities μ_w, and μ_o), **F** is the fractional-flow matrix (with entries that have functional dependencies similar to those of T), and u is a vector of total well flow rates or bottom-hole pressures with non-zero values in those elements that correspond to grid blocks penetrated by a well. Here, R is the residual vector that should be equal to zero.

State-Space Formulation. We note that for our purposes, we can recast eq. (7) into the so-called state-space formulation. To this end, we rewrite it explicitly in terms of the derivative of the states ($\dot{x}$):

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = -\mathbf{V}(x)^{-1}\mathbf{T}(x)X - \mathbf{V}(x)^{-1}\mathbf{F}(x)u \tag{8}$$

or in an even more concise way:

$$\dot{x}(t) = f_1(x(t)) + f_2(x(t))u(t) \tag{9}$$

where f_1 and f_2 are nonlinear functions dependent on the states that lump all the previous nonlinear functions. Also, we can see that controls enter linearly into the flow model; thus, this leads to a controlaffine function in eq. (9). We note that this conclusion will play an important role in the development of this work.

Koopman Theory

The Koopman theory suggests that it is possible to completely characterize a nonlinear system's behavior through measurement functions. In other words, the Koopman operator is an effective coordinate transformation in which the nonlinear dynamics appear linear through a tractable finite-dimensional representation. Although its roots were developed almost 100 years ago (Koopman 1931), Koopman's theory has attracted much attention in the last decade due to the high interest in data modeling. For a complete report about this theory, the reader is encouraged to go to Brunton et al. 2021. Here, we will summarize the idea.

Consider a nonlinear autonomous system:

$$\dot{x} = f(x(t)) \tag{10}$$

for this system, we are interested in finding a new set of coordinates that transform the original spate "x" into "z" such that

$$z(t) = \phi(x(t)) \tag{11}$$

where the dynamics are ideally linearized or, more realistic speaking, simplified. So that the eq. (10) becomes

$$\begin{gathered} \dot{z}(t) = \mathbf{L}z(t), \quad \text{or} \\ \dot{\phi}(x(t)) = \mathbf{L}\phi(x(t)) \end{gathered} \tag{12}$$

Simple example. Let's see a simple example to help us have a more practical feeling about the theory. Consider the following nonlinear system:

$$\dot{x} = x^2 \tag{13}$$

where we omit the time dependency for simplicity. If we measure it through the function $\phi(x) = e^{-\frac{1}{x}}$, and take the total derivative of it:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}(\phi(x)) &= \frac{\partial}{\partial x}(\phi(x))\frac{\mathrm{d}}{\mathrm{d}t}(x)\\ &= \left(x^{-2}e^{-\frac{1}{x}}\right)\left(x^{2}\right)\\ &= e^{-\frac{1}{x}}\end{aligned} \tag{14}$$

then we finally conclude that

$$\frac{\mathrm{d}}{\mathrm{d}t}(\phi(x)) = \phi(x) \tag{15}$$

which is linear!

Pictorial example. The underlying concept of the Koopman approach is pictorially depicted in fig. 1 for a discrete system. On the left, we see the original dynamics evolving in a nonlinear manifold. On the right, we see a linearized manifold by applying the Koopman operator through a measurement function.

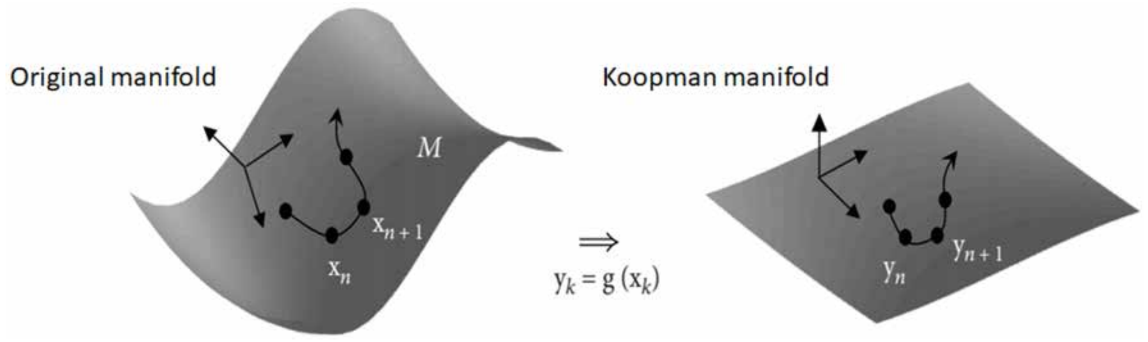

Figure 1—Koopman theory representation (Kutz, Proctor, and Brunton 2018).

Koopman with Control. The next step is to investigate how the Koopman operator changes with the addition of a control input (u). We recall eq. (12), but now we assume that the dynamical system has a control-affine form $\left(\dot{x}(t) = f(x) + \sum_{j=1}^{q} g_j(x)u_j(t)\right)$. Then we take the total derivative and apply the chain rule (Kaiser, Kutz, and Brunton 2021)

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}(\phi(x)) &= \nabla_x\phi(x)\cdot\dot{x}(t)\\ &= \nabla_x\phi(x)\cdot\left(f(x) + \left(\sum_{j=1}^{q} g_j(x)u_j(t)\right)\right)\\ &= \nabla_x\phi(x)\cdot f(x) + \nabla_x\phi(x)\cdot\left(\sum_{j=1}^{q} g_j(x)u_j(t)\right)\\ &= \mathbf{L}\phi(x) + \nabla_x\phi(x)\cdot\left(\sum_{j=1}^{q} g_j(x)u_j(t)\right),\end{aligned} \tag{16}$$

where $\nabla_x\phi(x)\cdot f(x) := \mathrm{L}\phi(x)$ at last line using the Koopman definition in eq. (12).

From eq. (16), the transformed manifold through Koopman loses the linearity property as in the autonomous case (eq. (12)). Now, the manifold has a control-affine form, being linear in $\phi(x)$ but nonlinear in $u(t)$ because of the term $g(x)$. The non-linearity of the Koopman operator when the original system has inputs bring a lot of concerns about the practicability of the theory.

However, in the case $\nabla_x\phi(x)\cdot g_j(x)$ lies in the span of $\phi(x)$ for all $j = 1, \ldots q,$ then the system eq. (16) is bilinearizable (Goswami and Paley 2021), leading eq. (16) to a bilinear form of the observable $\phi(x)$:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\phi(x)) = \mathbf{L}\phi(x) + \sum_{j=1}^{q} u_j\mathbf{N}_j\phi(x), \tag{17}$$

where $\mathbf{L}$ and $\mathbf{N}_j$ are constant matrices.

More importantly, an approximated bilinearization can produce accurate predictions, even if the system is not entirely bilinearizable (Goswami and Paley 2021).

Finding the Measurement Function. A central issue, and still an open question, is how to find the measurement function $\phi(x)$. The Koopman operator may be arbitrarily complex for relatively simple dynamical systems and will only be approximately represented on a finite basis (Brunton et al. 2021). Based on the success of Deep Learning representing arbitrarily complex functions in the last decade, the literature is currently putting its hope on it (Otto and Rowley 2019; Takeishi et al. 2017; Lusch, Kutz, and Brunton 2018; Gin et al. 2021; Han, Hao, and Vaidya 2020).

Methodology

This section describes a framework to create proxy models for reservoir numerical simulation. The goal is to have a reliable approximation that runs considerably faster than a conventional numerical simulator, allowing engineers to perform studies that remain "impossible" using today's technology due to the long time necessary to run conventional simulators. In particular, we are interested in enabling control optimizations in water flooding problems for reservoir management.

After an initial depletion of the reservoir, when the initial pressure produces the oil and gas, it is recommended to try to maintain the pressure above a specific value to avoid the formation of gas from the oil and keep the production. The most common way to do this is through the injection of water. Nevertheless, at a certain point, we get to a situation where there are several producing and several injectors wells, and for each well, we have some choices to make, either the bottom hole pressure or the rate of injection/production. Then, another question arises: what is the best setting for these that would maximize a particular metric? This is an optimization problem, and it relies on numerical simulators to test scenarios.

Over the last decades, impressive computer power has been made available to enable faster simulations, where we can cite High-Performance Computers (HPC) and, recently, Graphical Units Processing (GPU). However, with this increased power of computation, there was also an increase in the complexity of the models, to the point that some simulations still demand days to run.

It is our view that we need "simplified" versions of the original model to complement our understanding of physics by enabling very demanding computation studies. This is the goal of model order reduction (MOR), which has been very active in the past decade for reservoir simulation (e.g., upscaling, proxy and surrogate modeling, and parameterization). This paper does not intend to make a comprehensive study of all the MOR methods but to focus on the ones that meet some of our primary goals. That is, we want to develop a non-intrusive proxy; that is, we do not want access to the internal code of a numerical simulator and thus restrict ourselves to a specific program. This target leads us to data-driven methods such as DMD (Tu et al. 2014; Kutz et al. 2016) and rule out methods such as POD-DEIM (Chaturantabut and Sorensen 2010) and POD-TPWL (Rewienski and White 2003). However, as shown in Souza et al. 2020, this method did not successfully reproduce the behavior of the reservoir simulator since the underline assumption of this technique is the linearity of the model. We then drove our attention to emerging methods based on machine learning and artificial neural nets, such as the work of Coutinho, Dall'Aqua, and Gildin 2021, in which they used a neural net in an autoencoder architecture to construct a nonlinear map to the reduced space.

This work goes one step further by imposing linear constraints on the reduced space manifold. In such a way, the reduced space is somehow linear, and thus the dynamics can be simulated much faster, as conjectured by the Koopman theory (Brunton et al. 2021).

This section details the proposed methodology applied to two reservoir examples: one warmup problem, using one-dimensional classical Buckley Leverett, and one more complex problem, using a two-dimensional waterflooding study with injection wells.

Basics

We use a deep neural network to identify a more amenable set of coordinates for the reservoir dynamics. There are at least three critical components for successfully training the neural net: (a) network architecture, (b) domain knowledge constraints, and (c) data for training the network. These three issues will be developed in the following sections.

Proxy Architecture. For the first task, we have chosen an autoencoder architecture - AE (Goodfellow, Bengio, and Courville 2016) based on the suggestions of Brunton et al. 2021. However, a regular AE does not meet our need to impose linearity, so we add a matrix A in the latent space, mimicking the Koopman operator. As depicted in fig. 2, the encoder (Θ) acts as the Koopman operator, which brings the dynamics from the original space x_k to the new set of coordinates z_k (where k represents the time k), matrix A advances one time step the latent space z_k to $z_{k+}1$. Finally, the decoder, Θ^{-1}, acts as an inverse Koopman operator to bring the dynamics from the latent space $z_{k+}1$ to $x_{k+}1$. Note that multiple multiplications by the matrix A allow further time step prediction. So, the question of how this Artificial Neural Net (ANN) will achieve these goals will be answered in the section "Definition of the Loss Functions".

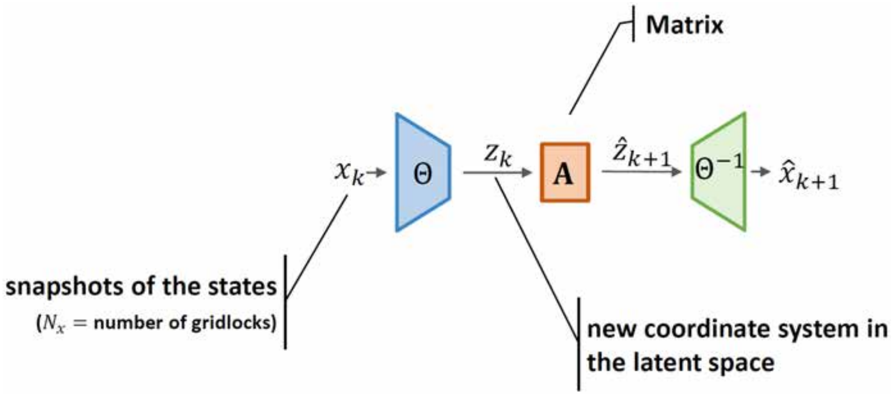

Figure 2—Koopman autoencoder network, where Θ represents the encoder, A is the matrix responsible for evolving the state in time, and Θ-1 represents a decoder.

In reality, the AE has a slightly more complex architecture than what fig. 2 shows. Inspired by Gin et al. 2021, the encoder and decoder are divided into two parts. The first part is a Neural Network, in which any conventional architecture can be used (e.g., fully connected, Convolution (CNN), Recurrent (RNN), Graph Neural Networks (GNNs), among others). On the other hand, the second part is a simple linear ANN (one layer with no activation function) to reduce or, if desirable, diagonalize the matrix A so there will be fewer parameters to discover during training. This framework is depicted in fig. 3.

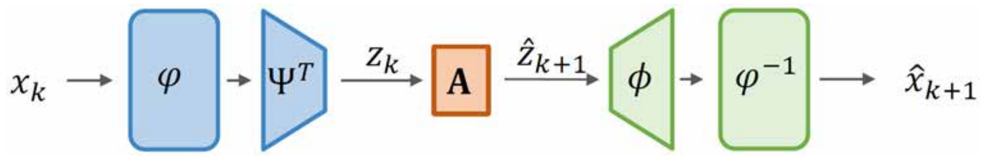

Figure 3—Detailed Koopman autoencoder architecture, where φ and φ^{-1} are conventional neural networks whose inputs and outputs have the same dimensions. Ψ^T and ϕ are linear one-layer neural networks.

By performing this separation, we improve the architecture's interpretability and allow the insertion of traditional MOR concepts, such as POD, into the workflow. Note that this second part acts as a projection.

From fig. 3, we can infer the following relations:

$$\varphi(x) = \Phi x \tag{18a}$$

$$x = \mathbf{\Psi}^{\mathbf{T}}\varphi(x), \tag{18b}$$

which corresponds precisely to the notion of projection discussed in the section "Physics-Based Method". Note the similarities between **W** and **V** with **Ψ** and **Φ**, respectively, for a case when the projections matrices are bi-orthogonal ($\mathbf{\Psi}^{\mathbf{T}}\mathbf{\Phi} = \mathbf{I}$).

Training/Validation versus Prediction versus Evaluation. Common to all machine learning (from the computer science community) and model order reduction algorithms (from the scientific computing community) is the strategy of dividing the process into two sequential steps. The first one, called "training" or "online", is when "labeled", or "known", data is used to identify the parameters of the proxy model. Note that "labeled" to dynamical systems means we know the output from a specific input.

Once the parameters of the proxy model are identified, for instance, weights, bias, and filters of a neural net, such calibrated model is used to perform predictions in an unknown dataset; that is, only the inputs are known. This second step is called "prediction".

However, there is an issue of not knowing if the prediction is accurate. To mitigate this risk, the training step subdivides the "labeled" data into training and validation sets. Then the training process calibrates the model using only the "training" set and estimates if the predictions will be accurate in the validation. More details about this process can be found in Hastie, Tibshirani, and Friedman 2009; James et al. 2021.

However, training a dynamic system is more complex than the typical application of machine learning (e.g., image recognition). Since a proxy not only has to predict the instantaneous output for an input but also make predictions in a future far from where it has been trained - in particular, dealing with extrapolations issues. In this case, the drawback is that the training procedure is more complex, and sometimes the network does not converge due to the gradient exploding in the automatic differentiation algorithm.

We have adopted two strategies to mitigate this issue. Firstly, we carefully initialize matrix A with all its eigenvalues with a magnitude less than one (that is, $|eig(\mathbf{A})| < 1$). This guarantees that the spectral radius is smaller than one ($\rho(\mathbf{A}) < 1$), and therefore any sequence of $x_{k+l} = \mathbf{A}x_k$ will converge to a finite number, independently of the initial value of x_k. This is a well-known property of numerical analysis; see Kincaid and Cheney 2002, for instance. Thus, initially, the system converges to a point, and the gradient from the AD algorithm does not explode.

The second strategy is to define a new hyperparameter,"p", and to insert a third step in the process, which we call "evaluation" (fig. 4). This new hyperparameter controls the number of predictions' timesteps by multiplying the matrix **A** "p" times so that $\hat{z}_{k+p} = A^p x_k$. The "evaluation" step is similar to the validation step but focuses only on the parameter p. The idea is that during the training/validation, we use the maximum possible value of p such that the process converges. Once this is done, we check how the tuned model behaves with a value of p corresponding to the maximum future we want to predict. This is similar to the hyperparameter optimization procedure (Géron 2017). We can note, then, that the parameter p controls two conflicting objectives of the training process. If p is small, the training will likely converge, but the proxy will likely perform poorly in long-term predictions. If, on the other hand, p is large, meaning that there is an ambitious goal to make very long-time predictions, the training procedure has a high risk of not converging.

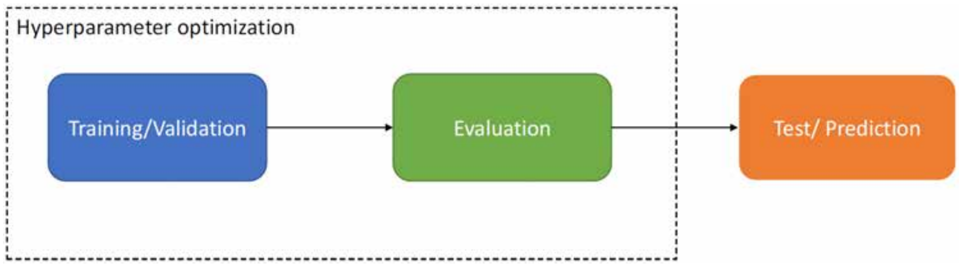

Figure 4—High-level proxy training flowchart from training to prediction.

Figure 5 gives a better picture of the differences between the evaluation and training/validation steps. Consider the example of a hypothetical dataset with four timesteps in the future; that is, one has the initial condition x_0 plus the four sub-sequential data x_1,x_2,x_3, and x_4. Ideally, one should set the parameter p to so that the proxy will learn to predict, at least, for the whole horizon. However, this is hard to achieve because the predictions go to infinity very soon since the initial guesses of the parameter are generally poor. To mitigate this, we use a small value horizon, for instance, p = 2 in fig. 5a. With this configuration, for each "real" data point, we advance two timesteps in the future (e.g., for data point x_2, we have the predictions $\hat{x}_{3_2}$ and $\hat{x}_{4_2}$ - the notation means the prediction of the data x at timesteps 3 and 4, respectively from the datapoint 2).

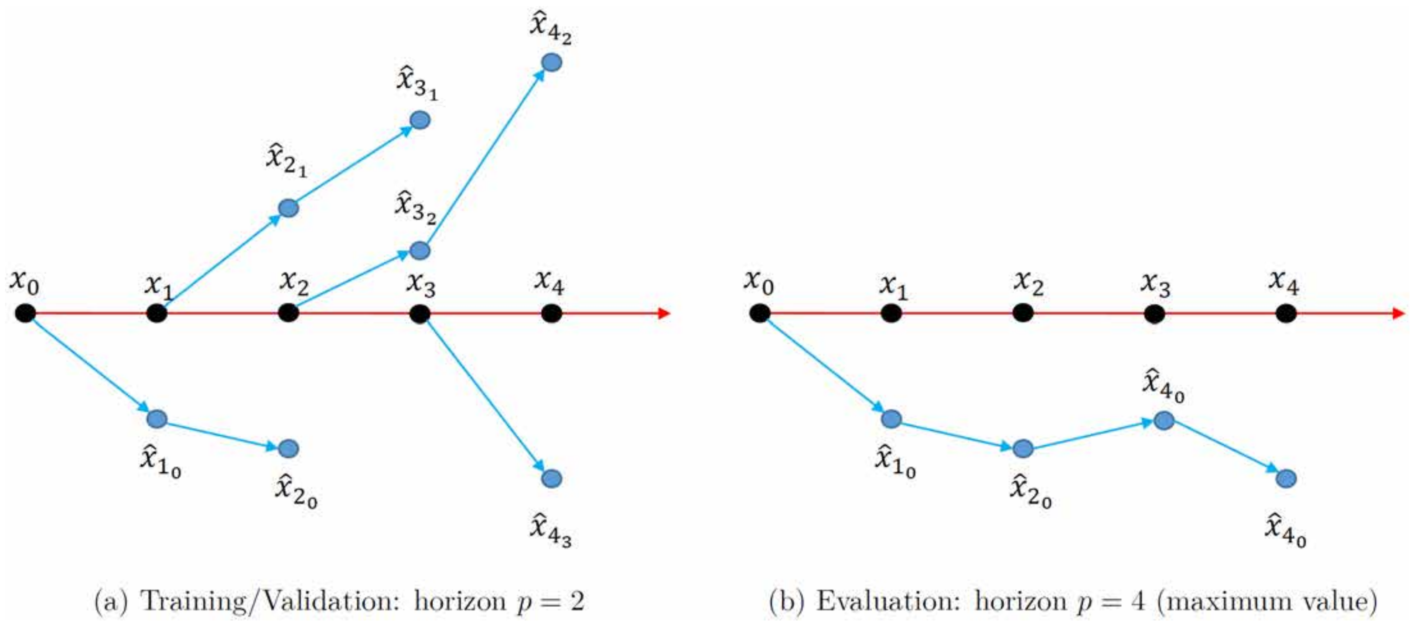

Figure 5—Training/Validation vs. Evaluation Step: where the red line represents the real-time evolution of the data and the blue lines the predictions and a regularization term:

Once the model is trained, the proxy parameter is "optimized"; it is possible to set the horizon to the maximum value available, that is, to the value of timesteps existing in the dataset. We start from the initial condition and check the value of the error until the final time (fig. 5b).

Definition of the Loss Functions. The key component for the network architecture of fig. 3 to work as the Koopman operator is the global loss function, which, in reality, will be the sum of different losses

$$\mathcal{L} = \sum_{i=1}^{L} \alpha_i \mathcal{L}_i + \mathcal{R}, \tag{19}$$

where L is the total number of losses, and α_i is a weight initially set to one as default.

A significant drawback of creating a cost function that is a summation of different losses is the question of how to scale the different values that can arise for simple reasons, such as the physical units of the metric. To the best of our knowledge, there is still no definitive answer for this problem, but it is an area of intense research (Wang, Yu, and Perdikaris 2022; Liu and Wang 2021). We try to address this issue by using a relative norm, thus making the values of each loss presents a low order of magnitude.

Each loss has a desired function which will be explained below and depicted in fig. 6. Note that we will call the total encoder Θ for notation, *N* is the different trajectories of the data (one forward simulation), *M* is the time steps of each trajectory, and *H* is the horizon considered (recall fig. 5a).

- Loss 1: Autoencoder loss: We want that the decoder acts as an invertible function of the encoder. So we can map back from the Koopman coordinate to the physical coordinates. Such loss is given by a relative mean squared error where the mean is taken over all initial conditions and all time steps:

$$\mathcal{L}_1 = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{M+1}\sum_{k=0}^{M}\frac{\| x_k^j - \Theta^{-1}\left(\Theta\left(x_k^j\right)\right)\|_2^2}{\| x_k^j \|_2^2} \tag{20}$$

- **Loss 2: Koopman loss:** Like loss 1, we want each part of the encoder and decoder to have its invertible function. Disentangling these two processes allows for better interpretability of what the neural net is doing under the hood:

$$\mathcal{L}_2 = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{M+1}\sum_{k=0}^{M}\frac{\| x_k^j - \varphi^{-1}\left(\varphi\left(x_k^j\right)\right)\|_2^2}{\| x_k^j \|_2^2} \tag{21}$$

- **Loss 3: Projection loss:** The projection loss has the same scheme as loss 2.

$$\mathcal{L}_3 = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{M+1}\sum_{k=0}^{M}\frac{\| \varphi\left(x_k^j\right) - \Phi\Psi^T\varphi\left(x_k^j\right)\|_2^2}{\| \varphi\left(x_k^j\right) \|_2^2} \tag{22}$$

  Worthwhile noting the relation with the projection framework by comparing the similarities between eqs. (3) and (22). They are identical if we disregard the relative norm and the averaging operator that only brings the value to a smaller magnitude, facilitating the optimizer's work and considering a Petrov-Galerkin projection.

- Loss 4: The prediction loss in the original manifold: The prediction loss will try to teach the ANN to predict the future. Starting from the initial condition, if we multiply *p* times the matrix *A* in the latent space and return to the original set of coordinates, the ANN should accurately predict state x_p^j.

$$\mathcal{L}_4 = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{M}\sum_{p=1}^{M}\sum_{i=\min(0,p-H)}^{p-1}\frac{\| x_p^j - \Theta^{-1}\left(A^p\Theta\left(x_{p_i}^j\right)\right)\|_2^2}{\| x_p^j \|_2^2} \tag{23}$$

- **Loss 5: Prediction loss in the reduced manifold:** Since the intrinsic coordinate has to be linear, we impose a prediction loss in the latent space:

$$\mathcal{L}_5 = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{M}\sum_{p=1}^{M}\sum_{i=\min(0,p-H)}^{p-1}\frac{\| \Theta\left(x_p^j\right) - A^p\Theta\left(x_{p_i}^j\right)\|_2^2}{\| \Theta\left(x_p^j\right) \|_2^2} \tag{24}$$

- **Regularization.** A common issue in all machine learning techniques is the trade-off bias-variance (Hastie, Tibshirani, and Friedman 2009; James et al. 2021). In a simplistic explanation, in extreme cases, the model is either too simple to capture all the nonlinearities of the data and is said to be "biased". Alternatively, it is too complex that it perfectly matches all the training data, including noise, thus "overfitting' in the training data but poorly predicting in an unknown data set. It is said that the NN has a high "variance" in the latter case.

Regularization attempts to reduce the variance by imposing a penalty on the parameters, then decreasing the "complexity" of the NN and hopefully preventing overfitting. Here, we have used Elastic Net regularization, which is simply a combination of L1 and L2 norms for all NN weights with two hyper-parameters factors:

$$\mathcal{R} = \lambda_1 \sum_i \| W_i \|_2 + \lambda_2 \sum_i \| W_i \|_1 \tag{25}$$

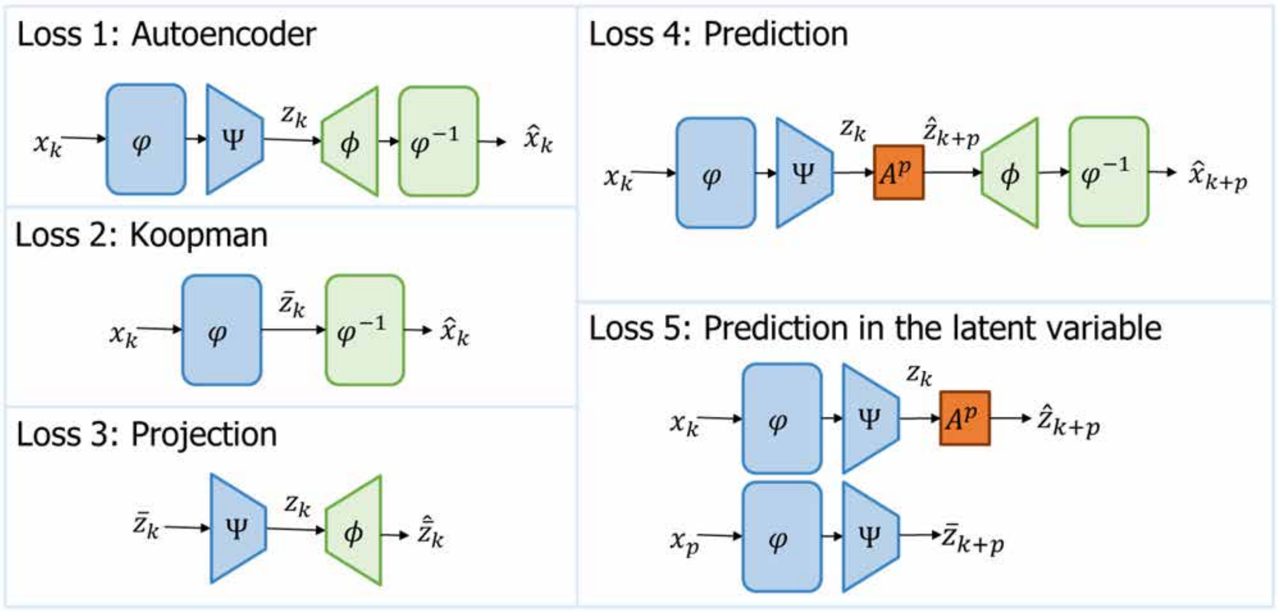

Figure 6—A depiction of Koopman autoencoder loss functions used for training the neural network. Adapted from Gin et al. 2021

Data Generation. Critical to the success of any machine learning and identification method is the overall quality of the training data (Katayama 2005), which should be "rich" enough (diverse) to learn a global coordinate transformation that holds for a wide variety of scenarios. The previous sentence is vague on purpose because, to the best of our knowledge, there is no definitive answer of what is the precise definition of "rich'.

However, we hope to achieve a good data set by generating several numerical simulations with different types of inputs so that there will be enough information in the data to allow the neural net to capture the system's dynamics. To this end, we will excite the simulator with a variety of signals depending on the model in question. More details about this will be given when we discuss some concrete examples.

As we will see, the amount of data necessary is still considerable. This is also a point where there is no definite answer in the field of neural networks and an interesting topic for future research: what is the minimum number of data necessary for a NN to be still able to perform good predictions?

Neural Networks Implementation and Practical Tricks. We have implemented all codes using Python 3.9 (Vanrossum 1995) and the Tensorflow framework (Abadi et al. 2015). We trained our models on GPUs NVIDIA A100 using the facilities of Texas A&M High-Performance Research Computing (www.hprc.tamu.edu).

We used a fully connected neural network for the forward and inverse Koopman blocks (fig. 3), with four hidden layers and an equal number of neurons corresponding to the total number of states of any specific problem. For instance, for a reservoir model with two phases, oil, and water, with the spatial discretization of 40×40×2 gridlocks for each dimension, there is a total of 6,400 states. Therefore our neural net has 6,400 neurons in all the layers (input, three hidden, and output). We used the hyperbolic tangent activation function *(tanh)* on all but the last layer.

We also tried different configurations, in particular the E2C (Watter et al. 2015). However, the results were not as promising, in addition to being much slower than the fully connected design. We conjecture, however, that Graph Neural Networks (GNNs) could be a good alternative for future work (Belbuteperes, Economon, and Kolter 2020; Eliasof, Haber, and Treister 2021; Chamberlain et al. 2021), but this application is out of the scope of the work presented here.

For the Projection block, as mentioned in section "Proxy Architecture", we used a neural network with only one layer, no activation function, and no biases. This configuration replicates a matrix multiplication, allowing this part of the proxy to behave as a projection.

We used the Adam optimization algorithm with an initial learning rate of 6.79×10^{-5} during the training procedure with a batch size of 32 (defined by a hyperparameter optimization). The sample is randomly selected to compose the training batch. We run the training procedure for 150 epochs.

Learning Rate Schedule. The learning rate of the optimizer is probably the most critical hyperparameter to set, along with the mini-batch size. It is usual to perform a hyperparameter optimization to decide its value, which generally consists of a brute force test of some pre-selected values. Needless to say, such a procedure is time-consuming and laborious.

Here, we tried to alleviate this burden by adopting a more strategic approach. We start a "high" learning rate and check the validation set. If we see no progress, we divide the learning rate by 2. See fig. 7 for more details. This process allows decreasing tests in the grid searching optimization. Instead of trying a wide range of values, we can focus only on a fewer number of maximum initial guesses.

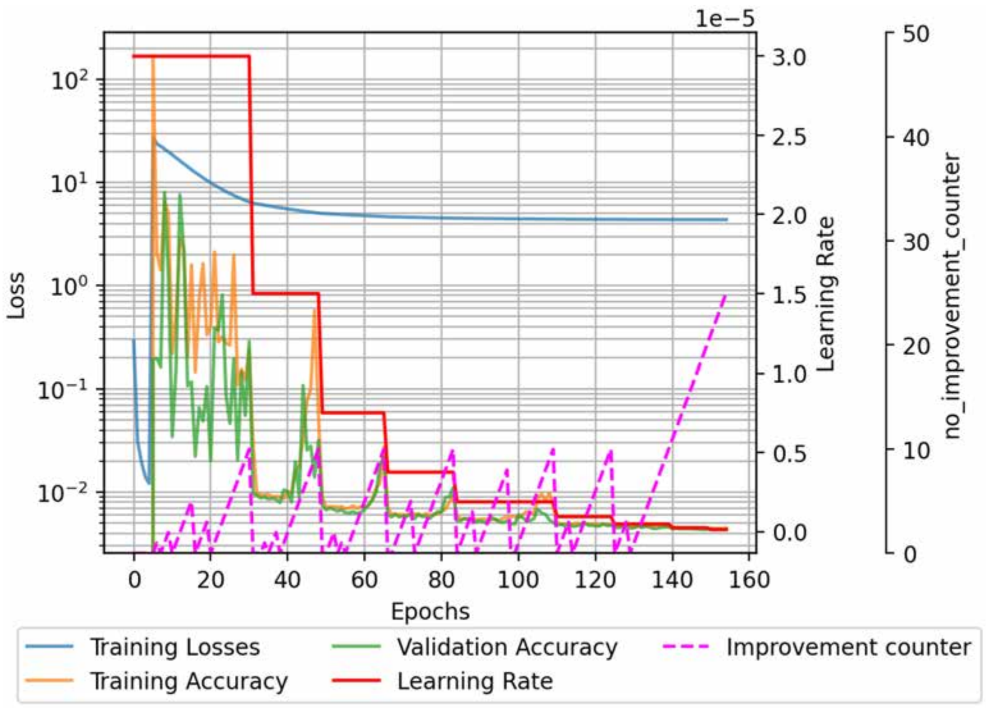

Figure 7—Learning rate schedule example: Here is an example of the scheduler. These figures show the training loss (blue line), training accuracy (orange line), and validation accuracy (green line) over 160 epochs. In addition, note that the learning rate decreases by half every time a counter (dashed pink line) reaches a threshold (10 in this example). Furthermore, the counter increase by one every time there is no improvement in the validation accuracy.

Normalization. To avoid the issue of input features having different scales (or units), which should be of approximately equal importance to the learning algorithm, we normalized our input data from the state and the well data and during the loss functions calculations. Normalization parameters (min and max) for each quantity are defined and used on this quantity over the training and prediction procedures. The normalization used here is:

$$Q_{NORM} = \frac{Q - Q_{min}}{Q_{max} - Q_{min}}, \tag{26}$$

where Q is the quantity to be normalized.

The minimum and maximum values for each quantity were selected from the input state (pressures and saturations), from the well data (well bottom hole pressure, oil flow rate, and water flow rate), and from the well controls (water injection rate, well bottom hole pressure).

Weight Initialization. Critical to the AD algorithm, initializing all parameters is one of the most important aspects to consider. There are two points to this: firstly, as in any gradient-based nonlinear optimization, the initial point dictates to which local minimum the algorithm will converge. To make the issue worse, we have no idea of the shape of the total cost function: we do not know if it is convex or concave and how many local optima (if any) exist. Therefore, we need to be extra cautious and use all the information available to increase the chances of success.

Secondly, we need to give variability to the initial guesses. The network computes the same output in an extreme case where all the neurons are initialized equally. Then they will also all compute the same gradients during back-propagation and undergo the same parameter updates.

In summary, for the Koopman blocks, we use a standard initializer from Tensoflow ("VarianceScaling"). The Projections blocks were initialized with an orthogonal matrix using a standard initializer from Tensoflow ("Orthogonal") because we want this block to be orthogonal to mimic a POD. The initialization of matrix A is more complex, and we used Dynamic Model Decomposition (DMD) and another MOR algorithm as an initializer (more details in section "Stepwise Training").

Stepwise Training. Continuing with the discussion of initialization, another strategy that has worked for us is what we call "Step-wise training". It consists of starting the training for a few epochs (3 or 5) using only the autoencoder blocks (Koopman and Projection) without any time evolution. Obviously, we only use losses 1 to 3 (eqs. (20) to (22)) at this point. After the training, we collect the snapshot of the reduced pseudo-linearized variable z, create the snapshot matrices for the DMD algorithm and thus initialize the matrix A (A = $Z'Z^{\dagger}$). After these two initialization steps, we go to the complete training of the proxy. Figure 8 depicts the flowchart of this process.

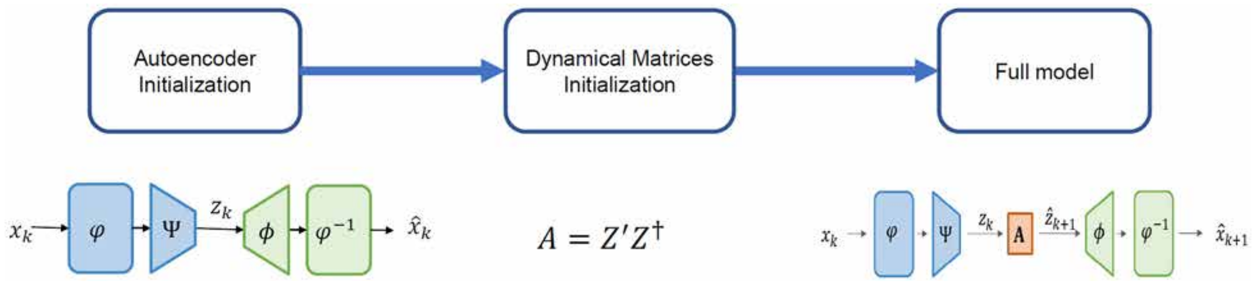
Autoencoder Initialization
Dynamical Matrices Initialization
Full model
x_k φ Ψ z_k ϕ φ^{-1} $\hat{x}_k$
$A = Z'Z^{\dagger}$
x_k φ Ψ z_k A $\hat{z}_{k+1}$ ϕ φ^{-1} $\hat{x}_{k+1}$

Figure 8—Step-wise training for the proxy with no controls: First, the autoencoder is initialized by training it for a few epochs. This first step will provide better initialization when training the full model. Second, we collect snapshots of the reduced linearized variable and performed DMD to initialize the dynamical matrix A. Lately, using the initialized autoencoder components and the matrix A, a full model training is performed

Noteworthy mentioning is the fact that training in parts is unnecessary, and you can train your proxy in one single shot. However, our tests have shown that the step-wise training led to better results. We see two probable explanations. Firstly, training only the autoencoder is more accessible than the proxy training with the time evolution requirement. We speculate that this first training brings the solution close to the attraction region of a local minimum. Secondly, the DMD algorithm brings knowledge from the data to the algorithm, which is better than random initialization.

Warmup Example: The Buckley-Leverett Problem. We apply the proxy described in section "Basics" for the Buckley-Leverett (BL) equation to prove the concept of using our architecture. This famous equation in the reservoir community models the displacement by the water of an immiscible and incompressible oil in porous media through a one-dimensional grid by combining mass balance equations and Darcy's law for each phase. We start with the partial differential equations (PDE) that dictates the flow:

$$\frac{\partial S_w}{\partial t} + \frac{\partial f_w(S_w)}{\partial x} = 0, \quad x \in [0, 1], \quad t \in [0, 1], \tag{27a}$$

$$S_w(x, \ t = 0) = 0, \tag{27b}$$

$$S_w(x = 0, \ t) = 1, \tag{27c}$$

Downloaded from http://onepetro.org/spersc/proceedings-pdf/23RSC/1-23RSC/D011S006R003/3092400/spe-212204-ms.pdf by China University of Petroleum (East China) user on 08 July 2023

where S_w is the water saturation and f_w is the fractional flow function. The fractional flow function is related to the properties of the rock-fluid interaction, and its values are usually obtained from the fluid's viscosity and relative permeability curves. The fractional flow will dictate the type of displacement one will observe, which can be grouped into three types of displacements with an increasing level of nonlinearity: concave, convex, and non-convex (see fig. 9).

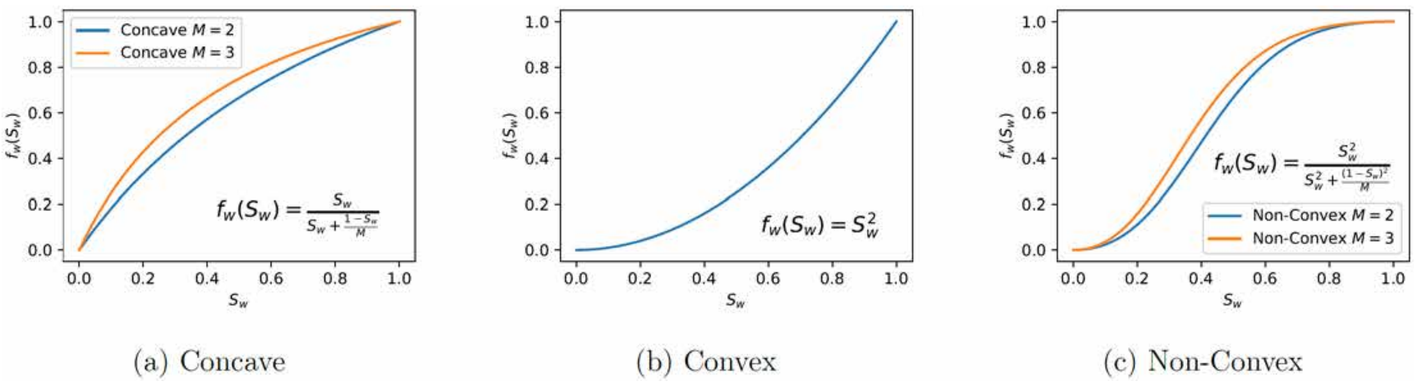

Figure 9—Fractional flow function types.

The concave fractional flow function is defined as:

$$f_w(S_w) = \frac{S_w}{S_w + \frac{(1-S_w)}{M}}, \tag{28}$$

which is displayed in fig. 9a. The parameter $M = \mu_o/\mu_w$ represents the viscosity ratio between the fluid originally in the porous media (oil) and the one used as a displacement fluid (water). This fractional flow function produces a solution that contains a single rarefaction wave, as seen in fig. 10a.

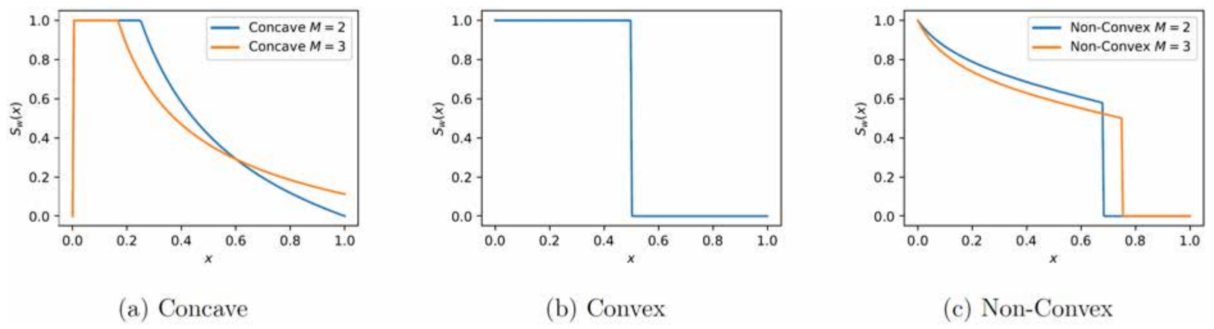

Figure 10—Exact solutions at $t = 0.5$ for the fractional flow functions in the previous figure.

The convex fractional flow function is simple:

$$f_w(S_w) = S_w^2, \tag{29}$$

which is displayed in fig. 9b. This fractional flow function leads to a single shock front in the solution, as seen in fig. 10b.

The most interesting case is that of a non-convex fractional flow function, given by:

$$f_w(S_w) = \frac{S_w^2}{S_w^2 + \frac{(1-S_w)^2}{M}}, \tag{30}$$

which can be seen in fig. 9c. The parameter M is as in the previous case. Figure 10c shows that this fractional flow function causes the solution to develop simultaneous rarefaction and shock waves.

Interestingly, BL's PDE is very similar to the Burger's equation, with the difference that the latter has an analytical Koopman operator: the Cole-Hopf transformation (Kevorkian 1990). Therefore, we have a twofold goal in this section: prove the concept as already mentioned and find the numerical linear transformation of the BL equation.

To this end, we will follow the stepwise training of fig. 8 where the states x are the pressure and saturations, and to create diversity in the training dataset we will generate several different initial conditions.

Data Set. A training data consist of 30,000 trajectories from the BL equation, each with 50 equally spaced time steps with dt = 10 days obtained by running a high-fidelity simulator. The validation data have the same structure as the training data but with 10,000 trajectories. To create a diversity of data, we came up with five different initial conditions: white noise, sines, square waves, Gaussians, and triangle waves (this one is reserved for the testing set, i.e., not used during the training).

The BL equation was discretized in 150 equally grid blocks, and the inner autoencoder ($\mathbf{\Psi}^T$ in fig. 3) was set to reduce its dimension to 21 (this is a hyperparameter and was not optimized).

Results. Figure 12 shows the results from the testing set for each type of curve when the initial condition is zero (which was not a part of the training set). It is important to mention that these results were achieved without any hyperparameter optimization; therefore, there is much room for improvement. Improving the results, however, is not the objective of this section, which is just to test the framework and see if it can work.

For the concave case (figs. 11a and 12a), we see almost perfect agreement between the exact solution and the result from the proxy at all times. For the nonconvex case (figs. 11b and 12b), we can see that the three-dimensional plot of the proxy is not as smooth as the exact solution. We also see a larger error in the snapshot at 50% of the total times, which decreases at 100%. Nevertheless, the agreement is good overall, considering that no hyperparameter optimization was performed.

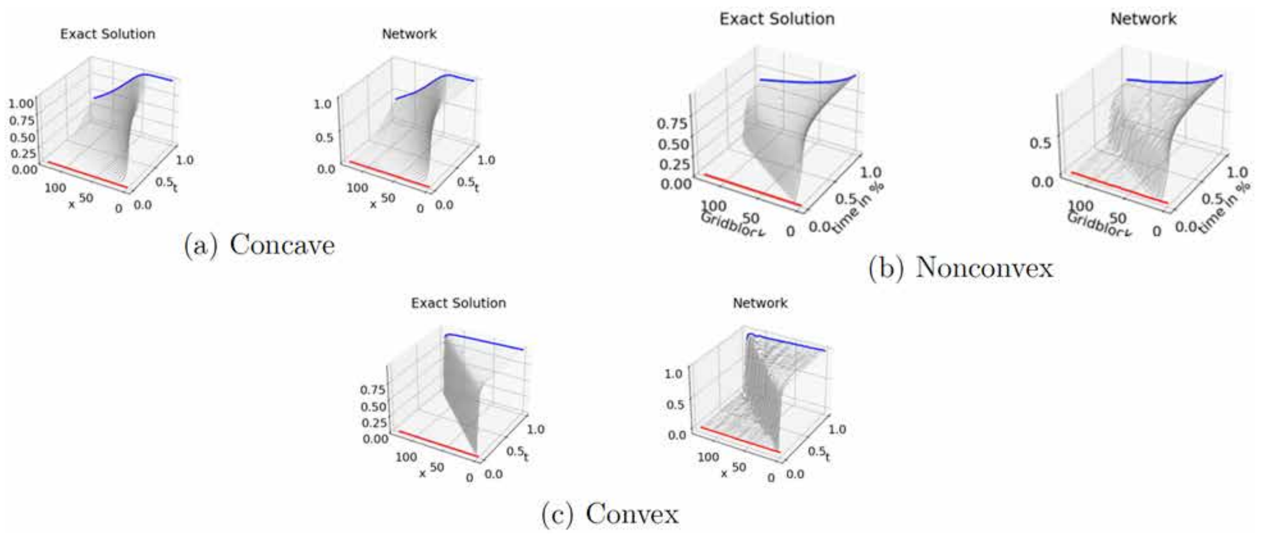

Figure 11—Three-dimensional plot showing the spatial dimension x, the temporal variable t, and the saturation in the vertical dimension. The left plot is the exact solution, and the right plot is the proxy

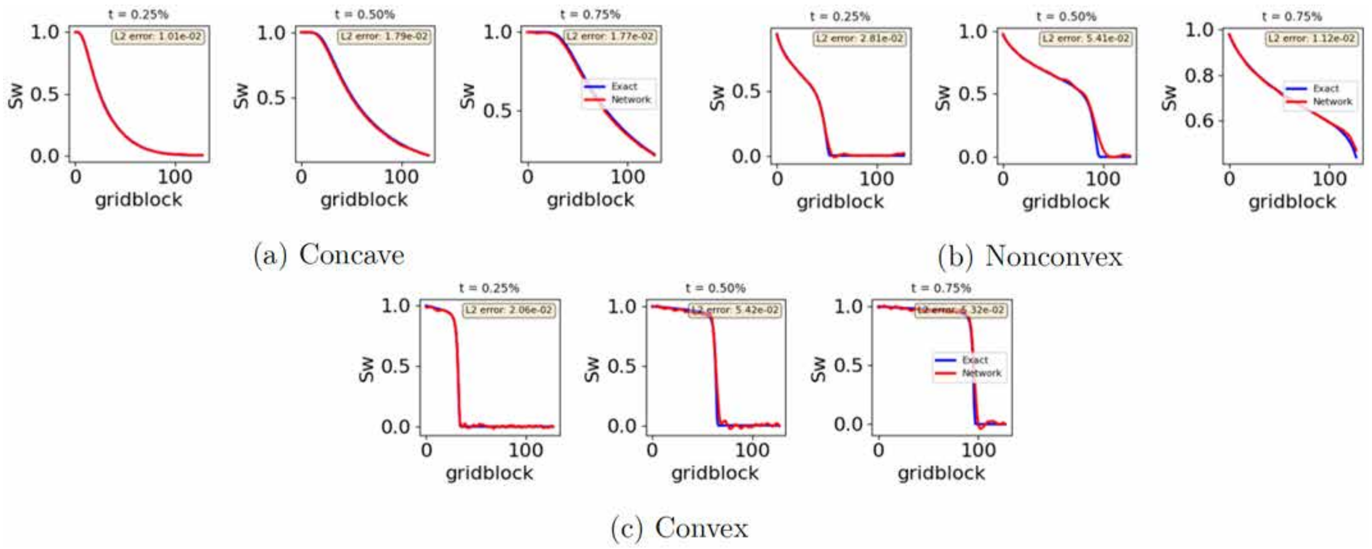

Figure 12—Snapshots at 25%, 50%, and 75% of the total time comparing the exact solution (blue curve) and the proxy solution (red curve)

For the convex case (figs. 11c and 12c), we continue to see the same oscillations as in the nonconvex case. All three snapshots show the same pattern in the smoother part of the plots. We infer that these oscillations come from the fact that the training process is trying to capture the discontinuities of the model by implicitly adding high-frequency modes to the proxy model. Nevertheless, as in the nonconvex case, the overall agreement between the exact and the predicted solutions is good.

Therefore, for the purposes of this section, we consider that the framework and methodology are tested for a case with no inputs. We acknowledge that we could improve the results by performing a hyperparameter optimization, for instance, selecting a judicious choice of latent space sizes.

Models with Control

Having proved the concept and checked the methodology for a case with no inputs, we drive our focus to the case of our interest: a system with inputs. However, including this additional term significantly increases the complexity of the Koopman operator. In the following sections, we review some solutions mentioned in section "Koopman with Control" and analyze the implications of the proxy architecture proposed here.

Koopman Operator with Input Term. Recalling eq. (16) and assuming only one input for simplicity, we have:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\phi(x)) = \mathbf{L}\phi(x) + \nabla_x\phi(x)g(x)u(t). \tag{31}$$

Note that the control u_k multiplies first a function $g(x)$, which can be analytically derived (Aziz and Settari 1979; Ertekin, Abou-kassem, and King 2001; Lie 2019). Then, it multiplies the gradient of the transformed value of x through the encoder with respect to inputs, which is relatively easy to obtain inside the TensorFlow framework. The AD algorithm (Baydin et al. 2018; Corliss, Hovland, and Naumann 2006) usually provides the derivatives of every function with respect to the parameters, in this case, the weights of the neural net; The methodology to compute the derivatives with respect to the inputs is analogous to the states, and thus, we leave it out of the discussions here due to the brevity of space.

To obtain more insights of the structure of eq. (31), we derive eq. (33). First, for simplicity, we assume that $\mathbf{B}(x) = \nabla_x\phi(x)g(x)$ and remove the time dependency notation, so that eq. (31) becomes:

$$\dot{\phi}(x) = \mathbf{L}\phi(x) + \mathbf{B}(x)u. \tag{32}$$

Then we reduces the dimension of ϕ by using a projection in the form of $\phi(x) = \mathbf{\Phi}z(x)$ with a residual projection of $\boldsymbol{\psi}$(see section "Physics-Based Method"):

$$\dot{z}(\mathbf{\Phi}z) = \mathbf{\Psi}^{\mathbf{T}}\mathbf{L\Phi}z(\mathbf{\Phi}z) + \mathbf{\Psi}^{\mathbf{T}}\mathbf{B}(\mathbf{\Phi}z)u. \tag{33}$$

The attractiveness of this formulation is that we keep the same number of parameters as before. Although the structure is more complex, the new term is obtained analytically or equal to the previous term. Figure 13 demonstrates the new architecture where the blue arrows indicate the information transfer between blocks.

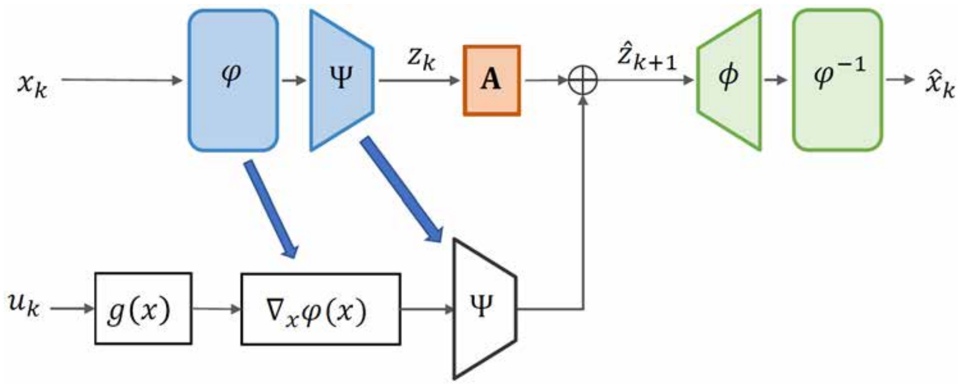

Figure 13—Koopman autoencoder with controls, where blue arrows represent the use of same elements on the state and also on the controls trajectory. *g(x)* is a function that can be obtained analytically (like the one used on a numerical reservoir simulator framework)

Intuitively, we expect this structure to yield results since the parameter number is the same as in the case with no control (fig. 3), but there is more information to the system. Since the Koopman block ($\varphi(x)$) has to perform the Koopman transformation and provide the derivatives for the control path. We see this process as a more constrained problem for neural net training, which in theory, would lead to results closer to physical ones. However, our tests did not confirm our expectations.

Bilinear Koopman. Architecture fig. 13 has faced severe issues with converging in our experiments during the training process. Either because the gradient vanishes or the gradient explodes. We believe the problem is due to instabilities on the $\nabla_x\phi(x)$ term calculation since the parameter of the Koopman block is initially randomly guessed. Thus, at the early stages of the training process, the computation of the control term is especially unstable.

This issue motivated us to look for the bilinearization of the Koopman operator (see section "Koopman with Control"). Below we give the general discrete bilinear equation:

$$z_{k+1}(x) = \mathbf{A}z_k(x) + \mathbf{N}\left(u_k \bigotimes z_k(x)\right) + \mathbf{B}u_k, \tag{34}$$

where the matrix $\mathbf{A} \in \mathbb{R}^{r\times r}$, r is the size of the latent space (a hyperparameter), $\mathbf{N} \in \mathrm{R}^{n\times mr}$ and $\mathbf{B} \in \mathbb{R}^{r\times m}$, and m is the number of inputs. We note that in several books (for instance, Elliott 2009; Brunton et al. 2021), the bilinear representation is giving but a summation (eq. (35)) term instead of the Kronecker product, but this is just a matter of representation since the final result is the same.

$$z_{k+1}(x) = \mathbf{A}z_k(x) + \sum_{i=1}^{m} \mathbf{N}_i z_k(x) u_{i_k} + \mathbf{B}u_k, \tag{35}$$

where $\mathbf{N}_\mathrm{i} \in \mathbb{R}^{n\times r}$ in such a way that the total matrix $\mathbf{N}$ is the horizontal concatenation of all matrices $N_i\Big|_{i=1}^{m}$, that is $\mathrm{N} = [\mathrm{N}_1\ \mathrm{N}_2 \cdots \mathrm{N}_m]$. In this work, however, we will rather use eq. (34) due to its easier application inside the autoencoder framework. Figure 14 shows the final representation of the proxy architecture.

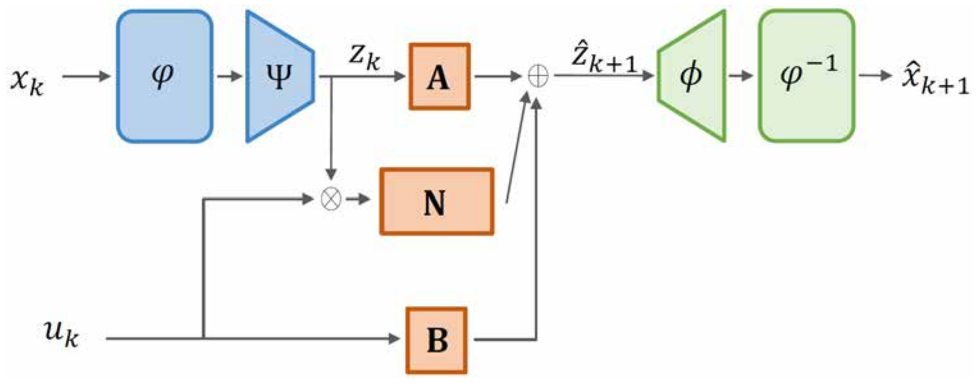

Figure 14—Bilinear Koopman autoencoder with controls, where it is possible to see that evolution in time and controls application happens on latent space

Stability. Next, we review the stability of bilinear systems. This analysis is crucial because it can bring more insights into constructing the proxy since an unstable system may blow up or not converge when an input is applied, no matter how small the magnitude of the input. This phenomenon does not hold in the case of reservoir simulation, where we know from real-world applications that a reservoir solution will not blow up. Therefore, if we guarantee that the proxy is stable, we avoid the risk of producing wrong and unrealistic predictions.

Stability analysis of linear dynamical systems is customarily studied considering the decomposition of the response as zero-input and zero-state responses (Chen 1998). Although this decomposition is usually not applicable for nonlinear systems, bilinear systems are transition systems between these two extremes and thus allow us to perform this analysis.

Zero-state response considers the case when there is no input, that is, $u_k = 0$, and we refer to this stability as internal (or marginal and asymptotic) stability. On the other hand, zero-state responses consider the case when output will be excited exclusively by the input, that is, $z(0) = 0$, and we refer to this stability as BIBO (bounded-input bounded-output) stability.

Internal stability of the system eq. (35) consists in the analysis of the equation $z_{k+1}(x) = \mathbf{A}z_k(x)$, which is stable if, and only if, all the eigenvalues of A have to lie inside the unit circle of the complex plane (Chen 1998). If this necessary condition is met for every initial condition, the system response approaches zero when time goes to infinity. This conclusion will then be used in the construction of the proxy.

The second type of stability, BIBO, is not entirely defined for bilinear systems. The works of Bose and Chen (1995) and Zhang et al. (2003) have only provided sufficient conditions which might be too conservative. We outline the main result in the following theorem.

Theorem 1 (BIBO stability of bilinear system (Zhang et al. 2003)). *For a bilinear system, any bounded input by a positive number M ($\max_j(u_j(k)) \leq M$) provides a bounded output if:*

i. *A is stable;*
ii. $M < \frac{1-\rho}{\Gamma\alpha}$, *for* $\Gamma = \sum_{j=1}^{m} \| N_j \|$, *Where N_j is defined using the formalism of* eq. (35). *And* $\rho \in [0,1)$ *and* $\alpha > 0$ *are such that* $\|A^i\| \leq \alpha\rho^i$ *holds.*

However, besides being only a sufficient and conservative condition, the BIBO stability check is somehow complicated to be implemented as a loss function. Thus, we decided not to use it in our development and left it as a suggestion for future work.

New Architecture: Avoiding Error by Randomness in Initial Guesses. As in section "Weight Initialization", the initialization of the matrices **A, N**, and **B** should be cautiously performed. We adopt the same strategy as before: run the training with only the autoencoder part without any dynamics for a few epochs (3 or 5), collect the snapshots in the latent dimension, and perform Bilinear Dynamic Decomposition (Goldschmidt et al. 2021). However, we modify the algorithm to take into consideration the linear control matrix **B** since the original proposition has considered a system like $z_{k+1}(x) = \mathbf{A}z_k(x) + \mathbf{N}(u_k \otimes z_k(x))$ (compare with eq. (34)). For completeness, we present the modified version below:

Algorithm 1 Modified Bilinear Dynamic Mode Decomposition

```
1: $\Xi \leftarrow \begin{bmatrix} X^T & (U \odot X)^T & U^T \end{bmatrix}^T$
2: $\tilde{U}, \tilde{\Sigma}, \tilde{V} \leftarrow SVD(\Xi, \tilde{r})$          ▷ Truncated $\tilde{r}$-rank SVD of $\Xi$
3: $\begin{bmatrix} \tilde{U}_A & \tilde{U}_N & \tilde{U}_B \end{bmatrix} \leftarrow \tilde{U}$          ▷ Decompose $\tilde{U}$ according to $A$, $N$, and $B$
4: $\tilde{A} \leftarrow X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_A^{\dagger}$          ▷ Estimate for A
5: $\tilde{N} \leftarrow X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_N^{\dagger}$          ▷ Estimate for N
6: $\tilde{B} \leftarrow X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_B^{\dagger}$          ▷ Estimate for B
7: $\hat{U}, \hat{\Sigma}, \hat{V} \leftarrow SVD(X', \hat{r})$          ▷ Truncated $\hat{r}$-rank SVD of $X'$
8: $\hat{A} \leftarrow \hat{U}^{\dagger}\tilde{A}\hat{U}$          ▷ Low-rank approximation for $A$
9: $W, \Lambda \leftarrow \text{EIG}(\hat{A})$          ▷ Eigendecomposition of $\hat{A}$
10: $\Phi \leftarrow \tilde{A}\hat{U}W$          ▷ DMD modes for $A$
```

However, for the current purpose, we only go until step 6 of the algorithm since we are only interested in the estimate of matrices **A, N**, and **B**. There is no need to perform a low-rank approximation because the Projection block (the inner autoencoder) already takes care of this function, much less identifying the DMD modes. Figure 15 depicts the process in a very similar way to fig. 8, where we run only the autoencoder for a few epochs, perform the bilinear version of the DMD algorithm to initialize the dynamical matrices, and finally perform the complete training for several epochs.

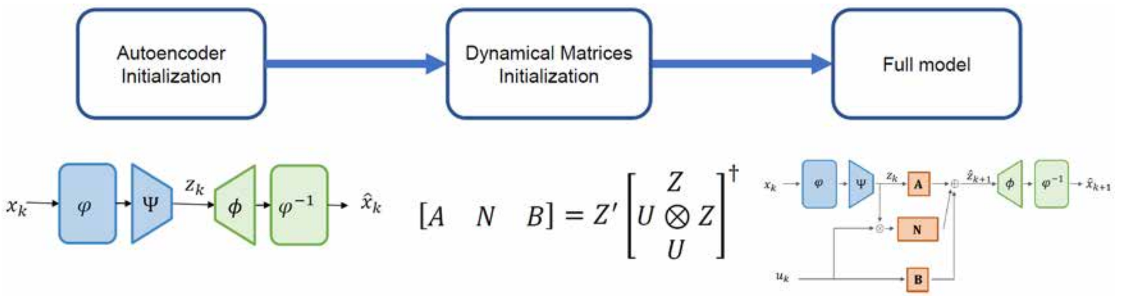
Autoencoder Initialization
Dynamical Matrices Initialization
Full model
$[A \quad N \quad B] = Z' \begin{bmatrix} Z \\ U \otimes Z \\ U \end{bmatrix}^{\dagger}$

Figure 15—Step-wise training for controlled proxies: First, the autoencoder is initialized by training it for a few epochs, like in fig fig. 8. Second, we collect snapshots of the reduced linearized variable and controls and perform biDMD to initialize the dynamical matrices A, B and N. Lately, using the initialized autoencoder components and matrices A, B and N estimated by biDMD, the full model training is performed

This strategy proved essential to the neural net's training process in our experiments. Without it, the horizon for training (fig. 5a) was significantly smaller, which caused the results to be consistently worse.

Additional Losses Function. Since the architecture has changed to a more complex structure, one also adds more loss functions to guide the backpropagation algorithm of the neural network training to more reasonable results. Specifically, we add two more losses from the physics of the problem we are dealing with. The first relates to how the outputs are computed in the reservoir simulation. The standard approach is through the definition of a well model and the Peaceman's equation (eq. (36))

$$q_\alpha = J\lambda_\alpha(s_w)\left(p_{grid} - bhp\right), \tag{36}$$

where J is the productivity index (dependent on the geometry of the well and the static property of the reservoir), λ_α is mobility at the connection of the phase α (which is dependent on the saturation by the dynamic properties of the reservoir fluids, i.e., $\lambda_\alpha = \frac{kr_\alpha(s_w)}{\mu_\alpha(T)}$), p_{grid} is pressure in the perforated grid block, and v is bottom-hole pressure of the well, more details can be found in any reservoir textbook such as Aziz and Settari 1979; Ertekin, Abou-kassem, and King 2001; Lie 2019.

The second additional loss that we include in the training process accounts for the zero-input stability of the real system. Bellow, we introduce these new losses with specific details:

- **Loss 6: Well State Loss**: since our final goal is well control optimization, we know from the Peaceman's equation that the states (pressures and saturations) of the gridblock containing a well are crucial to the output solution. Therefore, we add extra weight to the mismatch of pressures of these gridblocks by introducing a new loss function:

$$\mathcal{L}_6 = \frac{1}{N_{\text{wells}}} \sum_{i=1}^{N_{\text{wells}}} \frac{1}{M+1} \sum_{k=0}^{M} \frac{\| x_i^k - \hat{x}_i^k \|_2^2}{\| x_i^k \|_2^2}. \tag{37}$$

- **Loss 7: Stability loss:** Also, we know from the physics of the reservoir that the system is stable. That is, the pressure of the reservoir will increase unless an external factor adds energy to the system, for instance, injecting wells or aquifers. This fact and based on the internal stability analysis of section "Stability" motivated us to insert the following loss function:

$$\mathcal{L}_7 = \sum_{i=1}^{N_\lambda} \max(|\lambda_i|, 1), \tag{38}$$

where N_λ is the number of eigenvalues with an absolute value bigger than one, the implementation of this loss function made the system internally stable (fig. 16).

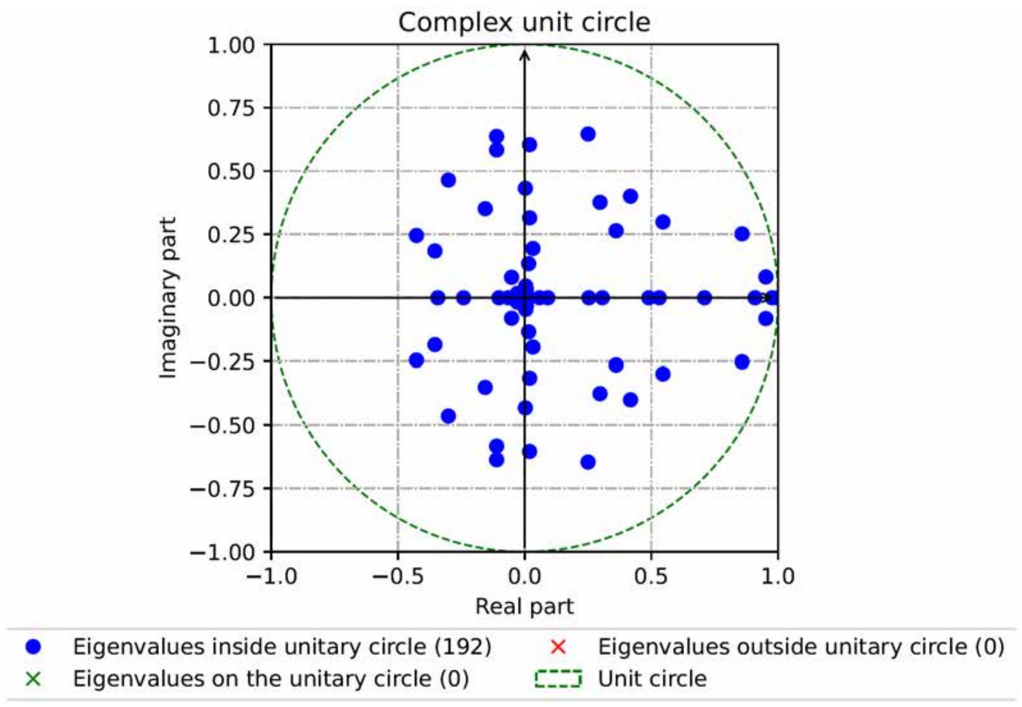

Figure 16—Eigenvalues of the matrix A. This is an example after training is complete. Note that all eigenvalues remain inside the unity circle respecting the zero-input stability condition

Losses in Training vs. Validation Process. There are, in total, seven different losses the proxy needs to equate to form the total cost function. The cost function is what the neural network training process - a combination of optimization and automatic differentiation - will train to minimize.

In summary, these loss functions are: (1) Autoencoder loss, (2) Koopman loss, (3) Projection loss, (4) Prediction in "x" loss, (5) Prediction in "z" loss, (6) Well States loss, (7) Stability loss, plus the regularization term. Thus the total loss function is:

$$\mathcal{L}_{train} = \sum_{i=1}^{9} \alpha_i \mathcal{L}_i + \mathcal{R}. \tag{39}$$

As mentioned in section "Definition of the Loss Functions", how to set a_i is still an open problem. Currently, we keep all the α's equal to one as it performs best. However, more work needs to be done to optimally pick α's.

During the training process, we are focused on reducing the values of all losses (1-7) as much as possible. However, during the validation process, we only check loss (4), the prediction in "x", which controls the prediction quality of pressures and saturations over time. The reason is that, at the validation step, we check the errors to avoid over-fitting and select the best model (in the sense that we chose the model at the epoch with the best validation error and not the last epoch). Therefore, we try to guarantee that both of these objectives are optimal for the most important loss of the proxy. In summary, the checking loss of the validation step is:

$$\mathcal{L}_{validation} = \mathcal{L}_4. \tag{40}$$

Application to Two-phase (Oil and Water) Two-Dimensional Reservoir

Increasing the physical model's complexity, from the BL example, we move to a two-dimensional reservoir case with two phases (oil and water) with a cartesian grid with 60 × 60 × 1 gridblocks and a fixed permeability field (fig. 17).

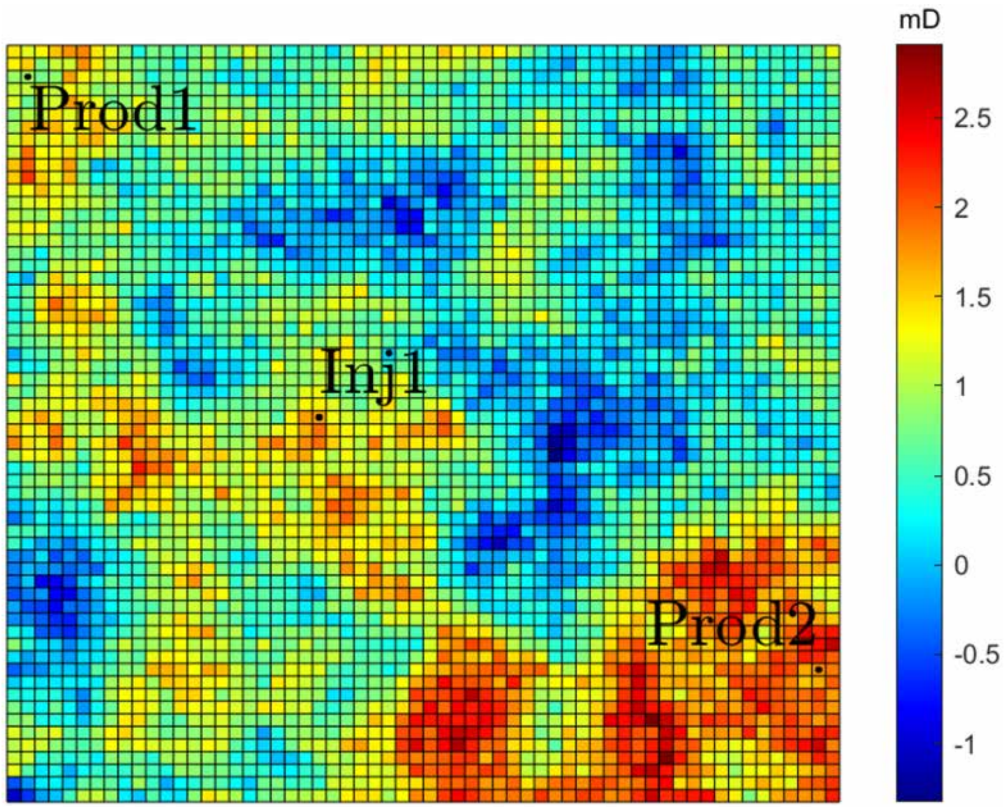

Figure 17—Permeability field [milli-Darcy] (logarithmic scale). Note the position of the wells along the off-diagonal, with producers at the extremes and the injector at the middle of the reservoir.

The reservoir has two producers and one injector well along the diagonal of the grid. The producer wells are controlled by bottom hole pressure (BHP), and the injector is governed by the water injection rate. This model runs through 1280 days, changing controls every 120 days. Table 1 summarizes the properties of this reservoir used in all simulations.

Additionally, the relative permeability is given by the Corey model, a power-law relationship given by

Table 1—Reservoir model properties the following equation:

Property	Value	Unit
Rock Compressibility	1E — 6	$psia^{-1}$
Oil Formation Volume Factor	1.1	res-bbl/std-bbl
Water Formation Volume Factor	1.0	res-bbl/std-bbl
Oil Viscosity	1.2	cP
Water Viscosity	0.6	cP
Oil Density	600	kg/m^3
Water Density	1000	kg/m^3
Porosity	0.5	
Initial Pressure	100	barsa

Property	Value	Unit
Initial Water Saturation	0.0	

$$s^* = \frac{S_w - S_{wc}}{1 - S_{wc} - S_{or}} \tag{41}$$

$$kr_w = (s^*)^{n_w} k_w^0 \tag{42}$$

$$kr_o = (1 - s^*)^{n_0} k_o^0 \tag{43}$$

where the exponents $n_{w,o}$ are monomials describing relative permeability, and the constants $k_{w,o}^0$ are the end-point scaling for each phase. Table 2 summarizes the properties used for the relative permeability calculation.

Table 2—Relative Permeability Property

Property	Value (dimensionless)
Connate Water Saturation (S_{wc})	0.2
Residual Oil Saturation (S_{or})	0.2
k_{rw}^0	1.0
k_{ro}^0	1.0
n_w	2
n_o	2

Data Set

As in most machine learning algorithms and any system identification process, we need a "rich" set of data to train the model. Unfortunately, there is neither a definitive answer to the number of necessary data nor the types of inputs.

To clarify, data is comprised of snapshots of the states (pressures and saturation) generated by a High Fidelity Simulator (HFS), which in our study is MRST (Lie 2019). We call one simulation a "trajectory," that is, the dynamical evolution of the states over time. A collection of trajectories forms a data set.

The objective of a proxy is to reproduce output from an HFS, which is usually unknown to the training process. Thus, we need to provide a set of trajectories that excite the dynamical system in several different ways so that the training process would be able to identify all the frequency responses of the original model. The previous description is the informal definition of the "rich" concept.

Inspired by system identification methods (Katayama 2005), our strategy to create a "rich" dataset is to run several HF (high fidelity) simulations using four types of inputs: pseudo-random binary (PRBS), steps, sinusoidal, ad "smooth steps (fig. 18). We set the same range of minimum and maximum change for all types. PRBS signals (fig. 18a) have a 50% probability of staying either at the maximum or the minimum value. Steps signals (fig. 18b) are free to go to any value between the range at each control change. Sinusoidal signals (fig. 18c) obey a sinus shape but with variable frequency. Moreover, smooth steps (fig. 18d) behave similarly to step changes, but with the limitation that each change cannot be greater than 20% of the current value.

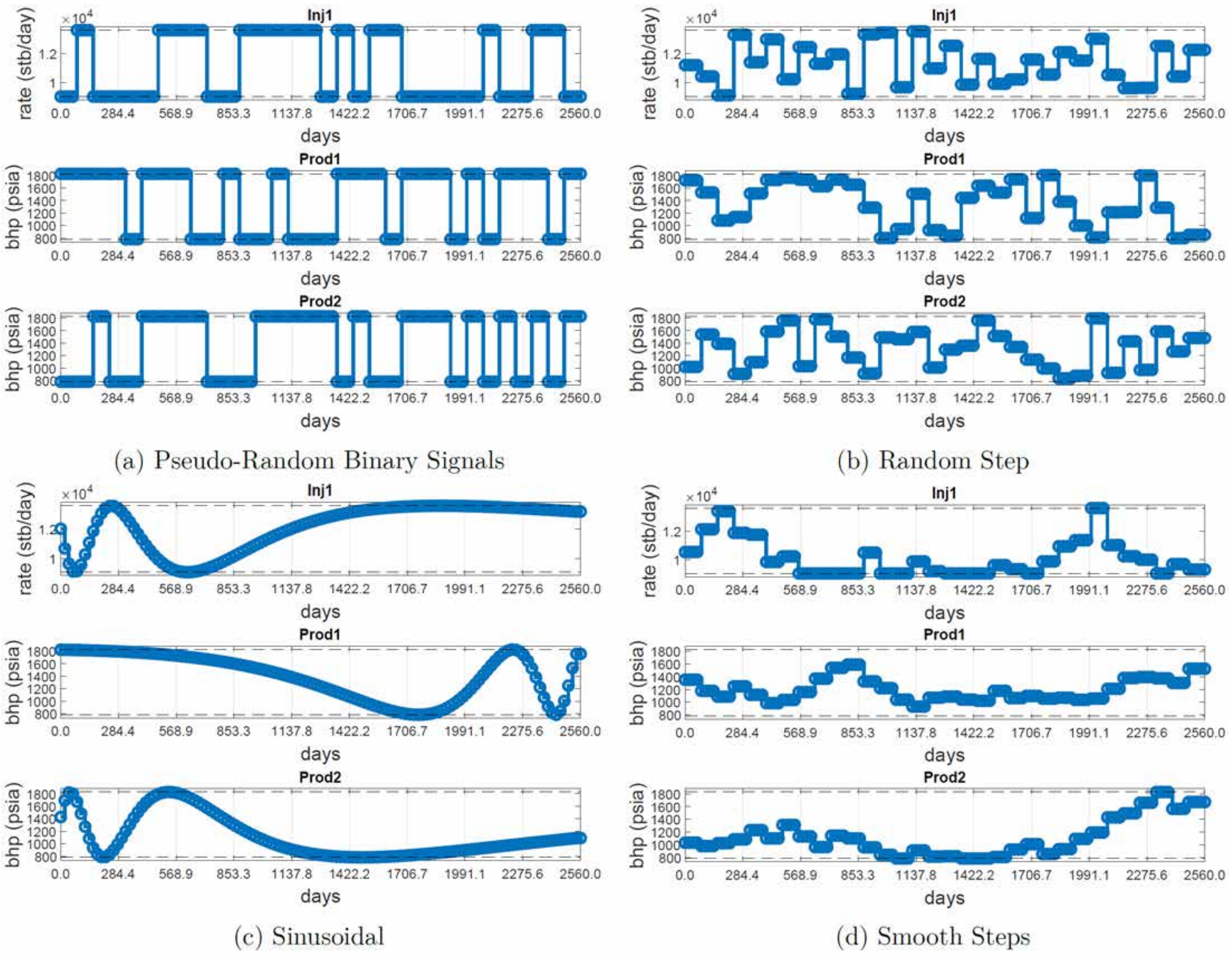

Figure 18—Randomly picked examples for each type of controls types applied for each well. PRBS signals that can randomly go between the minimum and maximum. Step signals that can change to any value within the range. Sinusoidal types of input where the randomness comes from the frequency. Lastly, a very similar signal to step one with the difference that the change is restricted to a certain threshold depending on the current value, thus having a smoother behavior.

Therefore, the training data set consists of 2,560 trajectories, each with 54 equally spaced time steps of 20 days. There is a mix of inputs (PRBS, Sinus, Steps, Smooth steps), and each input signal is randomly created to avoid repetitions. The validation data set have the same structure but with 640 trajectories.

Hyper-parameters Tuning

The most time-consuming part of deep learning techniques is hyper-parameter tuning, which are parameters that need to be set before the training itself. Examples of hyper-parameters are training batch size, number of epochs, learning rate, size of the latent space, and neural network model. On the other hand, the training procedure defines the parameters and is relatively straightforward from a user's point of view. Examples of parameters are the neural net weights and the dynamical matrices **A, B**, and **N**.

Our strategy is to isolate each hyper-parameter and test some combination of pre-defined values. In our experiments, the most crucial ones are the learning rate and batch size of the stochastic gradient descent algorithm (ADAM solver) and the size of the latent space (projection dimensionality reduction). Our test indicates that an initial value of 6.79E — 5 for the learning rate scheduler (section "Learning Rate

Schedule"), latent space of 192, and batch size of 32 for fully connected layers (8 for CNNs) generally give the best results.

Results

This section presents the prediction results of the proxy. After adequate training, we select one model based on the validation error. Then, we test the proxy in a test dataset to see how the proxy would behave in a real-case scenario, where the test data set contains trajectories that were never seen by the proxy.

Figure 19 shows the results for a randomly chosen trajectory for four different times: 25%, 50%, 75%, and 100% of the total prediction time (1280 days). The third column of results for each plot shows the absolute error for saturation and the relative error for pressure. Note the fixed limits of the colobar. One can see from all four plots (figs. 19a to 19d) that the error in saturation depicts a similar trend. The biggest error is always at the waterfront border, which is intuitively what we were expecting since this is the region where all the dynamics occur for the saturation. Nevertheless, the visual error is small and acceptable since most of the plot tends to be deep blue, and even at the borders, the error tends to be light blue. The maximum error for the saturation of 0.25 must have happened at one specific time and probably at the waterfront's edge.

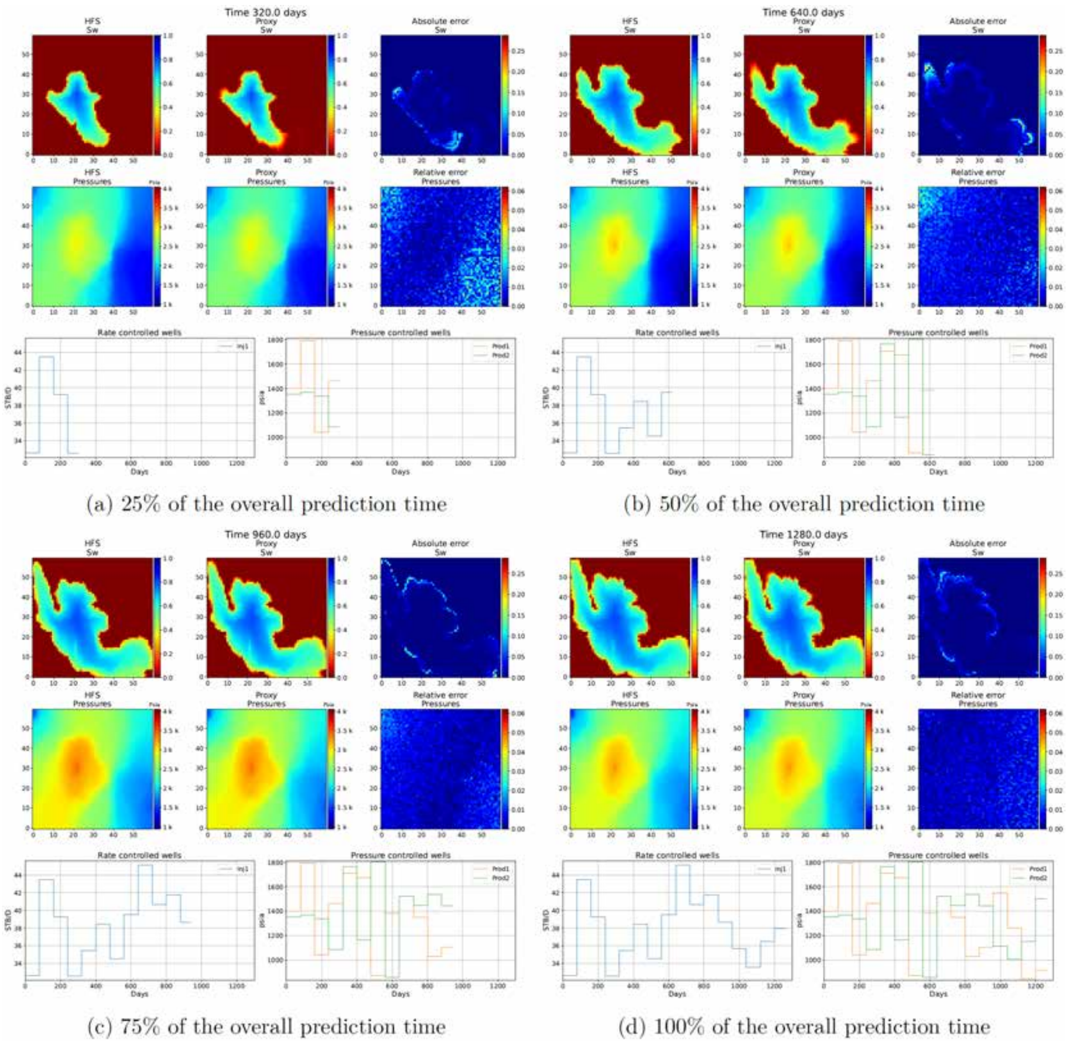

Figure 19—Reservoir States Evolution for one trajectory: each plot's first and second rows shows the pressures and the saturation, respectively. For each plot, the first column shows the results from a High Fidelity Simulator (HFS), the second column the results from the proxy, and the third column show the error between HFS and Proxy results. The last row depicts the control applied in the wells that created these states dynamics

The pressure maps do not have a clear evolution as the saturation maps. But we can see how the well locations dictate the disposition, with the low-pressure areas related to the producer wells and the high-pressure regions of the injector well position. The error map depicts a more pixelated behavior, with areas of light blue around the producer wells. In all cases, the maximum error is at 6%, which is not presented in the frames of fig. 19, with a maximum error of around 2%.

This outstanding result was only for one trajectory, but we also want to check how the proxy performs for various controls. We use a test data size with 320 trajectories and create a metric (M) to measure how good each trajectory is predicted (eq. (44)).

$$\mathcal{M}_{pressure} = \frac{1}{n_T} \| \frac{p - \hat{p}}{p} \|_1 \times 100 \tag{44a}$$

$$\mathcal{M}_{saturation} = \frac{1}{n_T} \| s_w - \hat{s}_w \|_1 \times 100 \tag{44b}$$

where p, s_w are the "real" values of pressure and water saturation from an HFS, $\hat{p}$, $\hat{s}_w$ are the values predicted by the proxy, n_T is the size of the Test data set (320 in our case). We chose the l_1 instead of the l_2 since the first is more robust to outliers, in the sense that the l_2 norm square the error, putting extra weight on outliers. We note that eq. (44a) is a Mean Absolute Percentage Error (MAPE), and eq. (44b) is a Mean Absolute Error (MAE) multiplied by 100.

Figure 20 shows the box plots for both states for all trajectories. Note that the pressure error according to the metric eq. (44a) is similar to those observed in fig. 19, and all the values have a low MAPE value. On the other hand, the saturation box plot depicts considerably small values. By comparing with fig. 19, we note that most of the value in the Cartesian grid is close to zero (deep blue color), and only at a few grids does the error increase (at the waterfront). This fact explains the low value in the box plot since the MAE is effectively a means of all the errors. Both box plots show that the proxy does an excellent job of predicting the states.

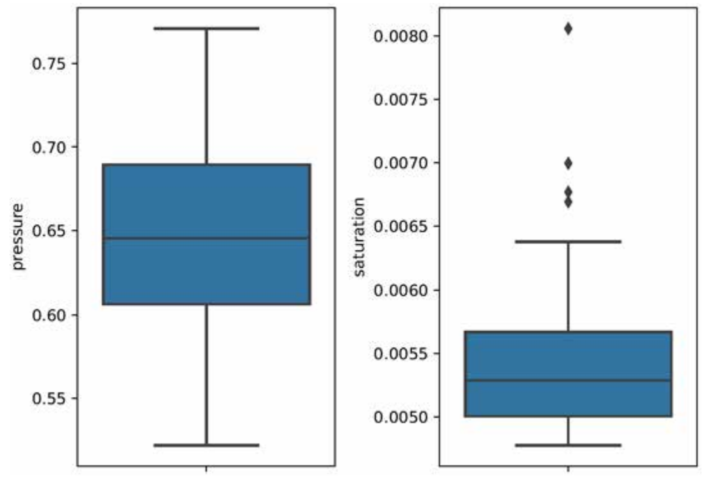

Figure 20—Box plot of the set errors using the metrics defined in eq. (44) for pressure (left) and saturations (right). The bottom and top of the blue region of each box represent 25th and 75th-percentile errors. The middle line inside is the median of the distribution. And the prolongations describe the region where an outside value is an outlier.

In addition, we would like to check the results at well locations since this is the final goal of any reservoir simulation. We can see in fig. 21 the predictions at well producer has an excellent agreement with the exact solution. Note that the forecast of the water breakthrough is entirely accurate, something that has been a challenge for the other model order reduction techniques such as POD, POD-DEIM, and TPWL.

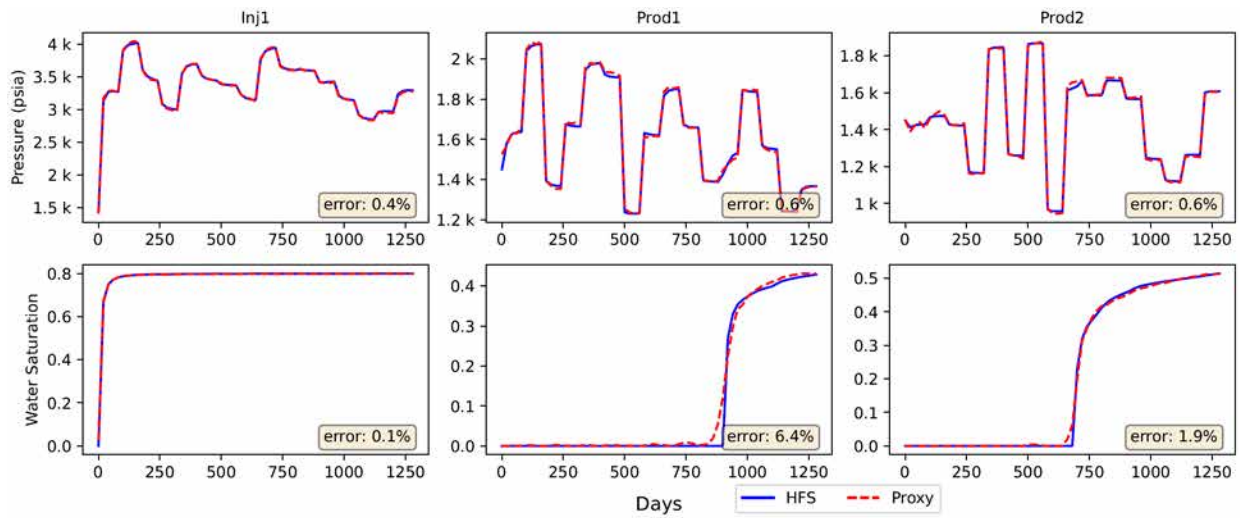

Figure 21—Pressure and Saturation at wells locations. The blue line is the high-fidelity solution obtained from MRST, and the red line is the result of the proposed proxy. And the error value follows the metrics defined by eq. (44).

Speed-ups

Table 3 shows the time comparison between the simulations in three environments, HFS in MRST (Lie 2019), HFS in a commercial reservoir simulator (IMEX - CMG 2019), and the proposed proxy. First, note that the proxy performs considerably faster than MRST and 3.3 faster than IMEX, but as a word of caution, these comparisons are not entirely fair, especially to commercial simulators, which are highly efficient in terms of programming. We expect the proxy to perform better if a better programming paradigm is applied.

Table 3—Time comparison between full order simulation and the proxy model. The speed-up of the proposed proxy model reaches six times for running one simulation at a time. In case we need to run 320 simulations, the speed up increases to more than 20 times when compared with a commercial numerical reservoir simulator

Simulations	MRST - Matlab	IMEX - 1 CPU	Proxy
1	45 sec	2.39 sec	0.36 sec
320	4 hours	764.8 sec	38 sec

More importantly, we highlight the outstanding scalability of the proxy. Note that a standard reservoir simulation will take a long time to run 320 forward runs in a sequential simulation. The proxies work differently: since they have no Newton-Raphson iteration, just matrix multiplications, increasing one more simulation is equivalent to increasing one column to multiplication in a computer environment. This is something that scientific languages (such as Python/Numpy, Matlab, Fortran, and C++) are very efficient in handling.

Conclusion

The scope of this study was to develop a hybrid proxy model ideal for reservoir control optimizations, which accurately predicts the nonlinearities (e.g., water breakthrough time). To accomplish this, we used two main theoretical drives: the concept of the Koopman operator and Projections. The first states that we can find a new set of coordinates in which nonlinear dynamics behaves linearly, and the second states that we can find a reduced projection of the original system with lower dimensions. In our case, the tool to implement these concepts is deep learning neural networks.

We developed a novel framework combining the classic neural net architecture of auto-encoder, dynamic system aspects, and several loss functions, including some derived from first principles. The reason for doing this is to ultimately guide the training process to achieve reasonable and physical results. Traditional neural net applications suffer from extrapolation problems when the prediction is outside the range of the training data set. The key value proposition of the above methodology is to accelerate High Fidelity Simulations (HFS) after initial training, but without losing model fidelity while working outside the range of initial training data.

We first benchmarked our approach with Buckley-Leverett 1D model as a state space equation with no external inputs. The results proved that it is possible to effectively linearize the dynamics using only data achieving similar results to the analytical Cole-Hopf transformation.

The second problem is more complex a waterflooding study in a two-dimensional two-phase (oil and water) reservoir subject to a waterflooding plan with three wells including one injector and two producers. The external inputs for the Koopman operator are addressed by designing a bilinear manifold in the new set of coordinates found by the Deep Koopman operator (deep neural network). This new architecture can accurately predict the system states (pressure and saturation) with considerable speed-ups (125 times for a single run) and accuracy of 1-3 percent error on the pressure and saturation predictions. It is worthwhile noting that this method is a non-intrusive data-driven method since it does not need access to the reservoir simulation internal structure.

An important observation to realize is the immense scalability power of the proxies. In contrast to traditional simulators that solve the nonlinear equation using Newton-Raphson (NR) iterations and are thus restricted to one forward run per time, the proposed proxy can easily make several simulations simultaneously. Since the proxy does not solve any iteration, the way the proxy deals with multiple simulations is analogous to tensor multiplication. Different conditions are treated as a new dimension transforming the matrices into tensors. This property makes computations much more efficient than solving the equations sequentially. Although we have given a step forward in identifying a robust proxy (and developed methodology) for reservoir simulation, many open problems still need to be solved to make this work viable in all cases. The main assumption of relying on a well-calibrated HFS is not restrictive but could be relaxed using other frameworks ranging from purely data-driven to grid network-based models. Also, we do not know how the stochastic gradient descent behaves inside the neural net, but we see it works empirically. Finally, we believe this paper confirms that deep learning techniques are powerful tools that enable studies considered impossible in the recent past, but care must be taken when making generalizations.

Acknowledgements

The authors acknowledge Xecta Digital Labs for allowing the publication of this paper's results and partially funding this project.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., et al 2015. Tensor-Flow: Large-scale Machine Learning on Heterogeneous Systems.

Antoulas, A. C. 2005. Approximation of Large-scale Dynamical Systems (advances in Design and Control). *Society for Industrial / Applied Mathematics*.

Aziz, K. and Settari, A. 1979. Petroleum Reservoir Simulation. Applied Science Publishers.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. 2018. Automatic Differentiation in Machine Learning: A Survey. *Journal of Marchine Learning Research* **18**:1–43.

Belbute-peres, F. D. A., Economon, T. D., and Kolter, J. Z. 2020. Combining Differentiable Pde Solvers and Graph Neural Networks for Fluid Flow Prediction. https://doi.org/10.48550/ARXIV.2007.04439.

Benner, P. and FaSSbender, H. 2013. Model Order Reduction: Techniques and Tools. *In: Encyclopedia of Systems and Control*, edited by J. Baillieul and T. Samad, 1–10. London: Springer London. https://doi.org/10.1007/978-1-4471-5102-9_142-1.

Benner, P., Gugercin, S., and Willcox, K. 2015. A Survey of Projection-based Model Reduction Methods for Parametric Dynamical Systems. *SIAM Review* **57** (4): 483–531. https://doi.org/10.1137/130932715.

Bose, T. and Chen, M. Q. 1995. BIBO Stability of the Discrete Bilinear System [in en]. *Digital Signal Processing* **5** (3): 160–166. https://doi.org/10.1006/dspr.1995.1016.

Brunton, S. L., Budii, M., Kaiser, E., and Kutz, J. N. 2021. Modern Koopman Theory for Dynamical Systems.

Brunton, S. L. and Kutz, J. N. 2019. Data-driven Science and Engineering. Cambridge University Press. https://doi.org/10.1017/9781108380690.

Cardoso, M. A. and Durlofsky, L. J. 2010. Linearized Reduced-order Models for Subsurface Flow Simulation. *Journal of Computational Physics* **229** (3): 681–700. https://doi.org/10.1016/j.jcp.2009.10.004.

Carvajal, G., Maucec, M., and Cullick, S. 2017. Intelligent Digital Oil and Gas Fields: Engineering Concepts, Models, and Implementation. 420. Elsevier Science & Technology Books. https://doi.org/10.1016/c2015-0-02216-x.

Chamberlain, B. P., Rowbottom, J., Gorinova, M., Webb, S., Rossi, E., and Bronstein, M. M. 2021. Grand: Graph Neural Diffusion. https://doi.org/10.48550/ARXIV.2106.10934.

Chaturantabut, S. and Sorensen, D. C. 2010. Nonlinear Model Reduction Via Discrete Empirical Interpolation. *SIAM Journal on Scientific Computing* **32** (5): 2737–2764. https://doi.org/10.1137/090766498.

Chen, C.-t. 1998. Linear System Theory and Design. The Oxford Series in Electrical and Computer Engineering. Oxford University Press.

Chen, Z., Huan, G., and Ma, Y. 2006. Computational Methods for Multiphase Flows in Porous Media. *Society for Industrial & Applied Mathematics,U.S.*, 2006.

CMG. 2019. Imex, Black Oil & Unconventional Simulator.

Corliss, G., Hovland, P., and Naumann, U. 2006. Automatic Differentiation: Applications, Theory, and Implementations. Edited by M. Bücker, G. Corliss, U. Naumann, P. Hovland, and B. Norris. 388. Springer. https://doi.org/10.1007/3-540-28438-9.

Coutinho, E. J. R., Dall'Aqua, M., and Gildin, E. 2021. Physics-aware Deep-learning-based Proxy Reservoir Simulation Model Equipped with State and Well Output Prediction. *Frontiers in Applied Mathematics and Statistics* **7**:49. https://doi.org/10.3389/fams.2021.651178.

Eliasof, M., Haber, E., and Treister, E. 2021. Pde-gcn: Novel Architectures for Graph Neural Networks Motivated by Partial Differential Equations. https://doi.org/10.48550/ARXIV.2108.01938.

Elliott, D. 2009. Bilinear Control Systems : Matrices in Action. Dordrecht Netherlands New York: Springer.

Ertekin, T., Abou-kassem, J. H., and King, G. R. 2001. Basic Applied Reservoir Simulation. Society of Petroleum Engineers.

Florez, H. and Gildin, E. 2019. Global/local Model Order Reduction in Coupled Flow and Linear Thermal-poroelasticity. *Computational Geosciences* (2019). https://doi.org/10.1007/s10596-019-09834-7.

Géron, A. 2017. Hands-on Machine Learning with Scikit-learn and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.

Ghasemi, M. and Gildin, E. 2015. Model Order Reduction in Porous Media Flow Simulation Using Quadratic Bilinear Formulation. *Computational Geosciences* **20** (3): 723–735. https://doi.org/10.1007/s10596-015-9529-0.

Ghommem, M., Gildin, E., and Ghasemi, M. 2016. Complexity Reduction of Multiphase Flows in Heterogeneous Porous Media. *SPE Journal* **21** (01). https://doi.org/10.2118/167295-PA.

Gildin, E., Ghasemi, M., Romanovskay, A., and Efendiev, Y. 2013. Nonlinear Complexity Reduction for Fast Simulation of Flow in Heterogeneous Porous Media. Presented at the SPE Reservoir Simulation Symposium. Society of Petroleum Engineers. https://doi.org/10.2118/163618-ms.

Gin, C., Lusch, B., Brunton, S. L., and Kutz, J. N. 2021. Deep Learning Models for Global Coordinate Transformations That Linearise Pdes. *European Journal of Applied Mathematics* **32** (3): 515–539. https://doi.org/10.1017/S0956792520000327.

Goldschmidt, A., Kaiser, E., Dubois, J. L., Brunton, S. L., and Kutz, J. N. 2021. Bilinear Dynamic Mode Decomposition for Quantum Control. *New Journal of Physics* **23** (3): 033035. https://doi.org/10.1088/1367-2630/abe972.

Goodfellow, I., Bengio, Y., and Courville, A. 2016. Deep Learning. MIT Press.

Goswami, D. and Paley, D. A. 2021. Bilinearization, Reachability, and Optimal Control of Control-affine Nonlinear Systems: A Koopman Spectral Approach. *IEEE Transactions on Automatic Control*, 1–1. https://doi.org/10.1109/TAC.2021.3088802.

Han, Y., Hao, W., and Vaidya, U. 2020. Deep Learning of Koopman Representation for Control. Presented at the 2020 59th IEEE Conference on Decision and Control (CDC), 1890-1895. IEEE. https://doi.org/10.1109/cdc42340.2020.9304238.

Hastie, T., Tibshirani, R., and Friedman, J. 2009. The Elements of Statistical Learning. Springer-Verlag New York Inc., 2009.

Hedengren, J. D., Udy, J., Hansen, B., Maddux, S., Petersen, D., Heilner, S., Stevens, K., and Lignell, D. 2017. Review of Field Development Optimization of Waterflooding, Eor, and Well Placement Focusing on History Matching and Optimization Algorithms. *Processes* **5** (4): 34. https://doi.org/10.3390/pr5030034.

Heijn, T., Markovinovic, R., and Jansen, J. D. 2004. Generation of Low-order Reservoir Models Using System-theoretical Concepts. *SPE Journal* **9** (02): 202–218. https://doi.org/10.2118/88361-PA.

Hou, J., Zhou, K., Zhang, X.-s., Kang, X.-d., and Xie, H. 2015. A Review of Closed-loop Reservoir Management. *Petroleum Science* **12**, no. 1 (2015): 114 –128. https://doi.org/10.1007/s12182-014-0005-6.

Izenman, A. J. 2008. Modern Multivariate Statistical Techniques. Springer New York. https://doi.org/10.1007/978-0-387-78189-1.

James, G., Witten, D., Hastie, T., and Tibshirani, R. 2021. An Introduction to Statistical Learning. Springer US, 2021.

Jansen, J. D. 2013. A Systems Description of Flow through Porous Media. Springer International Publishing, 2013.

Jansen, J. D. and Durlofsky, L. J. 2017. Use of Reduced-order Models in Well Control Optimization. *Optimization and Engineering* **18** (1): 105–132. https://doi.org/10.1007/s11081-016-9313-6.

Jin, Z. L., Liu, Y., and Durlofsky, L. J. 2020. Deep-learning-based Surrogate Model for Reservoir Simulation with Time-varying Well Controls. *Journal of Petroleum Science and Engineering* **192:107273**. https://doi.org/10.1016/j.petrol.2020.107273.

Kaiser, E., Kutz, J. N., and Brunton, S. 2021. Data-driven Discovery of Koopman Eigenfunctions for Control. Machine Learning: Science and Technology, https://doi.org/10.1088/2632-2153/abf0f5.

Katayama, T. 2005. Subspace Methods for System Identification. Springer London. https://doi.org/10.1007/1-84628-158-x.

Kevorkian, J. 1990. Partial Differential Equations: Analytical Solution Techniques. **547**. Wadsworth & Brooks/Cole Advanced Books & Software.

Kincaid, D. and Cheney, W. 2002. Numerical Analysis: Mathematics of Scientific Computing: (The Sally Series; *Pure and Applied Undergraduate Texts*, Vol.**2**). American Mathematical Society.

Koopman, B. O. 1931. Hamiltonian Systems and Transformation in Hilbert Space. Proceedings of the national academy of sciences of the united states of america **17** (5): 315.

Kramer, B. and Willcox, K. E. 2019. Nonlinear Model Order Reduction Via Lifting Transformations and Proper Orthogonal Decomposition. *AIAA Journal* **57** (6): 2297–2307. https://doi.org/10.2514/1.J057791.

Kramer, B. and Willcox, K. E. 2020. Balanced Truncation Model Reduction for Lifted Nonlinear Systems.

Kutz, J. N., Brunton, S. L., Brunton, B. W., and Proctor, J. L. 2016. Dynamic Mode Decomposition : Data- driven Modeling of Complex Systems. Philadelphia: Society for Industrial / Applied Mathematics.

Kutz, J. N., Proctor, J. L., and Brunton, S. L. 2018. Applied Koopman Theory for Partial Differential Equations and Data-driven Modeling of Spatio-temporal Systems. **2018**:1–16. https://doi.org/10.1155/2018/6010634.

LeCun, Y., Bengio, Y., and Hinton, G. 2015. Deep Learning. *nature* **521** (7553): 436–444. https://doi.org/10.1038/nature14539.

Lie, K.-a. 2019. An Introduction to Reservoir Simulation Using Matlab/gnu Octave. Cambridge University Press, 2019.

Liu, D. and Wang, Y. 2021. A Dual-dimer Method for Training Physics-constrained Neural Networks with Minimax Architecture. *Neural Networks* **136**:112–125. https://doi.org/10.1016/j.neunet.2020.12.028.

Lusch, B., Kutz, J. N., and Brunton, S. L. 2018. Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics. *Nature Communications* **9** (1). https://doi.org/10.1038/s41467-018-07210-0.

Navrátil, J., King, A., Rios, J., Kollias, G., Torrado, R., and Codas, A. 2019. Accelerating Physics-based Simulations Using End-to-end Neural Network Proxies: An Application in Oil Reservoir Modeling. *Frontiers in Big Data* **2**. https://doi.org/10.3389/fdata.2019.00033.

Otto, S. E. and Rowley, C. W. 2019. Linearly Recurrent Autoencoder Networks for Learning Dynamics. *SIAM Journal on Applied Dynamical Systems* **18** (1): 558–593. https://doi.org/10.1137/18m1177846.

Rewienski, M. and White, J. 2003. A Trajectory Piecewise-linear Approach to Model Order Reduction and Fast Simulation of Nonlinear Circuits and Micromachined Devices. *IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems* **22** (2): 155–170. https://doi.org/10.1109/TCAD.2002.806601.

Schmid, P. J. 2010. Dynamic Mode Decomposition of Numerical and Experimental Data. *Journal of Fluid Mechanics* **656**:5–28. https://doi.org/10.1017/s0022112010001217.

Sirovich, L. 1987. Turbulence and the Dynamics of Coherent Structures. *Coherent I. Structures. Quarterly of applied mathematics* **45** (3): 561–571.

Souza, A., Castro, A., Dall'Aqua, M., Tueros, J., Horowitz, B., and Gildin, E. 2020. Nonlinear State Constraints Handling in Waterflooding Optimization through Reduced Order Models, 2020:**1-19**. 1. European Association of Geoscientists & Engineers. https://doi.org/10.3997/2214-4609.202035070.

Swischuk, R., Mainini, L., Peherstorfer, B., and Willcox, K. 2019. Projection-based Model Reduction: Formulations for Physics-based Machine Learning. *Computers & Fluids* **179**:704–717. https://doi.org/10.1016/j.compfluid.2018.07.021.

Takeishi, N., Kawahara, Y., Tabei, Y., and Yairi, T. 2017. Bayesian Dynamic Mode Decomposition. Presented at the IJCAI, 2814–2821.

Tan, X., Gildin, E., Florez, H., Trehan, S., Yang, Y., and Hoda, N. 2018. Trajectory-based DEIM (TDEIM) Model Reduction Applied to Reservoir Simulation. *Computational Geosciences* **23** (1): 35–53. https://doi.org/10.1007/s10596-018-9782-0.

Tu, J. H., Brunton, S. L., Luchtenburg, D. M., Rowley, C. W., and Kutz, J. N. 2014. On Dynamic Mode Decomposition: Theory and Applications. *Journal of Computational Dynamics* **1** (2): 391–421. https://doi.org/10.3934/jcd.2014.1.391.

Vanrossum, G. 1995. Python Reference Manual. *Department of Computer Science [CS], no. R 9525*.

Wang, S., Yu, X., and Perdikaris, P. 2022. When and Why Pinns Fail to Train: A Neural Tangent Kernel Perspective. *Journal of Computational Physics* **449**:110768. https://doi.org/10.1016/j.jcp.2021.110768.

Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. 2015. Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images. arXiv preprint arXiv:1506.07365.

Willcox, K. E., Ghattas, O., and Heimbach, P. 2021. The Imperative of Physics-Based Modeling and Inverse Theory in Computational Science. *Nature Computational Science* **1** (3): 166–168. https://doi.org/10.1038/s43588-021-00040-z.

Williams, M. O., Kevrekidis, I. G., and Rowley, C. W. 2015. A Data-driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. *Journal of Nonlinear Science* **25**, no. 6 (2015): 1307–1346. https://doi.org/10.1007/s00332-015-9258-5.

Yang, Y., Ghasemi, M., Gildin, E., Efendiev, Y., and Calo, V. 2016. Fast Multiscale Reservoir Simulations with POD-DEIM Model Reduction. *SPE Journal* **21** (06): 2141–2154. https://doi.org/10.2118/173271-pa.

Zalavadia, H., Sankaran, S., Kara, M., Sun, W., and Gildin, E. 2019. A Hybrid Modeling Approach to Production Control Optimization Using Dynamic Mode Decomposition. Presented at the SPE Annual Technical Conference and Exhibition. 2019. Society of Petroleum Engineers. https://doi.org/10.2118/196124-ms.

Zhang, L., Lam, J., Huang, B., and Yang, G.-h. 2003. On Gramians and Balanced Truncation of Discretetime Bilinear Systems. *International Journal of Control* **76** (4): 414–427. https://doi.org/10.1080/0020717031000082540.

Zubarev, D. I. 2009. Pros and Cons of Applying Proxy-Models as a Substitute for Full Reservoir Simulations. Presented at the SPE Annual Technical Conference and Exhibition. OnePetro, SPE. https://doi.org/10.2118/124815-ms.