Computers and Mathematics with Applications 147 (2023) 278–299

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Markov chain generative adversarial neural networks for solving Bayesian inverse problems in physics applications

Nikolaj T. Mücke [a,b,*], Benjamin Sanderse [a], Sander M. Bohté [a,c,d], Cornelis W. Oosterlee [b]

[a] Centrum Wiskunde & Informatica, Science Park 123, 1098 XG Amsterdam, Netherlands
[b] Mathematical Institute, Utrecht University, Utrecht, Netherlands
[c] Swammerdam Institute of Life Sciences (SILS), University of Amsterdam, Amsterdam 1098XH, Netherlands
[d] Bernoulli Institute, Rijksuniversiteit Groningen, Groningen 9747AG, Netherlands

ARTICLE INFO

Keywords:
Generative adversarial networks
Markov chain Monte Carlo
Inverse problems
Bayesian inference

ABSTRACT

In the context of solving inverse problems for physics applications within a Bayesian framework, we present a new approach, the Markov Chain Generative Adversarial Neural Network (MCGAN), to alleviate the computational costs associated with solving the Bayesian inference problem. GANs pose a very suitable framework to aid in the solution of Bayesian inference problems, as they are designed to generate samples from complicated high-dimensional distributions. By training a GAN to sample from a low-dimensional latent space and then embedding it in a Markov Chain Monte Carlo method, we can highly efficiently sample from the posterior, by replacing both the high-dimensional prior and the expensive forward map. This comes at the cost of a potentially expensive offline stage in which training data must be simulated or gathered and the GAN has to be trained. We prove that the proposed methodology converges to the true posterior in the Wasserstein-1 distance and that sampling from the latent space is equivalent to sampling in the high-dimensional space in a weak sense. The method is showcased in two test cases where we perform both state and parameter estimation simultaneously and it is compared with two conventional approaches, polynomial chaos expansion and ensemble Kalman filter, and a deep learning-based approach, deep Bayesian inversion. The method is shown to be more accurate than alternative approaches while also being computationally faster, in multiple test cases, including the important engineering setting of detecting leaks in pipelines.

1. Introduction

The Bayesian inference approach is popular for solving inverse problems in various fields including physics and engineering [1–4], mainly due to the fact that it does not only provide an estimate of the solution but also quantifies the uncertainty of the estimate. Information about the distribution of a computed quantity is important, for example, for digital twins [5].

The general idea of Bayesian inference is to use observations to update a given prior distribution towards a resulting posterior distribution over the parameters of interest. The observations and parameters are linked through a forward map and a noise distribution that make up the likelihood function. The main task in the Bayesian approach is to connect the prior and the likelihood in order to compute the posterior distribution. Since the posterior is typically not analytically tractable, one must use numerical sampling techniques such as Monte Carlo methods to approximate the distribution. However, for each sample, it is necessary to compute the likelihood which in turn requires the evaluation of the forward map. For nontrivial problems, such as high-dimensional or nonlinear partial differential equation (PDE) problems, this becomes a computational bottleneck and often results in unacceptable computation times. In Fig. 1, the general schematics of an inverse problem are shown.

The two most common approaches for overcoming this problem are to either minimize the required number of samples by making certain assumptions about the posterior or to reduce the computational complexity associated with the forward map by approximating it with a surrogate model. The first approach includes methods such as Kalman filters [2] and Markov Chain Monte Carlo (MCMC) methods [6,7]. With Kalman filters, one minimizes the number of necessary samples by assuming Gaussian distributions. While this is efficient, it is often quite restrictive when it

* Corresponding author at: Centrum Wiskunde & Informatica, Science Park 123, 1098 XG Amsterdam, Netherlands.
E-mail addresses: nikolaj.mucke@cwi.nl (N.T. Mücke), b.sanderse@cwi.nl (B. Sanderse), s.m.Bohte@cwi.nl (S.M. Bohté), c.w.oosterlee@uu.nl (C.W. Oosterlee).

https://doi.org/10.1016/j.camwa.2023.07.028

Available online 16 August 2023

comes to highly nonlinear problems. MCMC methods, while being quite efficient, are based on fewer assumptions but still require many samples. See Fig. 1 for a visualization of a common workflow for Bayesian inference using Markov chain Monte Carlo methods (MCMC). The surrogate modeling approach includes methods such as reduced basis methods [8], polynomial chaos expansion (PCE) [9], and Gaussian processes [10]. While a surrogate model enables fast likelihood evaluations, it requires a forward map that can be approximated by a low-order representation. This is however not trivial for problems with a so-called slow Kolmogorov n-width decay, such as very high-dimensional problems and problems involving discontinuities in either the parameters or the state.

In this paper, we consider an approach that overcomes the above mentioned challenges (high-dimensionality, nonlinearity, discontinuities, expensive sampling) by utilizing machine learning. Specifically, we will make use of neural networks which have already been recognized as promising tools in scientific computing, especially for the case of high-dimensional and nonlinear problems that we wish to address [11–17]. While there exist several types of neural networks, each aiming at solving specific problems, we focus on generative models in this paper. Generative models aim to learn a distribution from data in order to enable sampling from it at later times [18]. Such models include generative adversarial networks (GANs) [19], variational autoencoders (VAEs) [20], diffusion models [21], and Normalizing Flow models [22].

Examples where generative models have been successfully used for solving Bayesian inverse problems already exist. In [23] and [24], a VAE and Normalizing Flow, respectively, are embedded into a variational Bayesian inference approach and in [25] and [26] a GAN and a VAE, respectively, are used as the prior distribution in MCMC sampling. While [25] and [26] combine generative models for parameter prior approximation with MCMC sampling in order to get samples from the posterior distribution, they do not achieve significant speed-ups. Since the generative models are only used to approximate the parameter prior they still need the expensive forward problem being solved to match synthetic observations with the real observations. Furthermore, in [27] a GAN has been trained to directly sample from the posterior distribution. This is done by using a conditional GAN that generates samples conditioned on the observations. This approach achieves speed-ups as MCMC sampling is bypassed, but is restricted to the sensors configuration used for training. That is, the observation operator must be chosen when training in order to form the training set and cannot be changed without changing the architecture of the neural network and retraining it.

We will focus on GANs due to their success in learning complicated high-dimensional distributions. When choosing a generative model, there are essentially three aspects to consider [28]: Quality of samples, sampling speed, and mode coverage. VAEs generally generate lower quality samples than GANs, as they tend to blur the samples. Diffusion models [29], on the other hand, generate high quality samples, but they are significantly slower than both VAEs and GANs and are therefore not suitable for solving inverse problems in real-time. While the original GAN was known to suffer from mode collapse, it has been shown that the extension, the Wasserstein GAN (WGAN) [30], overcomes this issue to a large extent. Furthermore, the GAN training is more stable but it comes at a cost of computational time. As the training takes place in the offline stage, this is not a serious problem.

Specifically, GANs learn a target distribution by training a generator to map latent space samples to samples that mimic a nontrivial high-dimensional target distribution. So, GANs provide a way to represent a complicated high-dimensional distribution by means of a low-dimensional latent space distribution.

We here present the novel Markov Chain Generative Adversarial Network (MCGAN) method, visualized in Fig. 1. In short, we train a GAN to approximate the prior distribution for the states and parameters and thereby obtain a corresponding latent representation. By using an MCMC method, we can then efficiently sample from a latent space posterior instead of the high-dimensional posterior. As a result, we achieve dimensionality reduction, due to the approximation of the desired posterior, and furthermore the forward map is replaced by the generator. In practice, this gives significant computational speed-ups as the computational bottleneck is significantly reduced. The methodology presented draws inspiration from [25], but utilizes the GAN in a different manner. Our extension is hence well-suited for both state and parameter estimation in real-time.[1] Furthermore, we prove that sampling in the latent space is the same as sampling in the high-dimensional space in a weak sense and we provide a proof of convergence of the posterior distribution in the Wasserstein-1 distance.

The paper's outline is as follows. In Section 2, we explain the setting of Bayesian inverse problems as well as the MCMC methods and GANs. Then, in Section 3, we present the details of our proposed methodology, the MCGAN methodology, including the theoretical findings. In Section 4, we show the MCGAN performance on two problems: a stationary Darcy flow and leakage localization in a pipe flow. The results are compared to ensemble Kalman filters, MCMC methods with PCE as the surrogate model and the likelihood-free deep Bayesian inversion. Finally, in Section 5, we conclude this work.

2. Notation, problem setting, and preliminaries

Throughout the paper, we will make use of the following notation: capital letters will denote random variables, e.g. X and Y. The distribution of X is denoted P_x, where $P_x(A) = P_x(X \in A)$ is the probability of observing $X \in A$. Similarly, the probability of x is denoted $P_x(x) = P_x(X = x)$. We assume that all distributions have a probability density function (PDF), ρ_x. A stochastic variable, X, conditioned on another stochastic variable, Y, is denoted $X|Y$ and is distributed according to $P_{x|y}(X|Y)$ with PDF, $\rho_{x|y}$.

2.1. Problem setting

Let $\mathbf{q} \in \mathbb{R}^{N_q}$ denote the state, $\mathbf{m} \in \mathbb{R}^{N_m}$ the parameters, and $\mathbf{y} \in \mathbb{R}^{N_y}$ the available observations. Note that $\mathbf{q}$ encapsulates the state at all discrete times for time-dependent problems. Hence, the state, $\mathbf{q}$, is the full space-time state of the system at hand. $\mathbf{q}$ is computed by solving a forward problem, typically a PDE, depending on the parameters, $\mathbf{m}$. We denote the vector of combined state and parameters, $\mathbf{u} = (\mathbf{q}, \mathbf{m}) \in \mathbb{R}^{N_u}$, $N_u = N_q + N_m$.

The inverse problem deals with the recovery of $\mathbf{u}$ from a vector of observations in space-time, $\mathbf{y} \in O \subset \mathbb{R}^{N_y}$. By setting up the inverse problem in both state and parameters, we allow for the case where the state is not necessarily determined by the parameters of interest alone. The relation between $\mathbf{u}$ and $\mathbf{y}$ is assumed to be of the form

$$\mathbf{y} = \mathbf{h}(\mathbf{u}) + \eta, \quad \eta \sim P_\eta, \quad \eta \in \mathbb{R}^{N_y}, \tag{1}$$

where $\mathbf{h} : \mathbb{R}^{N_u} \to O \subset \mathbb{R}^{N_y}$ is referred to as the observation operator and η is a random variable denoting the observation or measurement noise.

[1] Note that "real-time" is dependent on the specific problem at hand.

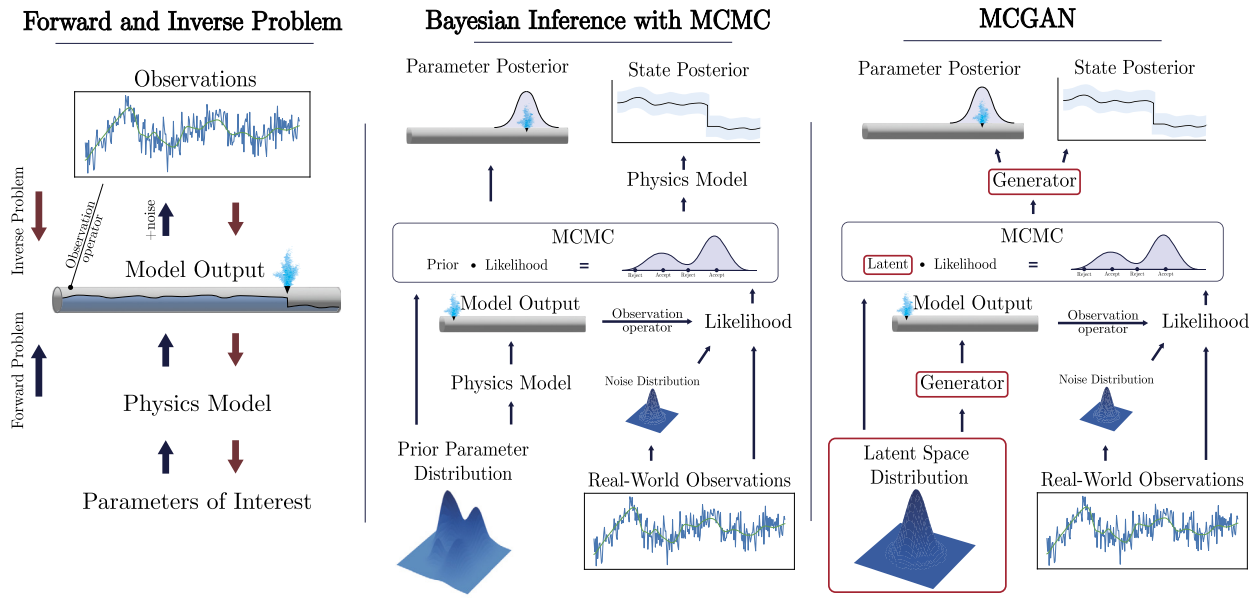


Fig. 1. Left: overview of the forward problem and the inverse problem. The parameters of interest are typically boundary and/or initial conditions, or physical parameters. The physics model depends on the system at hand and is here a PDE modeling pipe flow. The model output is the result obtained from a numerical simulation, such as pressure or velocity in the case of fluid dynamics. Observations are either observed from a set of sensors or created synthetically from the model output through the observation operator. Middle: a typical approach for doing Bayesian inference with MCMC (see Section 2). Right: our proposed method, the MCGAN approach as explained in section 3. Note that the complicated prior distribution is replaced with a simple latent distribution. Furthermore, the physical model is replaced with a generator that enables us to evaluate the full forward problem, more or less, instantaneously.

From Eq. (1), we can write the PDF associated with the probability of observing $\mathbf{y}$ given $\mathbf{u}$, $\rho_{y|u}(\mathbf{y}|\mathbf{u})$, as:

$$\rho_{y|u}(\mathbf{y}|\mathbf{u}) = \rho_\eta(\mathbf{y} - \mathbf{h}(\mathbf{u})). \tag{2}$$

When observations are given, one can view this as a function of $\mathbf{u}$, i.e., $\Phi(\mathbf{u}) = \rho_{y|u}(\mathbf{y}|\mathbf{u})$, in which case it is referred to as the *likelihood* since it is not a PDF with respect to $\mathbf{u}$.

We assume that, before observing any data, the probability of $\mathbf{u}$ has the PDF ρ_0, which is referred to as the *prior*. The goal of the Bayesian inverse problem is to identify the PDF, $\rho_{u|y}(\mathbf{u}|\mathbf{y})$, i.e. the PDF of $\mathbf{u}$ given observations, $\mathbf{y}$. Using Bayes theorem, we can write this as:

$$\rho_{u|y}(\mathbf{u}|\mathbf{y}) = \frac{\rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u})}{\int_{\mathbb{R}^{N_u}} \rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u})\,\mathrm{d}\mathbf{u}} = \frac{\rho_\eta(\mathbf{y} - \mathbf{h}(\mathbf{u}))\rho_0(\mathbf{u})}{\int_{\mathbb{R}^{N_u}} \rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u})\,\mathrm{d}\mathbf{u}}. \tag{3}$$

The denominator is called the *evidence* and serves as a normalization constant; the lefthand side is the *posterior*.

However, in order to compute the likelihood in Eq. (3), a PDE must be solved for a given set of parameters. Moreover, choosing a suitable prior is not always an easy task, and the evidence can be restrictive to compute in high dimensions.

It should be noted that the last problem is alleviated in many methods such as maximum likelihood estimation and MCMC methods as we will describe below. The other two complications will be minimized using our proposed methodology.

2.2. Markov chain Monte Carlo methods

MCMC methods [6,7] form a class of algorithms for sampling from probability distributions. Stated in terms of the posterior PDF, we have:

$$\rho_{u|y}(\mathbf{u}|\mathbf{y}) \propto \rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u}), \tag{4}$$

and we aim to generate a set of points distributed according to the PDF $\rho_{u|y}$. The general idea is to construct a Markov chain, $\{\mathbf{u}_1, \ldots, \mathbf{u}_N\}$, with a stationary PDF, $\tilde{\rho}_{u|y}$, that approximates $\rho_{u|y}$. We then sample according to $\tilde{\rho}_{u|y}$ by computing the next element in the chain. For MCMC algorithms, we have the following result, under some reasonable assumptions [6]:

$$\lim_{N_{mcmc}\to\infty} \frac{1}{N_{mcmc}} \sum_{i=1}^{N_{mcmc}} f(\mathbf{u}_i) = \mathbb{E}_{\mathbf{u}\sim P_{u|y}}[f(\mathbf{u})], \quad \mathbf{u}_i \sim \tilde{P}_{u|y} \tag{5}$$

where $\tilde{P}_{u|y}$ is the probability distribution associated with the density $\tilde{\rho}_{u|y}$. Eq. (5) indicates that with enough samples from the chain, we can approximate some statistics of the true posterior arbitrarily well, i.e. the distribution, $\tilde{P}_{u|y}$, converges weakly to $P_{u|y}$.

The arguably most common MCMC sampler is the Metropolis-Hasting (MH) algorithm [31,32]. However, it is well-known that the MH algorithm converges very slowly in high-dimensional settings. Therefore, in this paper, we make use of the Hamiltonian Monte Carlo Method (HMC), which can be considered a special case of the MH algorithm. Instead of computing new proposals by a random walk, the HMC algorithm computes a new sample by moving in a state space defined by a Hamiltonian ODE system.

Starting with a 'momentum' vector, $\mathbf{p}$, of the same size as $\mathbf{u}$, and a joint PDF $\rho_{u,p|y}(\mathbf{u},\mathbf{p}|\mathbf{y})$, we define a Hamiltonian as:

$$\begin{aligned} H(\mathbf{u},\mathbf{p}) &= -\log \rho_{u,p|y}(\mathbf{u},\mathbf{p}|\mathbf{y}) = -\log \rho_{p|u}(\mathbf{p}|\mathbf{u}) - \log \rho_{u|y}(\mathbf{u}|\mathbf{y}) \\ &\propto \underbrace{\frac{1}{2}\mathbf{p}^T M^{-1}\mathbf{p}}_{=K(\mathbf{p})} \underbrace{-\log\left[\rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u})\right]}_{=U(\mathbf{u})}, \end{aligned} \tag{6}$$

where we choose the conditional distribution of the momentum given $\mathbf{u}$ to be normally distributed, $P_{p|u}(\mathbf{p}|\mathbf{u}) \sim \mathcal{N}(0, M)$. $K(\mathbf{p})$ is referred to as the kinetic energy and $U(\mathbf{u})$ the potential energy. One can compute trajectories on level sets of the Hamiltonian by solving the Hamiltonian dynamical system. A new sample is then computed by perturbing the current sample, integrating the Hamiltonian system in time and using the final state as the new sample with acceptance probability:

$$\alpha = \min\left\{1, \frac{\exp\left(-H(\mathbf{u}',\mathbf{p}')\right)}{\exp\left(-H(\mathbf{u}_i,\mathbf{p}(0))\right)}\right\}, \tag{7}$$

where $(\mathbf{u}',\mathbf{p}')$ is the terminal state of the trajectory. Intuitively, this procedure will be biased towards sampling from level sets in the phase space that maximize the likelihood $U(\mathbf{u})$. Furthermore, $K(\mathbf{p})$ ensures that the algorithm explores other areas of the phase space to a degree decided by M and the integration horizon, T. Compared to the standard MH algorithm, this reduces the correlation between elements in the chain by traversing long distances in the phase space while maintaining a high acceptance probability due to the energy preserving properties of Hamiltonian dynamics.

When sampling using the HMC algorithm, a series of choices have to be made, like the number of time steps in the integration and the end time, T. If T is too small the sampling will resemble a random walk, while T too large may result in trajectories making a 'U-turn' and return to their initial condition. To avoid this, we utilize the No U-Turn Sampler (NUTS) [33].

The idea is to integrate backward and forward in time until a U-turn condition is satisfied. Then, a random point from the computed trajectory is chosen, and the algorithm continues from there.

It is important to emphasize that the HMC algorithm can only be utilized when the likelihood and the prior are differentiable. Furthermore, the derivative should be cheap to compute to get the desired speed-up.

Even though HMC with NUTS is efficient, one still needs many samples to converge. With a good initial sample, the method converges significantly faster. There are several ways of computing a suitable initial guess, one of which is the maximum a posteriori (MAP) estimate [4], which we will use in this work. This is typically computed using the log PDFs:

$$\mathbf{u}_{\text{MAP}} = \arg\max_{\mathbf{u}} \log(\rho_{u|y}(\mathbf{y}|\mathbf{u})) + \log(\rho_0(\mathbf{u})). \tag{8}$$

$\mathbf{u}_{\text{MAP}}$ can be computed using standard optimization methods such as gradient descent methods. In our case, both $\rho_{y|u}$ and ρ_u are known PDFs so it is easy to compute derivatives using standard software libraries such as PyTorch.

2.3. Alternative methods

Here, we will comment on some well-known alternative methods that exist to speed up solving the Bayesian inverse problem, which we will use to compare our proposed method to. We will also comment on their respective shortcomings.

Ensemble Kalman filter Ensemble Kalman filtering (EnKF) is a Kalman filter variant that is suitable for high-dimensional and nonlinear problems [2]. The general idea is to compute the sample mean and sample covariance from an ensemble and then update the prior accordingly. However, as all distributions are assumed to be Gaussian, it means that it is not directly suitable for non-Gaussian problems. In cases with very nonlinear or high-dimensional features, large ensembles are necessary which in turn makes it computationally slow.

Alternative approaches exist, such as particle filters, that do not assume a Gaussian distribution. However, such methods are, in general, computationally very expensive and will not be further discussed.

Surrogate models Instead of replacing the sampling method, the forward computations can be done using a surrogate model, such as polynomial chaos expansion (PCE) methods [9], Gaussian processes [34], or reduced basis methods [8]. The idea is to approximate the parameter-to-observations map or the forward map by a low-order model that is computationally fast to evaluate. These approaches have been shown to speed up the sampling significantly. However, in high-dimensional cases the curse of dimensionality hampers the applicability of such methods. Moreover, they usually do not perform well in discontinuous and/or highly nonlinear cases unless the approach is tailored to the problem at hand.

Likelihood-free methods As mentioned above, the computationally most expensive task is to evaluate the likelihood as this requires the solution of the forward problem. Alternatively, one can compute the posterior distribution without computing the likelihood. Such approaches are termed likelihood-free methods.

There are several ways of making use of likelihood-free methods. One approach that has become increasingly popular is to learn the posterior directly in an offline stage [27]. Here, pairs of state/parameters and corresponding observations, $(\mathbf{u},\mathbf{h}(\mathbf{u}))$, are required in the training stage. Then a model is trained to approximate the posterior, $P(\mathbf{u}|\mathbf{h}(\mathbf{u})) \approx P(\mathbf{u}|\mathbf{y})$.

While this approach enables fast computation of the posterior in the online stage, because no sampling is needed, it is less flexible, as $\mathbf{h}$ cannot change between the offline and the online stages, meaning that sensor configurations must be constant. Hence, in cases where the sensor configuration is not known a priori, the methodology is not usable. Furthermore, if the noise levels in the observations change, this cannot be incorporated in the online stage.

2.4. Generative adversarial neural networks

In this section, we will give a brief overview of generative adversarial networks (GANs), see [19,35] for more details. We will focus on a version of GANs called Wasserstein GAN (WGAN) [30].

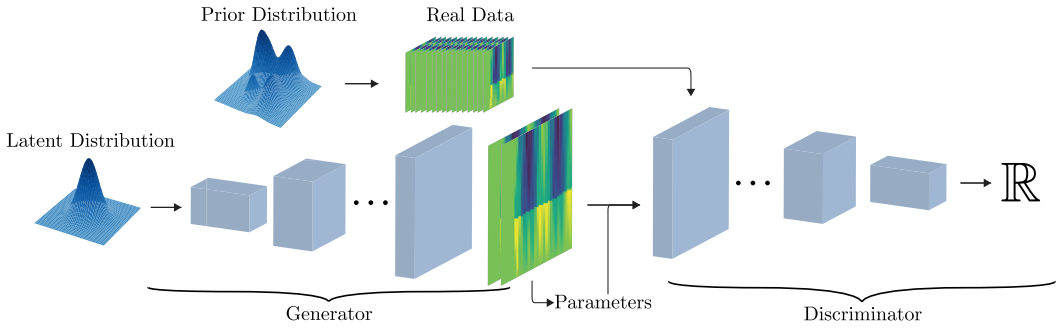


Fig. 2. GAN architecture.

GANs deal with the problem of learning an unknown distribution from samples. Consider a probability distribution, P_u^r, on a data space which is a subset of $\mathbb{R}^m$. We aim to approximate P_u^r with another distribution, P_u^g. We will refer to P_u^r as the real data probability distribution or the target distribution, and P_u^g the generated distribution. In order to compute P_u^g, we define a stochastic latent variable, $Z \in \mathbb{R}^{N_z}$, with prior distribution P_z^g, typically chosen to be a Gaussian. Then, we define a generator, $G_\theta : \mathbb{R}^{N_z} \to \mathbb{R}^{N_u}$, which is a neural network parameterized by its weights, θ. G_θ takes in the latent variable and outputs $G_\theta(Z) \sim P_u^g$. Hence, $P_u^g = G_{\theta\#} P_z^g$ is the *pushforward* of the latent space distribution with PDF $\rho_u^g = \rho_z^g \circ G^{-1}$ [36]. By choosing $N_z \ll N_u$, we effectively get a low-dimensional representation of the N_u-dimensional distribution. Therefore, the variable z can be considered a latent/low-dimensional representation of samples from P_u^r.

Next, we introduce the discriminator, $D_\omega : \mathbb{R}^{N_u} \to \mathbb{R}$. The discriminator takes in samples from either the real data probability distribution or the generated probability distribution, and returns a real number called the *score*. A large score means that the discriminator believes the sample comes from the real data distribution. D_ω is a neural network parameterized by its weights, ω.

In order to learn the target distribution, a zero-sum game between the generator and the discriminator is set up. The generator aims to maximize the discriminator output, while the discriminator tries to minimize the score of generated samples while simultaneously trying to maximize the score of the real samples. For the WGAN, this game is mathematically formulated as [30]:

$$\inf_\theta \sup_\omega \quad \mathbb{E}_{X\sim P_u^r}\left[D_\omega(X)\right] - \mathbb{E}_{Z\sim P_z^g}\left[D_\omega(G_\theta(Z))\right]. \tag{9}$$

It can be shown that this inf-sup problem is equivalent to minimizing the Wasserstein-1 distance between P_u^r and P_u^g due to the Kantorovich-Rubinstein duality [30]. The WGAN framework requires the discriminator to be Lipschitz continuous with respect to the input. Therefore, we introduce a gradient penalty term to constrain the gradient of the discriminator [37]:

$$\inf_\theta \sup_\omega \quad \mathbb{E}_{X\sim P_u^r}\left[D_\omega(X)\right] - \mathbb{E}_{Z\sim P_z^g}\left[D_\omega(G_\theta(Z))\right] - \lambda \mathbb{E}_{\hat{\mathbf{x}}\sim P_{\hat{X}}}\left[\left(||\nabla_{\hat{X}} D_\omega(\hat{X})|| - 1\right)^2\right], \tag{10}$$

where λ is a regularization parameter to be tuned, $\hat{X} = \epsilon X + (1-\epsilon)G_\theta(Z)$, and ϵ is a small positive number.

In practice, we do not update the weights of the generator and the discriminator at the same time. Instead, we split (10) into two subproblems: a generator loss that aims to minimize (10) and a discriminator loss that aims to maximize (10) by minimizing the negative value:

$$L_G = -\mathbb{E}_{Z\sim P_z^g}\left[D_\omega(G_\theta(Z))\right], \qquad \text{(Generator Loss)} \tag{11a}$$

$$L_D = -\mathbb{E}_{X\sim P_u^r}\left[D_\omega(X)\right] + \mathbb{E}_{Z\sim P_z^g}\left[D_\omega(G_\theta(Z))\right] + \lambda \mathbb{E}_{\hat{X}\sim P_{\hat{X}}}\left[\left(||\nabla_{\hat{X}} D_\omega(\hat{X})|| - 1\right)^2\right]. \qquad \text{(Discriminator Loss)} \tag{11b}$$

For details about the training, see Appendix C. The WGAN is visualized in Fig. 2.

It has been shown that if the generator and the discriminator have sufficient capacity, the generated distribution converges to the real data probability distribution in the Wasserstein-1 distance [38].

3. Markov chain GAN

In this section, we will outline our proposed method, the Markov Chain GAN (MCGAN) method. The general purpose is to combine MCMC methods with GANs in order to perform state and parameter estimation in a computationally fast and accurate way. As mentioned in the previous section, similar approaches exist using polynomial surrogate models [39,40] and Gaussian processes [10]. However, as will be discussed, using GANs gives significant advantages over these alternatives.

3.1. Proposed algorithm

In short, the proposed algorithm aims to speed up posterior sampling without compromising too much on accuracy. The general methodology is to replace the forward model in the likelihood computation with the generator and replace the data prior with the GAN latent distribution (see Fig. 2).

Firstly, in an offline stage, we train the GAN to generate discrete solutions to the PDE for the desired time span and corresponding parameters. The GAN is trained on samples from the real prior distribution, P_0^r, i.e. solutions computed through conventional numerical methods (finite elements, finite volumes, etc.) and aims to learn a generated prior distribution, P_0^g. Samples from P_0^r are typically computed by sampling the parameters and then solving the physical model to get the state. This can be a lengthy process, but the training data can be simulated completely in parallel on several computer cores. After training, the (single) generator may generate pairs of states and parameters from a latent sample, $\mathbf{z}$:

$$G_\theta(\mathbf{z}) = (G_\theta^q(\mathbf{z}), G_\theta^m(\mathbf{z})) = (\mathbf{q}^g, \mathbf{m}^g) = \mathbf{u}^g \sim P_0^g \tag{12}$$

Hence, the generated distribution is approximating the real prior distribution, $P_0^g \approx P_0^r$. In the training, the discriminator will receive pairs of states and parameters, $(\mathbf{q}^r, \mathbf{m}^r)$, sampled from the real data prior and generated pairs of states and parameters, $(\mathbf{q}^g, \mathbf{m}^g)$, sampled from the generated distribution, in order to ensure that the generator learns to generate states and parameters that match.

Remark. It should be noted that since the GAN is trained on discrete solutions on a specific grid, it will generate samples on the same grid. Therefore, it is important to train on discrete solutions that have all the necessary properties for the online phase (sufficient resolution, etc.).

In order to take advantage of the low-dimensional latent space, we need to be able to sample solutions and parameters from a posterior on the latent space, instead of the generated distribution in the full data space. To this end, we have the expression for the latent space posterior density:

$$\rho^g_{z|y}(\mathbf{z}|\mathbf{y}) = \frac{\rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})}{\int_{\mathbb{R}^{N_z}} \rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})\,\mathrm{d}\mathbf{z}} = \frac{\rho_\eta(\mathbf{y}-\mathbf{h}(G_\theta(\mathbf{z})))\rho^g_z(\mathbf{z})}{\int_{\mathbb{R}^{N_z}} \rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})\,\mathrm{d}\mathbf{z}} \propto \rho_\eta(\mathbf{y}-\mathbf{h}(G_\theta(\mathbf{z})))\rho^g_z(\mathbf{z}). \tag{13}$$

By training the generator to generate pairs of states and parameters, there is no need for the expensive forward model, as it is replaced by an evaluation of the generator at the sampled $\mathbf{z}$. This means that in the online stage we only have to evaluate the GAN once in order to get both the state and parameters. This is in contrast to "conventional" approaches where a surrogate model only approximates the forward map or the parameters. In that case, one would have to make use of two surrogates – one for the parameters and one for the forward map. Furthermore, it also takes into account the slightly more general case where there is not necessarily a deterministic relationship between the parameters and the state.

This approach yields a significant speed-up in online computation time since evaluating the generator is, more or less, instantaneous and we obtain the posterior over both the state and parameters at no extra cost. The derivation of (13) is given in Section 3.2 in Theorem 1.

In the online stage, we then use the MCMC method, as discussed in Section 2.2, to sample from the latent space posterior. We can make use of the highly efficient HMC algorithm since derivatives of the likelihood are easily obtained through back propagation of the neural network generator. For conventional numerical methods, this is rarely the case as computing derivatives of the forward model typically requires expensive numerical approximations or adjoint methods. Furthermore, the MAP estimate is also cheap to compute for the same reasons, which provides an excellent starting point for the HMC algorithm. Since the dimension is lowered significantly, and derivatives and an appropriate starting point are cheaply available, the Markov chain will converge significantly faster and we can get by with much fewer samples. In conclusion, with the MCGAN methodology each sample is cheap and we need fewer samples than for conventional methods.

Given the MCMC samples, we can compute derived quantities, e.g. for a given quantity of interest f, we can compute the expected value by:

$$\mathbb{E}_{\mathbf{q}\sim P^r_{q|y}}[f(\mathbf{q})] \approx \mathbb{E}_{\mathbf{q}\sim P^g_{q|y}}[f(\mathbf{q})] = \mathbb{E}_{\mathbf{z}\sim P^g_{z|y}}[f(G^q_\theta(\mathbf{z}))] \approx \frac{1}{N_{\mathrm{MCMC}}}\sum_{i=1}^{N_{\mathrm{MCMC}}} f(G^q_\theta(\mathbf{z}_i)), \quad \mathbf{z}_i \sim \tilde{P}^g_{z|y}, \tag{14a}$$

$$\mathbb{E}_{\mathbf{m}\sim P^r_{m|y}}[f(\mathbf{m})] \approx \mathbb{E}_{\mathbf{m}\sim P^g_{m|y}}[f(\mathbf{m})] = \mathbb{E}_{\mathbf{z}\sim P^g_{z|y}}[f(G^m_\theta(\mathbf{z}))] \approx \frac{1}{N_{\mathrm{MCMC}}}\sum_{i=1}^{N_{\mathrm{MCMC}}} f(G^m_\theta(\mathbf{z}_i)), \quad \mathbf{z}_i \sim \tilde{P}^g_{z|y}. \tag{14b}$$

By choosing an appropriate f, we can thereby compute various quantities of interest by sampling the latent space posterior. See Theorem 1 in Section 3.2 for details.

For an overview of the methodology, see Algorithm 1 for the offline stage and Algorithm 2 for the online stage. Here is a summary of the distinct advantages of the proposed method compared to the alternatives discussed in Section 2.3:

- The latent vector $\mathbf{z}$ is, in general, of significantly lower dimension than the state and parameters, effectively reducing the dimension of the stochastic space resulting in significantly faster convergence of MCMC methods;
- The computationally expensive forward problem is replaced by the generator, whose cost is computationally negligible to evaluate once it has been trained;
- Since the forward map is replaced by a neural network, derivatives of the log-likelihood function can be computed efficiently, which enables computationally fast MAP estimation and allows us to utilize the highly efficient HMC method for sampling.

While the advantages are clear, it is worth mentioning the drawbacks as well:

- There is no immediate way of choosing the dimension of the latent space. However, one can consider it a hyperparameter and perform hyperparameter optimization;
- Training a GAN is not always an easy task, since commonly known problems of training neural networks, such as local minima and generalization, also apply here;
- It is necessary to generate much training data in order to ensure accuracy of the GAN.

Note that the drawbacks are not unique to this methodology, but general when dealing with neural networks. The first two points are a matter of hyperparameter tuning, and the last point is a matter of time in the offline stage. Furthermore, with an efficient numerical solver and the fact that the offline stage can be easily parallelized (since the training samples are independent), the generation of data is often feasible within a reasonable timeframe. If the forward solver would be too expensive to allow for this, it is recommended to first obtain a simpler forward model e.g. by model reduction techniques such as reduced order models.

Remark. The purpose of the proposed methodology is to solve Bayesian inverse problems computationally fast in an online stage. As mentioned, this comes at a cost of an expensive offline stage in which the GAN is trained on simulated data. However, when the GAN is trained it can be deployed in several settings. Therefore, the method is highly suitable in settings where computational speed is crucial and offline training time is less important. This is, for example, the case for digital twins and model predictive control where repeated real-time state estimation and parameter calibration are necessities.

Algorithm 1: MCGAN offline stage.

Input : N_{train}, GAN hyperparameters, GAN architecture
1 Generate training samples, $\{(\mathbf{q}_i, \mathbf{m}_i)\}_{i=1}^{N_{\text{train}}} \sim P_0^r$, by solving the forward problem;
2 Train the GAN to approximate the prior, $P_0^g \approx P_0^r$ (see Section 2.4);
Output : Trained generator, G_θ

Algorithm 2: MCGAN online stage.

Input : Generator from Algorithm 1, MCMC parameters, observations ($\mathbf{y}$),
1 Compute the MAP estimate in the latent space, using a gradient descent algorithm, as initial sampling point (see Eq. (8));
2 Use MCMC algorithm (see Section 2.2) to sample from the latent space posterior (see Eq. (13));
3 Generate states and parameters from the posterior latent space samples:

$$\{\mathbf{q}_i, \mathbf{m}_i\} = \{G_\theta^q(\mathbf{z}_i), G_\theta^m(\mathbf{z}_i)\} = \{G_\theta(\mathbf{z}_i)\}, \quad i = 1, \dots, N_{\text{samples}} \quad \mathbf{z}_i \sim \tilde{P}_{z|y}^g.$$

4 Compute the relevant statistics, such as mean and variance;
Output : $\{\mathbf{q}_i, \mathbf{m}_i\}_{i=1}^{N_{\text{samples}}}$, statistics

3.2. Latent space sampling

Here we prove that sampling from the latent space posterior essentially yields the same results as sampling from the full data space in a weak sense. From [41], we have the following results for push forward distributions:

$$\mathbb{E}_{U\sim P_u}[f(U)] = \int_E f(\mathbf{u})\rho_u(\mathbf{u})\mathrm{d}\mathbf{u} = \int_{G^{-1}(E)} f(G(\mathbf{z}))\rho_z(\mathbf{z})\mathrm{d}\mathbf{z} = \mathbb{E}_{Z\sim P_z}[f(G(Z))], \tag{15}$$

where $\mathbf{u} = G(\mathbf{z}) \in E$, P_u and P_z are the distributions of $\mathbf{u}$ and $\mathbf{z}$ with PDFs ρ_u and ρ_z, respectively, and f is a measurable function on E. Here, $P_u = G_\# P_z^g$ is the push forward of P_z by G. The derivation of Eq. (15) only requires that G is measurable.

Theorem 1. *Let G_θ be a generator. Let Z be a latent space variable distributed according to a latent space distribution, P_z^g, with PDF ρ_z^g, and let $U = G_\theta(Z)$ be distributed according to the push forward distribution of the latent space distribution, $U \sim P_u^g = G_{\theta\#} P_z^g$, with PDF ρ_0^g. Then, the push forward posterior distribution, conditioned on data $\mathbf{y}$, is equal to the latent space posterior distribution conditioned on the same data in a weak sense, i.e. for all measurable functions f, the following holds:*

$$\mathbb{E}_{U\sim P_{u|y}^g}[f(U)] = \mathbb{E}_{Z\sim P_{z|y}^g}[f(G(Z))], \tag{16}$$

where the associated PDFs are given by:

$$\rho_{u|y}^g(\mathbf{u}|\mathbf{y}) = \frac{\rho_{y|u}^g(\mathbf{y}|\mathbf{u})\rho_0^g(\mathbf{u})}{\int_{\mathbb{R}^{N_u}} \rho_{y|u}^g(\mathbf{y}|\mathbf{u})\rho_0^g(\mathbf{u})\,d\mathbf{u}}, \quad \rho_{z|y}^g(\mathbf{z}|\mathbf{y}) = \frac{\rho_{y|u}^g(\mathbf{y}|G_\theta(\mathbf{z}))\rho_z^g(\mathbf{z})}{\int_{\mathbb{R}^{N_z}} \rho_{y|u}^g(\mathbf{y}|G_\theta(\mathbf{z}))\rho_z^g(\mathbf{z})\,d\mathbf{z}}. \tag{17}$$

The proof can be found in Appendix A.

Using Theorem 1, we can conclude that sampling from the latent space posterior, $P_{z|y}^g$, and pushing forward using the generator, G_θ, yields the same results as sampling directly from the generated posterior, $P_{u|y}^g$ in a weak sense. While small perturbations in the latent domain might result in different output in the full data space, Theorem 1 shows that, in a weak sense, the posterior obtained from pushing forward the latent samples is equal to the full data posterior.

3.3. Convergence of generated posterior

In this subsection, we prove that $P_{u|y}^g \approx P_{u|y}^r$ when $P_0^g \approx P_0^r$ and under some additional reasonable assumptions on the generator. That is, the case where the prior is approximated in the Wasserstein metric and the likelihood is approximated with a surrogate forward map. While [42] proves the cases where either the likelihood or the prior is approximated, we provide a proof where both are being approximated.

Before stating the theorem, we need to define the appropriate spaces and metrics. Let (E, d_E) be a complete metric space. $E \subset \mathbb{R}^d$ is the set containing the state and parameter vectors, $\mathbf{u} \in E$ and $d_E : E \times E \to \mathbb{R}_+$ assigns non-negative distances between two elements of E. Furthermore, in this formulation, the observation operator, $\mathbf{h} : E \to O$ maps elements from E to the observation space.

We can then define the relevant space of probability distributions:

Definition 1. On a metric space, (E, d_E), we define the space of probability distributions as:

$$\mathcal{W}_q(E) = \left\{ P \;:\; |P|_{\mathcal{W}_q} < \infty \right\}, \quad |P|_{\mathcal{W}_q} = \inf_{x_0 \in E} \left(\int_E d_E(x, x_0)^q \rho(x)\,\mathrm{d}x \right)^{1/q}.$$

Then, we define the Wasserstein-1 distance and its dual representation [43]:

Definition 2. For two probability distributions, $P_1, P_2 \in \mathcal{W}_1(E)$, the Wasserstein-1 distance is defined as:

$$W_1(P_1, P_2) = \inf_{\gamma \in \Gamma(P_1, P_2)} \left| \int_E \int_E d_E(x, y)\gamma(x, y)\mathrm{d}x\mathrm{d}y \right|,$$

where $\Gamma(P_1, P_2)$ is the set of joint PDFs, γ, for combined probability distributions with P_1 and P_2 as marginal distributions, respectively. From the Kantorovich–Rubinstein duality, we can write the Wasserstein-1 distance as:

$$W_1(P_1, P_2) = \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(x)\rho_1(x)\mathrm{d}x - \int_E f(x)\rho_2(x)\mathrm{d}x \right|,$$

where $f : E \to \mathbb{R}$ is a Lipschitz continuous function, $\mathrm{Lip}(f)$ is its corresponding Lipschitz constant, and ρ_1 and ρ_2 are the PDFs of P_1 and P_2, respectively.

Besides the Wasserstein distance, we will also be working with the weighted norms:

$$||f||_{L^1_\rho} = \int |f(x)|\rho(x)\,\mathrm{d}x, \quad ||f||_{L^2_\rho} = \left(\int |f(x)|^2\rho(x)\,\mathrm{d}x \right)^{1/2}.$$

With the proper spaces, norms, and metrics defined, we can state the following theorem inspired by [42]:

Theorem 2. *Let (E, d_E) be a bounded metric space with $\sup_{\mathbf{x}_1,\mathbf{x}_2\in E} d_E(\mathbf{x}_1, \mathbf{x}_2) \leq D < \infty$.*

Let $P_0^r \in \mathcal{W}_2(E)$ denote the prior probability distribution of real data, and let $P_0^r \in \mathcal{W}_2(E)$ denote the generated prior probability distribution of generated data.

Let the real data and generated likelihoods satisfy

$$\rho^r_{y|u}(\mathbf{y}|\mathbf{u}) \propto \Phi^r(\mathbf{u}) = e^{-l^r(\mathbf{u})}, \quad \Phi^r : E \to \mathbb{R}_+, \quad l^r : E \to \mathbb{R}_+,$$
$$\rho^g_{y|u}(\mathbf{y}|\mathbf{u}) \propto \Phi^g(\mathbf{u}) = e^{-l^g(\mathbf{u})}, \quad \Phi^g : E \to \mathbb{R}_+, \quad l^g : E \to \mathbb{R}_+,$$

where l^r and l^g are the log-likelihood functions for the real and generated data, respectively, and Φ^r and Φ^g are Lipschitz continuous functions with Lipschitz constants $\mathrm{Lip}(\Phi^r)$ and $\mathrm{Lip}(\Phi^g)$, respectively. Furthermore, let $\Phi^r, \Phi^g \in L^2_{\rho_0^g}$, where $L^2_{\rho_0^g}$ is the weighted L^2 space with ρ_0^g as the weight function.

Assume the GAN has converged, i.e. $W_1(P_0^r, P_0^g) \leq \epsilon_1$. Furthermore, assume that this implies convergence of the log-likelihood, as follows,

$$||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^1_{\rho_0^g}} \leq \epsilon_2, \quad ||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^2_{\rho_0^g}} \leq \epsilon_3, \tag{18}$$

where $||\cdot||_{L^1_{\rho_0^g}}$ is the weighted L^1-norm with ρ_0^g as the weight function. Then, the Wasserstein-1 distance between the real posterior probability distribution given observations and the generated posterior probability distribution given observations satisfies:

$$W_1(P^r_{u|y}, P^g_{u|y}) \leq C_1\epsilon_1 + C_2\epsilon_2 + C_3\epsilon_3, \tag{19}$$

where

$$C_1 = \frac{(1 + D\mathrm{Lip}(\Phi^r))}{Q^r_u(\mathbf{y})}, \quad C_2 = \frac{\max(\Phi^r, \Phi^g)}{Q^r_u(\mathbf{y})Q^g_u(\mathbf{y})}(1 + D\mathrm{Lip}(\Phi^r))|P_0^g|_{\mathcal{W}_1}, \quad C_3 = \frac{\max(\Phi^r, \Phi^g)}{Q^g_u(\mathbf{y})}|P_0^g|_{\mathcal{W}_2},$$

where Q^r_u and Q^g_u are the evidence from the real and generated posterior, respectively, and D denotes the maximum distance between two points in the metric space, E.

The proof can be found in Appendix B.

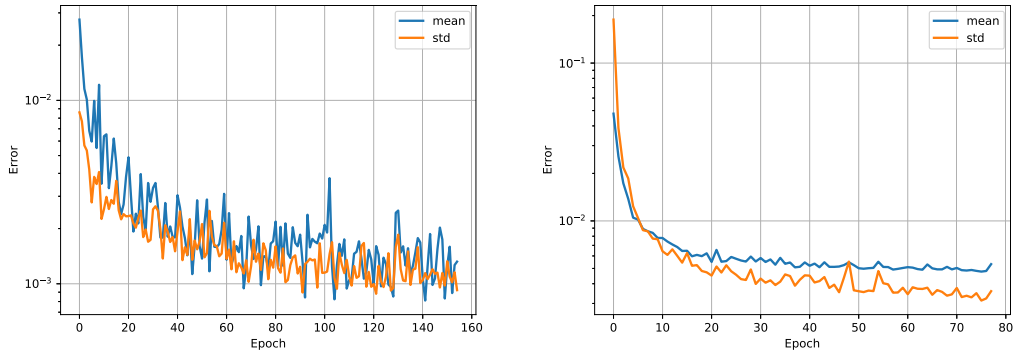
In short, the proof of Theorem 2 helps us understand when we can expect convergence of the posterior. While the assumptions of the Theorem might seem restrictive, this is actually not the case. Firstly, we assume that the metric space, E, is bounded, which is typically the case in many applications. Secondly, we assume that the likelihood is of the form $e^{-l(\mathbf{u})}$, Lipschitz continuous, and is in the weighted L^1 and L^2 spaces. For simple observation operators (e.g. linear), this is a consequence of the negative log-likelihood function typically being an L^2-norm. This leaves us with the question of the convergence of the prior, which directly determines ϵ_1, ϵ_2, and ϵ_3.

3.4. Convergence of the generated prior

The convergence of the generated prior is a matter of studying convergence properties of GANs. Such studies are beyond the scope of this paper. Instead, we refer to [30,37,38] where convergence properties of GANs are discussed. In short, ensuring convergence of GANs is similar to ensuring convergence of other types of neural networks. Hence, it is a matter of having enough data and performing hyperparameter tuning. For the MCGAN, the amount of data is, in general, not a problem, as we simulate the training data.

4. Results

In this section, we will present the results on two different problems using the MCGAN methodology. We show two distinct parameter and state estimation cases to highlight various advantages of using the MCGAN methodology. Firstly, we consider a Darcy flow case (stationary flow through a porous medium), with the aim of approximating the horizontal and vertical velocity, the pressure, and the permeability field. The purpose of



(a) Darcy flow. (b) Pipeflow.

Fig. 3. Convergence of mean and variance of the generated prior towards the prior computed from simulations. The error is computed using a test dataset.

this case is to emphasize the ability to deal with high-dimensional stochastic problems as the permeability field is spatially distributed and follows a high-dimensional distribution. Secondly, we consider the problem of leakage detection in pipe flow. Here, the challenge lies in dealing with a nonlinear hyperbolic PDE with discontinuities and a non-informative prior.

The results will be assessed using the relative root mean squared error (RRMSE):

$$\text{State RRMSE} = \frac{\sqrt{\sum_{i=1}^{N_q}(\mathbf{q}_i^* - \mathbf{q}_i)^2}}{\sqrt{\sum_{i=1}^{N_q}\mathbf{q}_i^2}}, \quad \text{Parameter RRMSE} = \frac{\sqrt{\sum_{i=1}^{N_m}(\mathbf{m}_i^* - \mathbf{m}_i)^2}}{\sqrt{\sum_{i=1}^{N_m}\mathbf{m}_i^2}}, \tag{20}$$

where $\mathbf{q}^*$ and $\mathbf{m}^*$ denote the approximated state and parameters, respectively, and $\mathbf{q}$ and $\mathbf{m}$ are the reference state and parameters, respectively. The reference values are computed using an appropriate numerical solver. These will be discussed in each test case.

Furthermore, we will look at the approximated posterior distributions resulting from the MCGAN.

For the details on the training and hyperparameters for each of the test cases as well as GAN architectures, see Appendix C. Furthermore, all the training data for the GANs are generated by sampling the parameter spaces according to the chosen distribution for the test case. The number of training samples is chosen based on the performance of the resulting GAN. Note that it is, in general, a difficult problem to choose the number of necessary training samples.

The specific architectures of the generators and discriminators for each test case can be found in Fig. C.8. It is worth noting that we make use of convolutional neural networks in all cases due to their success in problems dealing with spatially distributed degrees of freedom [14,11]. It should, however, be noted that convolutional neural networks can essentially only be applied to Cartesian grids. To deal with irregular grids, one could make use of alternative architectures, such as graph neural networks, as in [44], or operator neural networks, such as Fourier neural operators [16].

As mentioned in Section 3.4, it is not feasible to compute the Wasserstein distance for very high-dimensional distributions. Therefore, in order to show convergence of the generated prior, we show the convergence of the first two moments, mean and standard deviation, with the training epochs. Here, we have a value for the mean and variance at every grid point and we compute the error as the relative RMSE. The convergence plots are shown in Fig. 3. While this is a weaker type of convergence than convergence in the Wasserstein-1 distance, it still gives an indication that the GAN error is sufficiently small for the purpose of Bayesian inversion.

We compare the proposed set-up with three alternatives: Ensemble Kalman filter, polynomial chaos expansion, and deep Bayesian inversion (DBI) [27]. Brief summaries of each methods can be found in Appendix D. For the first test case, it is worth noting that we make use of a variation called ensemble Kalman inversion, that is suitable for stationary problems. For the DBI, we make use of the same architectures, except for the input. In the generator, the observations are concatenated with the latent variables and for the discriminator the observations are concatenated with output of the convolutional layers. Furthermore, both the PCE and the DBI approaches are trained to the specific sensor configurations. Hence, they are less flexible than the Kalman filter and MCGAN methods, which allow for varying sensor configurations and a change of likelihood function.

All results are generated using synthetic observations. Therefore, all observations are simulation-based and perturbed with artificial noise. To ensure that we are not subject to inverse crime [45], the synthetic observations are generated with a higher resolution than what is used for the training of the GANs and PCE models, for all experiments. Furthermore, the Kalman filter results are also generated with a lower resolution. Secondly, we will use another distribution for the likelihood function than for the noise in the synthetic observations. The specifics will be discussed in each test case.

The number of necessary MCMC samples was considered a hyperparameter to be tuned and we chose the smallest number of samples that did not sacrifice accuracy in both cases.

4.1. Darcy flow

As a first test case, we consider stationary two-dimensional Darcy flow:

$$\mathbf{v} + k\nabla p = 0, \qquad \mathbf{x} \in [0,1]^2, \tag{21a}$$

$$\nabla \cdot \mathbf{v} = 0, \qquad \mathbf{x} \in [0,1]^2, \tag{21b}$$

$$p = 1, \qquad \mathbf{x} \in 0 \times [0,1], \tag{21c}$$

$$p = 0, \qquad \mathbf{x} \in 1 \times [0,1], \tag{21d}$$

$$\mathbf{v}\cdot\mathbf{n}=0,\quad \mathbf{x}\in[0,1]\times\{0,1\}. \tag{21e}$$

$p:[0,1]^2\to\mathbb{R}$ denotes pressure, $\mathbf{v}:[0,1]^2\to\mathbb{R}^2$ denotes the velocity, $k:[0,1]^2\to\mathbb{R}$ is the spatially-dependent permeability field, and $\mathbf{x}=(x_1,x_2)$ are the spatial coordinates in the horizontal and vertical directions, respectively. The permeability field k is modeled as a lognormal field, $\log k=m\sim\mathcal{N}(0,C)$. The problem of state and parameter estimation for Darcy flow is often considered in data assimilation and in uncertainty quantification, see e.g. [46,47].

The covariance matrix C is derived from the class of Matérn functions [48]:

$$C(\mathbf{x}_i,\mathbf{x}_j)=\sigma^2\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{l}d(\mathbf{x}_i,\mathbf{x}_j)\right)^{\nu}K_\nu\left(\frac{\sqrt{2\nu}}{l}d(\mathbf{x}_i,\mathbf{x}_j)\right). \tag{22}$$

Γ is the gamma function and K_ν is the modified Bessel function of the second kind. $d(\mathbf{x}_i,\mathbf{x}_j)$ denotes the distance between two points, $\mathbf{x}_i$ and $\mathbf{x}_j$, in the domain, ν defines the smoothness, $\sigma^2>0$ is the variance, and $l>0$ is the correlation length.

We denote by $\mathbf{m}_N$ the discretized version of m defined on an $N\times N$ grid and the covariance matrix, $C_N\in\mathbb{R}^{N^2}\times\mathbb{R}^{N^2}$, has elements $(C_N)_{i,j}=C(\mathbf{x}_i,\mathbf{x}_j)$. Then, $\mathbf{m}_N$ can be sampled by computing

$$\mathbf{m}_N=\sum_{i=1}^{N^2}\sqrt{\lambda_i}\hat{\mathbf{m}}_i\boldsymbol{\psi}_i,\quad \hat{\mathbf{m}}\in\mathbb{R}^{N^2},\quad \hat{\mathbf{m}}\sim\mathcal{N}(0,I), \tag{23}$$

where $I\in\mathbb{R}^{N^2}\times\mathbb{R}^{N^2}$ is the identity matrix, λ_i are the eigenvalues of C_N in descending order, and $\boldsymbol{\psi}_i$ the corresponding eigenvectors. Hence, the permeability field is determined by $\hat{\mathbf{m}}_i$, $i=1,\dots,N$. A reduced representation of the permeability field can then be computed by choosing $n<N^2$:

$$\mathbf{m}_N^{(n)}=\sum_{i=1}^{n}\sqrt{\lambda_i}\hat{\mathbf{m}}_i\boldsymbol{\psi}_i. \tag{24}$$

Thereby, the reduced permeability field is determined by n, instead of N^2, parameters.

For generating the training data, Eq. (21) is solved using the finite element method. The velocity is discretized by discontinuous Raviart-Thomas elements of polynomial order 3 and the pressure is discretized by Lagrange elements of polynomial order 2. This is known to be a stable pairing of finite element spaces for the stationary Darcy flow [49]. The domain is divided into 32×32 squares, each divided into two triangles, resulting in 25793 degrees of freedom in total. The solutions are then evaluated on a 50×50 equidistant grid. The implementation is done using the FEniCS library [50].

The specific setting for creating the permeability field here is $n=1089$, $\nu=1.5$, $l=0.2$, and $\sigma=0.5$.

For the observations, we consider evenly distributed sensors at locations, $(\mathbf{x}_1,\dots,\mathbf{x}_{N_y})$, measuring the horizontal velocity at $N_y=100$ discrete points, see Fig. 5a. Thus, $\mathbf{h}:\mathbb{R}^{N\times N}\to\mathbb{R}^{N_y}$, and the measurements are created by:

$$\mathbf{y}=\mathbf{h}(\mathbf{v})+\eta,\quad \mathbf{h}(\mathbf{v})=(v_1(\mathbf{x}_1),\dots,v_1(\mathbf{x}_{N_y}))\quad \eta\sim\mathcal{N}(0,0.01^2I),\quad \eta\in\mathbb{R}^{N_y}. \tag{25}$$

The synthetic observations are generated using 50×50 squares divided into two triangles. The velocity is discretized with polynomial order 4 and the pressure with polynomial order 3. The test case is similar to the one presented in [46].

We compare the MCGAN method with the ensemble Kalman inversion (EKI) method [51] and Deep Bayesian Inversion [27]. The DBI is trained to the specific sensor locations. We do not compare with PCE since it is infeasible to compute a PCE model for a problem of this high dimensionality.

GAN setup

The discriminator of the GAN consists of convolutional layers and the generator consists of transposed convolutional layers. The generator is trained to generate the velocity in the horizontal direction, v_1, the velocity in the vertical direction, v_2, the pressure, p, and the log-permeability field, $\log(k)$. Each quantity is considered a channel in the sense of convolutional neural networks. Thereby, the generator outputs tensors of the shape $(4,N,N)$. To avoid boundary artifacts in the generated fields originating from the transposed convolutional layers, the generator is trained to generate fields of the shape $(4,N+l,N+l)$, $l>0$, which are then cropped to the desired size. For details on the exact architecture specifications, see Fig. C.8.

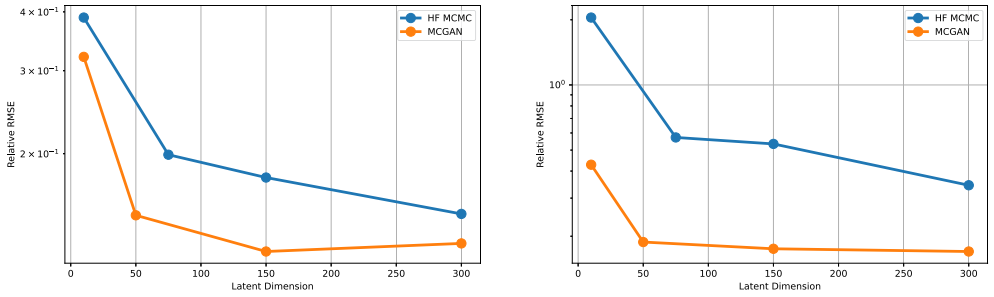
Results

The MCGAN results are computed with a single chain of 20,000 samples, where the first 12,500 samples are discarded to ensure that we only use samples with a converged chain. The MAP estimate is used as the initial MCMC sample, which reduces the time until convergence for the MCMC method significantly.

For the likelihood function, we use $\mathcal{N}(0,0.02^2I)$, which is different from the distribution used to generate the observation noise.

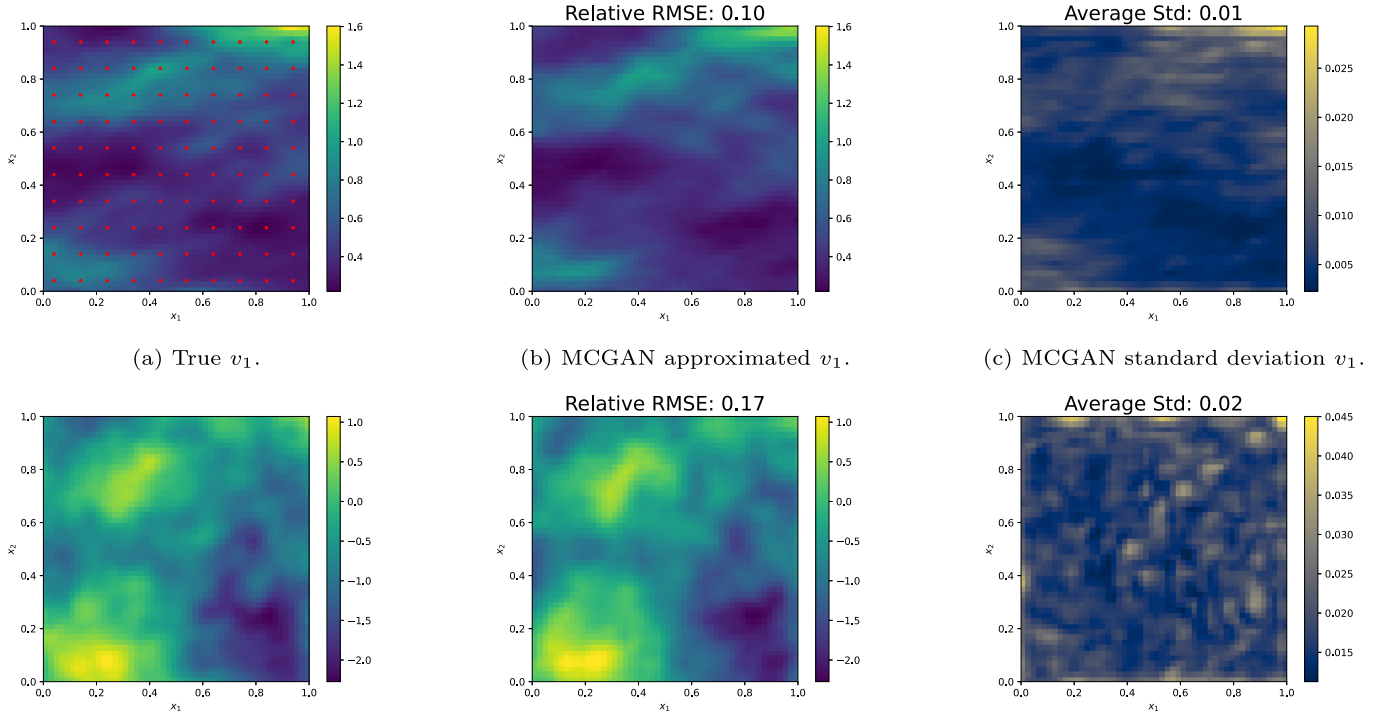
In Fig. 4, the convergence of relative RMSE is presented for the MCGAN approximated state and permeability with respect to the latent dimension. For each latent dimension, a new GAN is trained with the same architecture, hyperparameters, and training data. For comparison, we also compute the convergence of a high-fidelity MCMC procedure.[2] Clearly, with a latent dimension of 10 in the MCGAN, essentially the same accuracy was achieved for the permeability as for the high-fidelity MCMC method with between 150 and 300 modes. Furthermore, for the state, a dimension

[2] We sample the permeability and solve the forward model to get the likelihood. We use the same forward as for simulating the MCGAN training data and the standard deviation was also the same as for the MCGAN. The mean values of all accepted samples were used for computing the relative RMSE of the permeability. Similarly, the mean values of all corresponding states were used to compute the state relative RMSE. The latent dimension refers to the number of modes used for the permeability, as shown in (24). A Metropolis-Hastings algorithm with adaptive proposal standard deviation ensuring that the acceptance rate is between 0.2 and 0.5 was used. We employed 10 uncorrelated chains in parallel, with different initial conditions. The first 200,000 samples from each chain were discarded and then every 5th sample was saved to reduce the correlation between each sample in the chain until 3,000 samples per chain were reached. Hence, a total of 30,000 samples were used for computing the relative RMSE.



(a) State (b) Log-permeability

Fig. 4. Convergence of the MCGAN and high-fidelity MCMC for the state and log-permeability with respect to the latent dimension.



(a) True v_1. (b) MCGAN approximated v_1. (c) MCGAN standard deviation v_1.

(d) True $\log(k)$. (e) MCGAN approximated $\log(k)$. (f) MCGAN standard deviation $\log(k)$.

Fig. 5. Top row: v_1. Red dots are points of measurements. Bottom row: $\log(k)$.

of 50 in the MCGAN achieves similar accuracy as 300 modes in the high-fidelity MCMC. Hence, the MCGAN approach provides a significant dimensionality reduction, together with a speed-up of the evaluation.

In Fig. 5, the results from using MCGAN for the Darcy flow are shown. We see that the horizontal velocity is estimated accurately with a relative RMSE of 0.10 and a relatively low standard deviation. Not surprisingly, the standard deviation seems to be largest at the upper boundary where no measurements are available. Furthermore, larger uncertainty is observed in the areas of the domain where the magnitude v_1 is large.

Regarding the log-permeability, the MCGAN captures the structure of the true log-permeability as well as the sharp edges with a relative RMSE of 0.17.

For both the state and log-permeability, the Kalman inversion gives similar, but slightly worse, accuracy and significantly smoother results than the MCGAN approach (see Fig. D.9). Hence, the Kalman inversion is not able to capture the sharper edges. Furthermore, it is an order of magnitude slower (see Table 4). The DBI method gives similar results, but a lower accuracy on the parameter estimation is observed (see Fig. D.10.

4.2. Leakage detection in pipe flow

To show the method's generality, as a different problem, we consider unsteady single phase flow through a pipeline, until suddenly (at t = 10 s) a leak occurs. As a consequence, pressure waves start propagating through the pipeline, and the velocity field at the leak becomes discontinuous because of the mass flow leaving through the leak. We have only two measurement locations, one close to the inlet and one close to the outlet of the pipeline measuring pressure, and the goal is to infer the leak location and size based on these measurements and a physical model of the flow in the pipeline. This is a challenging problem because of the very sparse measurement data and the discontinuity in the solution.

Table 1
Parameters for the pipe flow equations, (26). Note that the discharge coefficient and the leakage location have values denoted by intervals, as they are the parameters to determine.

Physical quantity	Constant	Value	Unit
Pipe length	L	2000	m
Diameter	d	0.508	m
Cross-sectional area	A	0.203	m^2
Speed of sound in fluid	c	308	m/s
Ambient pressure	p_{amb}	101325	Pa
Reference pressure	p_{ref}	5016390	Pa
Reference density	ρ_{ref}	52.67	kg/m^3
Inflow velocity	v_0	4.0	m/s
Outflow pressure	p_L	5016390	Pa
Pipe roughness	ϵ	10^{-8}	m
Fluid viscosity	μ	$1.2 \cdot 10^{-5}$	$N \cdot s/m^2$
Leakage start time	t_l	10	s
Discharge coefficient	C_d	$[1.0 \cdot 10^{-4}, 9.0 \cdot 10^{-4}]$	m
Leakage location	x_l	[100, 1900]	m

The governing equations are given by the one-dimensional Euler equations for mass and momentum conservation [52,53]:

$$\partial_t q_1 + \partial_x q_2 = C_d \sqrt{\rho(p(\rho) - p_{\text{amb}})}\delta(x - x_l)H(t - t_l), \tag{26a}$$

$$\partial_t q_2 + \partial_x \left(\frac{q_2^2}{q_1} + p(\rho)A \right) = -\frac{1}{2d}\frac{q_2^2}{q_1} f_f(q), \tag{26b}$$

$$v(0,t) = v_0, \quad p(L,t) = p_L, \tag{26c}$$

where ρ is the fluid density (not to be confused with the probability density functions in previous sections), $p(\rho) = c^2(\rho - \rho_0) + p_0$ is the pressure, v is the velocity, $q_1 = \rho A$, $q_2 = \rho v A$, δ is the Dirac delta function, and H is the Heaviside function. v_0 represents the boundary conditions prescribed on the velocity at the left end of the pipe and p_L is the prescribed pressure at the right end of the pipe. d, A, p_{amb}, c, ρ_0, are all constants. The physical quantities they represent and the values we will be working with are found in Table 1. The righthand side in Eq. (26a) is the leakage, modeled as a discharge. t_l is the time at which the leakage occurs, x_l and C_d are the two parameters of interest. They represent the location and size of the leakage, respectively. The righthand side of Eq. (26b) is the friction, where f_f is the Darcy-Weisbach friction coefficient, which is given by the Haaland expression [54]:

$$\frac{1}{\sqrt{f_f}} = -\frac{1}{4} 1.8 \log_{10}\left[\left(\frac{\epsilon/D}{3.7} \right)^{1.11} + \frac{6.9}{Re} \right], \tag{27}$$

where Re is the Reynolds number, $Re = \frac{\rho v d}{\mu}$, with μ the fluid viscosity and ϵ the pipe roughness. The values and units of all parameters in the model are in Table 1. The initial condition is $(q_1, q_2) = (\rho_0 A, \rho_0 v_0 A)$.

Eq. (26) is solved using the nodal discontinuous Galerkin method [55]. We use Legendre polynomials for the modal representation of the local polynomials, and Lagrange polynomials for the nodal representation. The numerical flux is chosen to be the Lax-Friedrichs flux. To ensure stability and non-oscillatory behavior while ensuring high-order accuracy, a TVBM slope-limiter is applied after each time step [55]. The time-stepping is performed using the BDF2 method, with an initial implicit Euler step [56].

For the generation of the training data, we consider 75 elements with a local polynomial order of 3. The resulting solution is then evaluated on an equidistant grid consisting of 256 points. For the time-stepping, we consider a horizon of $T = 64$ seconds with 256 time steps. Hence, $(q_1, q_2) \in \mathbb{R}^{256\times256} \times \mathbb{R}^{256\times256}$.

We assume a uniform prior for both the leakage location, $x_l \sim \mathcal{U}(100, 1900)$, and the discharge coefficient, $C_d \sim \mathcal{U}\left(1.0 \cdot 10^{-4}, 9.0 \cdot 10^{-4}\right)$. Other choices of distributions of x_l and C_d are subject to future studies.

For the state and parameter estimation, only measurements of the pressure are observed. We consider the vector, $(x_1, \dots, x_{N_y})$, of measurement locations, and the vector of measurement times, $(t_1, \dots, t_{N_y})$. This gives rise to the synthetic observations:

$$\mathbf{y} = \mathbf{h}(p) + \eta, \quad \mathbf{h}(p) = (p(x_1, t_1), \dots, p(x_{N_y}, t_{N_y})), \quad \eta \sim \mathcal{N}(0, 1500^2 I), \quad \eta \in \mathbb{R}^{N_y}. \tag{28}$$

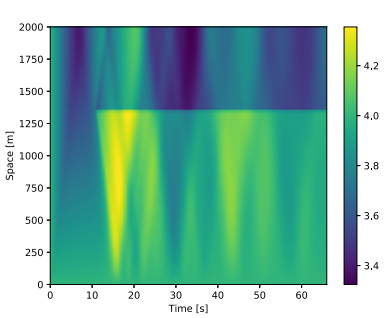
We specifically consider the case where we only observe at $x = 20$ m and at $x = 1980$ m and for all time instances, i.e. $(x_1, \dots, x_{N_y}) = (20, \dots, 20, 1980, \dots, 1980)$ and $(t_1, \dots, t_{N_y}) = (0.25, \dots, 64, 0.25, \dots, 64)$. Hence, $N_y = 2 \cdot 256 = 512$. For simulating the synthetic observations, we used 100 elements with a local polynomial order of 4.

GAN setup

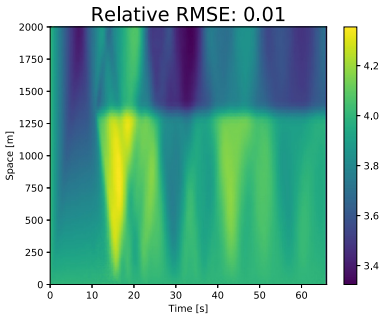
As for the above tests, we use convolutional layers for the discriminator and transposed convolutional layers for the generator. The GAN is trained to generate the velocity, v, and pressure, p, instead of generating the conservative variables q_1 and q_2, since v and p are the quantities of interest. The GAN is trained to generate full space-time solutions in the intervals, $x \in [0, L]$ and $t \in [0, T]$. v and p are considered channels in the sense of convolutional neural networks. Hence, the generator generates tensors of size $(2, 256, 256)$.

At the location of the leakage, there will be a discontinuity in the velocity, due to a drastic drop in the velocity. We use this information to compute the leakage location by identifying the spatial location of the discontinuity, by convolving the state with an appropriate kernel. Furthermore, a dense neural network takes in the generated state and outputs the discharge coefficient. See Fig. C.8 for a visualization of the GAN.

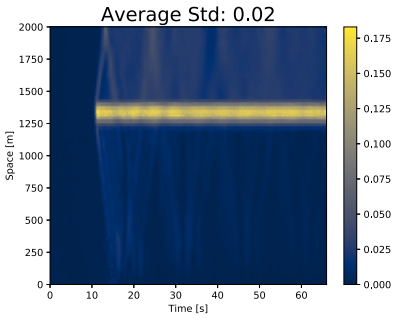
Due to the large differences in orders of magnitude, the velocity, pressure and discharge coefficients are scaled to have values between -1 and 1.



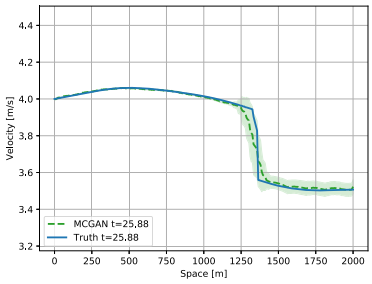
(a) True velocity.



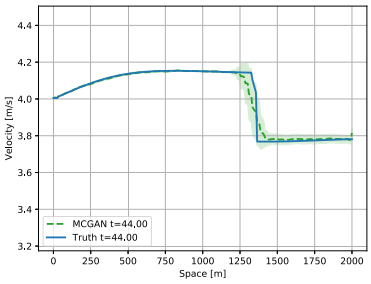
(b) MCGAN approximated velocity.



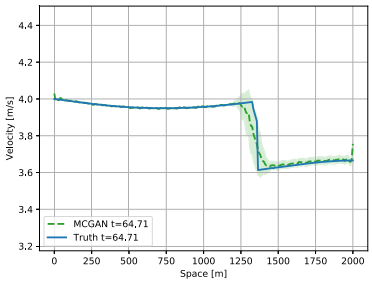
(c) MCGAN velocity standard deviation.



(d) Time 25.88 sec.

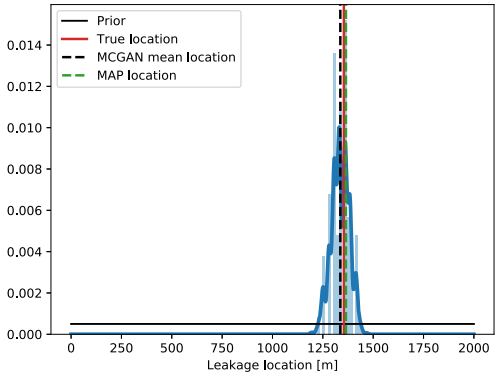


(e) Time 44.00 sec.

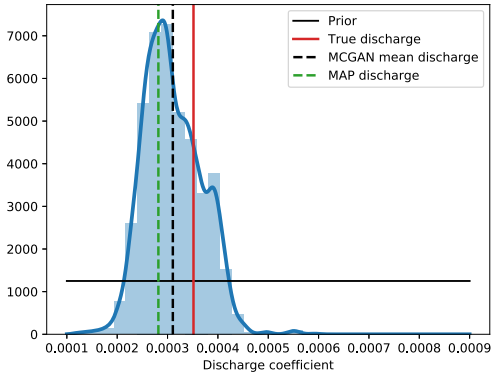


(f) Time 64.71 sec.

Fig. 6. Results for the MCGAN method applied to the pipe flow with a leakage, Eq. (26). (a)-(c) are space-time contour plots of the true state, the MCGAN estimated state, and the standard deviation, respectively. (d)-(f) show the state reconstruction at various instances in time with the shaded area denoting one standard deviation away from the reconstruction.



(a) Leakage location.



(b) Discharge coefficient.

Fig. 7. Posterior distributions of the leakage location and discharge coefficient in the pipe flow equation.

Results

The MCGAN results are computed with a single chain of 15,000 samples, where the first 10,000 samples are discarded to ensure that we only use samples after the chain has converged. The MAP estimate is, again, used as the initial MCMC sample in order to speed up convergence. For the likelihood function, we use $\mathcal{N}(0, 3000^2 I)$, which is different from the distribution used to generate the observation noise.

Fig. 6 presents the reconstruction of the velocity. It is apparent that the velocity is reconstructed very well with a relative RMSE of 0.01. It is especially worth noting that the uncertainty is largest around the drop in velocity, i.e. at the location of the leakage, as expected. This uncertainty information could further be used to estimate the location of the leakage. While the state estimation is accurate, it is apparent that the velocity estimation is slightly worse in the domain to the right of the leakage ($x > x_l$). The lack of accuracy is accompanied by an increased standard deviation in that part of the domain. Hence, the uncertainty estimates provide useful information.

While the MCGAN is performing well in the interior of the domain, it is noteworthy that the estimation at the boundary at $x = 2000$ is not as accurate.

In Fig. 7, we see the estimated posterior distributions of the leakage location and discharge coefficient, respectively. In the leakage location posterior, the estimated mean is close to the true mean (see also Table 2) and it is, more or less, symmetric. In the discharge coefficient posterior, on the other hand, the estimated value appears to be smaller than the true value. The MCGAN method significantly outperforms the PCE and EnKF methods in this case. The EnKF is initiated with $x_l = 1000$ and $C_d = 5 \cdot 10^{-4}$ and the MCMC with PCE is initiated at the MAP estimate. In Fig. D.11 and D.12, the results obtained using the EnKF and PCE method are shown. None of the two approaches manages to estimate the state or the parameters

Table 2
Estimated parameters for the pipe flow using MCGAN, PCE, and EnKF. The best estimates are highlighted in boldface.

Pars	True val.	MCGAN		PCE		EnKF		DBI	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
x_l	1354.5	1336.8	44.80	1099.2	69.98	1005.0	10.15	**1357.2**	35.7
C_d	$3.5{\cdot}10^{-4}$	$\mathbf{3.1{\cdot}10^{-4}}$	$5.7{\cdot}10^{-5}$	$2.6{\cdot}10^{-4}$	$2.4{\cdot}10^{-5}$	$7.2{\cdot}10^{-4}$	$3.2{\cdot}10^{-3}$	$2.7{\cdot}10^{-4}$	$1.2{\cdot}10^{-5}$

Table 3
Relative RMSE for the state and parameter estimation for the various test cases. For the Darcy flow, the Relative RMSE for (v_1, v_2, p) is computed. For the pipe flow, the Relative RMSE for (u, p) is computed. The best performing cases are highlighted in boldface.

	MCGAN		PCE		EnKF/EKI		DBI	
	State	Pars	State	Pars	State	Pars	State	Pars
Darcy flow	$\mathbf{1.3{\cdot}10^{-1}}$	$\mathbf{1.7{\cdot}10^{-1}}$	-	-	$2.0{\cdot}10^{-1}$	$4.2{\cdot}10^{-1}$	$1.9{\cdot}10^{-1}$	$4.9{\cdot}10^{-1}$
Pipe flow	$\mathbf{4.8{\cdot}10^{-3}}$	$1.3{\cdot}10^{-2}$	$1.3{\cdot}10^{-2}$	$1.9{\cdot}10^{-1}$	$3.3{\cdot}10^{-2}$	$2.6{\cdot}10^{-1}$	$7.0{\cdot}10^{-3}$	$\mathbf{2.0{\cdot}10^{-3}}$

Table 4
Comparison of online computation time. All simulations are run on CPU cores. Only the number of CPU cores varies.

	MCGAN (1 core)	PCE (20 cores)	EnKF/EKI (20 cores)	DBI (1 core)
Darcy flow	$3.17{\cdot}10^{2}$ s	-	$4.11{\cdot}10^{3}$ s	$3.31{\cdot}10^{1}$ s
Pipe flow	$7.10{\cdot}10^{2}$ s	$1.18{\cdot}10^{4}$ s	$1.25{\cdot}10^{4}$ s	$3.14{\cdot}10^{1}$ s

in a satisfying manner. Table 2 shows that the EnKF approach is unable to update the posterior and the PCE approach only performs marginally better.

The pipe flow equations are highly nonlinear and the solution exhibits a discontinuity at the location of the leakage. Both phenomena are not easy to handle with the PCE nor EnKF approaches, while neural networks have been shown to be well-suited for such tasks.

On the other hand, DBI performs highly satisfactory (see Fig. D.13). For the leak location, DBI is slightly more accurate, while for the leak size, MCGAN performs better. Furthermore, the distribution over the leak size computed by DBI is narrow, suggesting that the approximated value has small uncertainty associated with it, even though the approximation is not accurate. MCGAN, on the other hand, shows larger uncertainty associated with the approximation suggesting more accurate evaluation of the reliability. Lastly, we see that MCGAN approximates the state more accurately than DBI.

As mentioned, when using DBI, one has to choose the exact location and temporal frequency of incoming observations before training, to create the training set. This is not the case for MCGAN, where the exact sensor configuration and likelihood do not have to be specified before the online stage.

4.3. Summary of results

To summarize the results obtained using the proposed MCGAN method, we highlight accuracy and computation time. Firstly, in Table 3 the relative RMSE for the state and parameters are presented for both test cases. The MCGAN performs better than the three alternative approaches in almost all metrics. For the leakage localization in the pipe flow test case, the MCGAN method outperforms the PCE and the EnKF approaches, while performing similarly to DBI. The MCGAN results are very close to the true values with relative RMSEs that are one order of magnitude better than PCE and ensemble Kalman approaches for the state and parameter estimation. In Table 4, the online computation times for the methods applied to the two test cases are shown. It is interesting to note that the computation time does not change much in the two test cases for the MCGAN. This is due to the fact that there are only minor differences in computation time between evaluating a small neural network and a large one. DBI is the fastest approach since the GAN is trained to sample directly from the posterior, in contrast to the MCGAN approach that makes use of MCMC methods.

Lastly, we briefly comment on the offline training time. For the computationally most expensive case, the pipe flow, the most time consuming part is the generation of data. Generating 100,000 training trajectories took about 80 hours on 30 CPU cores (90 seconds per trajectory). The training of the GAN was finished in about 24 hours for both the MCGAN and DBI. In total, the offline stage took approximately 104 hours. The offline time for the Darcy flow was shorter, totaling around 50 hours. We did not experience high sensitivity to the hyperparameters, such as learning rate, batch size, etc. This might be a product of the large number of training samples.

5. Conclusion

We have presented a new method, named MCGAN, to efficiently and accurately solve Bayesian inverse problems in physics and engineering applications. The method combines Generative Adversarial Networks and Markov Chain Monte Carlo methods to sample from posterior distributions by utilizing a low-dimensional latent space and a push-forward map defined as a neural network.

The methodology is divided into two distinct stages, an offline stage, in which the GAN is trained on simulated training data in order to learn the prior distribution, and an online stage, in which the inverse problem is solved for a new set of observations. While the offline stage potentially takes significant computational time, the online stage is computationally very fast and efficient.

We presented a proof of theoretical convergence of the posterior distribution in the Wasserstein-1 distance, in the case where the GAN would be perfectly trained. Furthermore, we provided the insight that sampling from the latent space yields essentially the same results as sampling from the high-dimensional space, in a weak sense.

To showcase the method's performance, we applied it to two computational engineering test cases with different characteristics and compared it to three alternative approaches. In the high-dimensional problem, the Darcy flow with uncertain permeability field, an improved accuracy was found with MCGAN compared with EKI and DBI, as well as a speed-up of one order of magnitude compared to the EKI method. In the second test case, the leakage localization for flow in a pipe, the MCGAN approach showed increased accuracy for the state and leak size detection, while DBI was slightly more accurate regarding the leak location. Out of the four approaches, the MCGAN method is the one that provided accurate results, fast sampling without being trained to only work for a single sensor configuration.

It is worth noting that, similar to any surrogate modeling approach, it is unclear how well the methodology performs on out-of-distribution cases. This is a subject to future study.

While the MCGAN approach performed well on the two test cases, there is still room for future research. We believe the offline stage can be improved by identifying optimal ways of simulating training data and determining hyperparameters for the GAN. This includes determining the optimal size of the latent space. Furthermore, the GAN can be improved by incorporating physics knowledge either in the training or directly in the neural network architecture. This could possibly alleviate the boundary estimation problems. Also, possibilities of using the MCGAN framework in a sequential fashion, as is the case for Kalman filters, would be an interesting direction to explore.

In conclusion, we believe that the MCGAN methodology can form an important piece of the puzzle towards a well performing digital twin framework, in which real-time state and parameter estimation is of crucial importance.

CRediT authorship contribution statement

N. Mücke: Conceptualization, methodology, software, formal analysis, writing - original draft. **B. Sanderse:** Writing -review & editing, formal analysis. **S. Bohté:** Funding acquisition, writing -review & editing, supervision. **C. Oosterlee:** Funding acquisition, formal analysis, writing -review & editing, supervision, project administration.

Data availability

No data was used for the research described in the article.

Acknowledgement

This work is supported by the Dutch National Science Foundation NWO under the grant number 629.002.213, which is a cooperative project with IISC Bangalore and Shell Research as project partners. The authors furthermore acknowledge fruitful discussions with Dr. W. Edeling from CWI Amsterdam.

Appendix A. Proof of Theorem 1

Proof. Firstly, since neural networks with continuous activation functions are continuous, they are also measurable [41]. Therefore, the generator defines a push forward distribution and Eq. (15) is applicable.

Secondly, we look at the evidence. Assuming the likelihood is measurable with respect to $\mathbf{u}$, we have from Eq. (15):

$$Q_u(\mathbf{y}) = \int_{\mathbb{R}^{N_u}} \rho^g_{y|u}(\mathbf{y}|\mathbf{u})\rho^g_0(\mathbf{u})\,\mathrm{d}\mathbf{u} = \int_{\mathbb{R}^{N_z}} \rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})\,\mathrm{d}\mathbf{z} = Q_z(\mathbf{y}). \tag{A.1}$$

Consider the expected value of the likelihood times some measurable function, f, with respect to the prior:

$$\begin{aligned}\mathbb{E}_{U\sim P^g_0}[f(U)\rho^g_{y|u}(\mathbf{y}|U)] &= \int_E \underbrace{f(\mathbf{u})\rho^g_{y|u}(\mathbf{y}|\mathbf{u})}_{=\xi(\mathbf{u})}\rho^g_0(\mathbf{u})\,\mathrm{d}\mathbf{u}\\ &= \int_{G_\theta^{-1}(E)} \underbrace{f(G_\theta(\mathbf{z}))\rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))}_{=\xi(G_\theta(\mathbf{z}))}\rho^g_z(\mathbf{z})\mathrm{d}\mathbf{z}\\ &= \mathbb{E}_{U\sim P^g_z}[f(G_\theta(Z))\rho^g_{y|u}(\mathbf{y}|G_\theta(Z))].\end{aligned} \tag{A.2}$$

Note that ξ is the product of two measurable functions and is therefore measurable. Hence, Eq. (15) applies. Now using Eq. (A.1) and (A.2) we get:

$$\begin{aligned}\mathbb{E}_{U\sim P^g_{u|y}}[f(U)] &= \frac{1}{Q_u(\mathbf{y})}\mathbb{E}_{U\sim P^g_0}[f(U)\rho^g_{y|u}(\mathbf{y}|U)]\\ &= \frac{1}{Q_z(\mathbf{y})}\mathbb{E}_{Z\sim P^g_z}[f(G_\theta(Z)\rho^g_{y|u}(\mathbf{y}|G_\theta(Z))]\\ &= \mathbb{E}_{Z\sim P^g_{z|y}}[f(G(Z))].\quad \square\end{aligned}$$

Appendix B. Proof of Theorem 2

Proof. We write the Wasserstein-1 distance between the real prior and the generated prior in dual form:

$$W_1(P_0^r, P_0^r) = \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u})\rho_0^r(\mathbf{u})\mathrm{d}\mathbf{u} - \int_E f(\mathbf{u})\rho_0^g(\mathbf{u})\mathrm{d}\mathbf{u} \right|, \tag{B.1}$$

where $f : E \to \mathbb{R}$ is Lipschitz continuous with Lipschitz constant less or equal 1 and $f(\mathbf{u}_0) = 0$ for some $\mathbf{u}_0$. Note that any function, g, with Lipschitz constant less than or equal 1, is a contraction and therefore admits a fixed point. Now, assuming that $\mathbf{u}_0$ is the fixed point, we can simply define $f = g - \mathbf{u}_0$, which admits $f(\mathbf{u}_0) = \mathbf{u}_0$. Therefore, assuming $f(\mathbf{u}_0) = 0$ for some $\mathbf{u}_0$ is not a restriction. Furthermore, we have:

$$|f(\mathbf{u})| = |f(\mathbf{u}) + f(\mathbf{u}_0) - f(\mathbf{u}_0)| = |f(\mathbf{u}) + f(\mathbf{u}_0)| \leq \mathrm{Lip}(f) d(\mathbf{u}, \mathbf{u}_0) \leq D.$$

The Wasserstein-1 distance between the posteriors is given by:

$$\begin{aligned} W_1(P_{u|y}^r, P_{u|y}^g) &= \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u})\rho_{u|y}^r(\mathbf{u}|\mathbf{y})\mathrm{d}\mathbf{u} - \int_E f(\mathbf{u})\rho_{u|y}^g(\mathbf{u}|\mathbf{y})\mathrm{d}\mathbf{u} \right| \\ &= \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u})(\rho_{u|y}^r(\mathbf{u}|\mathbf{y}) - \rho_{u|y}^g(\mathbf{u}|\mathbf{y}))\mathrm{d}\mathbf{u} \right| \\ &= \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})\rho_0^r(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^g(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^g(\mathbf{y})} \right) \mathrm{d}\mathbf{u} \right|. \end{aligned}$$

Adding and subtracting the term $f(\mathbf{u})\frac{\Phi^r(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^r(\mathbf{y})}$ gives

$$\begin{aligned} W_1(P_{u|y}^r, P_{u|y}^g) &= \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})\rho_0^r(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^g(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^g(\mathbf{y})} + \frac{\Phi^r(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^r(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^r(\mathbf{y})} \right) \mathrm{d}\mathbf{u} \right| \\ &\leq \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u}) \frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} \left(\rho_0^r(\mathbf{u}) - \rho_0^g(\mathbf{u}) \right) \mathrm{d}\mathbf{u} \right| + \left| \int_E f(\mathbf{u})\rho_0^g(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^g(\mathbf{u})}{Q_u^g(\mathbf{y})} \right) \mathrm{d}\mathbf{u} \right|. \end{aligned}$$

Subsequently, adding and subtracting the term $f(\mathbf{u})\frac{\Phi^r(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^g(\mathbf{y})}$ in the second integral gives

$$\begin{aligned} W_1(P_{u|y}^r, P_{u|y}^g) \leq & \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u}) \frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} \left(\rho_0^r(\mathbf{u}) - \rho_0^g(\mathbf{u}) \right) \mathrm{d}\mathbf{u} \right| \\ & + \left| \int_E f(\mathbf{u})\rho_0^g(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^g(\mathbf{u})}{Q_u^g(\mathbf{y})} \right) + f(\mathbf{u})\rho_0^g(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})}{Q_u^g(\mathbf{y})} - \frac{\Phi^r(\mathbf{u})}{Q_u^g(\mathbf{y})} \right) \mathrm{d}\mathbf{u} \right| \\ \leq & \sup_{\mathrm{Lip}(f)\leq 1} \underbrace{\left| \int_E f(\mathbf{u}) \frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} \left(\rho_0^r(\mathbf{u}) - \rho_0^g(\mathbf{u}) \right) \mathrm{d}\mathbf{u} \right|}_{=I_1} \\ & + \underbrace{\left| \int_E \left(\frac{1}{Q_u^r(\mathbf{y})} - \frac{1}{Q_u^g(\mathbf{y})} \right) f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u} \right|}_{=I_2} \\ & + \underbrace{\left| \int_E \frac{1}{Q_u^g(\mathbf{y})} f(\mathbf{u})\rho_0^g(\mathbf{u}) \left(\Phi^r(\mathbf{u}) - \Phi^g(\mathbf{u}) \right) \mathrm{d}\mathbf{u} \right|}_{=I_3}. \end{aligned}$$

We will consider I_1, I_2, and I_3 individually. Starting with I_3, we use that

$$|e^{-x_1} - e^{-x_2}| \leq e^{-\min(x_1,x_2)}|x_1 - x_2| \Rightarrow |\Phi^r(\mathbf{u}) - \Phi^g(\mathbf{u})| \leq \max_{\mathbf{u}}(\Phi^r, \Phi^g)|l^r(\mathbf{u}) - l^g(\mathbf{u})|. \tag{B.2}$$

Using Eq. (B.2) and the fact that $|f(\mathbf{u})| \leq d(\mathbf{u}, \mathbf{u}_0)$, together with the Cauchy-Schwartz inequality, we get:

$$\sup_{\mathrm{Lip}(f)\leq 1} I_3 \leq \sup_{\mathrm{Lip}(f)\leq 1} \frac{\max(\Phi^r, \Phi^g)}{Q_u^g(\mathbf{y})} \left| \int_E f(\mathbf{u})\rho_0^g(\mathbf{u})|l^r(\mathbf{u}) - l^g(\mathbf{u})|\mathrm{d}\mathbf{u} \right|$$

$$\leq \frac{\max(\Phi^r,\Phi^g)}{Q_u^g(\mathbf{y})}\left|\int_E d(\mathbf{u},\mathbf{u}_0)\rho_0^g(\mathbf{u})|l^r(\mathbf{u})-l^g(\mathbf{u})|\mathrm{d}\mathbf{u}\right|$$
$$\leq \frac{\max(\Phi^r,\Phi^g)}{Q_u^g(\mathbf{y})}\left(\int_E d(\mathbf{u},\mathbf{u}_0)^2\rho_0^g(\mathbf{u})\mathrm{d}\mathbf{u}\right)^{1/2}||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^2_{\rho_0^g}}$$
$$\leq \underbrace{\frac{\max(\Phi^r,\Phi^g)}{Q_u^g(\mathbf{y})}|P_0^g|_{\mathcal{W}_2}}_{=C_3}||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^2_{\rho_0^g}}.$$

Considering I_2, we use the following [42]:

$$\left|\frac{1}{Q_u^r(\mathbf{y})}-\frac{1}{Q_u^g(\mathbf{y})}\right|=\frac{|Q_u^g(\mathbf{y})-Q_u^r(\mathbf{y})|}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}, \tag{B.3}$$

and

$$|Q_u^g(\mathbf{y})-Q_u^r(\mathbf{y})|\leq\max(\Phi^r,\Phi^g)||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^1_{\rho_0^g}}, \tag{B.4}$$

in order to get:

$$I_2=\left|\int_E\left(\frac{1}{Q_u^r(\mathbf{y})}-\frac{1}{Q_u^g(\mathbf{y})}\right)f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u}\right|$$
$$\leq\left|\frac{1}{Q_u^r(\mathbf{y})}-\frac{1}{Q_u^g(\mathbf{y})}\right|\left|\int_E f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u}\right|$$
$$\leq\frac{|Q_u^g(\mathbf{y})-Q_u^r(\mathbf{y})|}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}\left|\int_E f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u}\right|$$
$$\leq\frac{\max(\Phi^r,\Phi^g)}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}\left|\int_E f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u}\right|||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^1_{\rho_0^g}}$$

By defining the function, $g:E\to\mathbb{R}$, $g(\mathbf{u})=f(\mathbf{u})\Phi^r(\mathbf{u})$, one can show that g is Lipschitz continuous with Lipschitz constant $\mathrm{Lip}(g)=1+D\mathrm{Lip}(\Phi^r)$ [42]. Furthermore, we have $|g(\mathbf{u})|\leq d(\mathbf{u},\mathbf{u}_0)$. This gives:

$$\sup_{\mathrm{Lip}(f)\leq1}I_2\leq\frac{\max(\Phi^r,\Phi^g)}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}(1+D\mathrm{Lip}(\Phi^r))\left|\int_E d_E(\mathbf{u},\mathbf{u}_2)\rho_0^g(\mathbf{u})\mathrm{d}\mathbf{u}\right|||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^1_{\rho_0^g}}$$
$$\leq\underbrace{\frac{\max(\Phi^r,\Phi^g)}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}(1+D\mathrm{Lip}(\Phi^r))|P_0^g|_{\mathcal{W}_1}}_{=C_2}||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^1_{\rho_0^g}}.$$

Finally, we consider I_1. By using the function, $g:E\to\mathbb{R}$, $\mathbf{u}\mapsto f(\mathbf{u})\Phi^r(\mathbf{u})$, as defined above, we get:

$$\sup_{\mathrm{Lip}(g)\leq1}I_1\leq\sup_{\mathrm{Lip}(f)\leq1}\frac{(1+D\mathrm{Lip}(\Phi^r))}{Q_u^r(\mathbf{y})}\left|\int_E g(\mathbf{u})\left(\rho_0^r(\mathbf{u})-\rho_0^g(\mathbf{u})\right)\mathrm{d}\mathbf{u}\right|$$
$$=\underbrace{\frac{(1+D\mathrm{Lip}(\Phi^r))}{Q_u^r(\mathbf{y})}}_{=C_1}W_1(P_0^r,P_0^r)$$

Combining, I_1, I_2, and I_3 we then get:

$$W_1(P_{u|y}^r,P_{u|y}^g)\leq\sup_{\mathrm{Lip}(f)\leq1}I_1+I_2+I_3$$
$$=C_1W_1(P_0^r,P_0^r)+C_2||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^1_{\rho_0^g}}+C_3||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^2_{\rho_0^g}}$$
$$\leq C_1\epsilon_1+C_2\epsilon_2+C_3\epsilon_3.\quad\square$$

Appendix C. Training Wasserstein GANs

In the WGAN framework, it is important to properly train the discriminator. Therefore, it is common practice to update the discriminator parameters more frequently than the generator parameters. The number of discriminator updates, relative to those of the generator, is denoted by n_{disc}/n_{gen}. The hyperparameters for the training of the three test cases are shown in Table C.5. The specific architectures used are shown in Fig. C.8.

Table C.5
Hyperparameters for the WGANs for the three test cases.

Hyperparameters \ Test case	Darcy flow	Pipe flow
Optimizer	RMSProp	RMSProp
Learning rate	10^{-4}	10^{-4}
Batch size	64	64
Gradient penalty	5	5
n_{disc}/n_{gen}	1	2
N_{train}	300,000	100,000
Latent dimension (N_z)	150	50

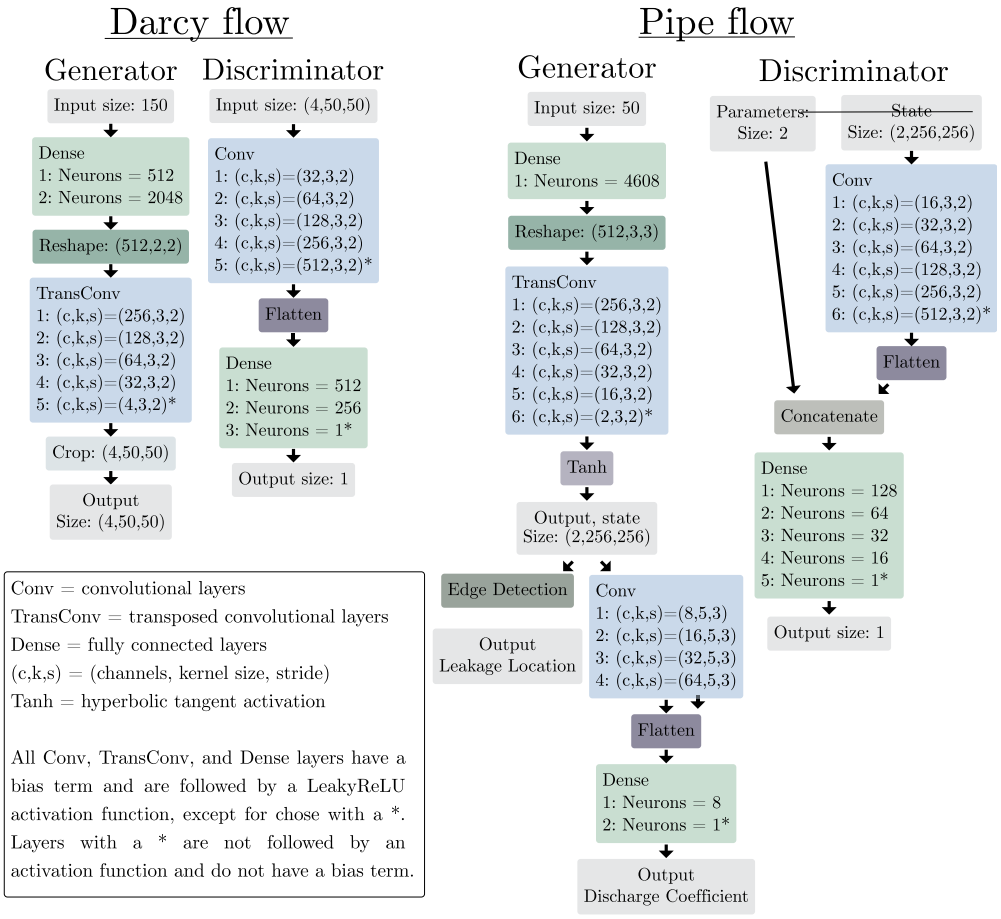


Fig. C.8. Generator and discriminator architectures for the two test cases.

We compared the Adam optimizer and the RMSprop optimizer for training, and found that RMSProp, in general, showed superior results in our test cases.

Appendix D. Alternative methods

D.1. Ensemble Kalman filter

We make use of two variations of the ensemble Kalman filter (EnKF):

- The (standard) EnKF for dynamic problems, where the state and parameter distributions are computed based on previous time steps along with data availability;
- Ensemble Kalman Inversion (EKI), used for stationary problems, where an artificial time dimension is introduced in order to iteratively update the posterior of the state and parameters.

For the pipe flow equations the standard EnKF is utilized while for the Darcy flow the EKI is used. The EnKF implementation is based on [2] and the EKI implementation is based on [51].

For simultaneously estimating the parameter and state, we make use of disturbance modeling [57]. Here, we define an augmented model:

$$\mathbf{q}_i = F(\mathbf{q}_{i-1}, \mathbf{m}_{i-1}) + \Gamma_q \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, Q_q), \tag{D.1a}$$

$$\mathbf{m}_i = \mathbf{m}_{i-1} + \Gamma_m \delta_i, \quad \delta_i \sim \mathcal{N}(0, Q_m), \tag{D.1b}$$

$$\mathbf{y}_i = \mathbf{h}(\mathbf{q}_i) + \eta_i, \quad \eta_i \sim \mathcal{N}(0, R), \tag{D.1c}$$

where F is the discrete one-step time advancement model, ϵ_i is the model noise, Q the model covariance, and R the observation covariance. With this formulation, both the state and parameters are updated in every step of the EnKF algorithm.

D.2. Polynomial chaos expansion

The basic idea behind polynomial chaos expansion (PCE) is to create a surrogate model that maps the stochastic parameters, $\mathbf{m}$, to a quantity of interest, Q [9]. The surrogate model is defined by a linear expansion of orthogonal polynomials:

$$Q(\mathbf{m}) = \sum_{i=1}^{N} \alpha_i \phi_i(\mathbf{m}), \tag{D.2}$$

where ϕ_i are the polynomials that are chosen based on the distribution of $\mathbf{m}$, and α_i are the generalized Fourier coefficients.

The coefficients, α_i, are typically computed using either spectral projection methods or by least squares minimization. In both cases, the evaluations are carefully chosen according to a quadrature rule.

In our test cases, we choose the quantity of interest to be the observations, i.e. $Q(\mathbf{m}) \approx \mathbf{h}(\mathbf{q}(\mathbf{m}))$ and $\alpha_i \in \mathbb{R}^{N_y}$. $\mathbf{m}$ are the parameters of interest, which are often the model parameters and/or initial and boundary conditions.

When the PCE is computed, the posterior PDF is defined by:

$$\rho_m^y(\mathbf{m}|\mathbf{y}) = \frac{1}{\rho_y(\mathbf{y})} \rho_\eta(\mathbf{y} - Q(\mathbf{m})) \rho_0^m(\mathbf{m}). \tag{D.3}$$

The expected state and parameters are then computed by:

$$\mathbb{E}_{\mathbf{q}}[\mathbf{q}] \approx \frac{1}{N_{sample}} \sum_{i=1}^{N_{sample}} \mathbf{q}(\mathbf{m}_i), \quad \mathbb{E}_{\mathbf{m}}[\mathbf{m}] \approx \frac{1}{N_{sample}} \sum_{i=1}^{N_{sample}} \mathbf{m}_i, \quad \mathbf{m}_i \sim P_m^y, \tag{D.4}$$

and the variance is computed in a similar manner.

It is important to notice that the sampling is done in the parameter space and the state is thereafter computed by using the sampled parameters as input for the forward problem. Directly sampling the state is infeasible due to the high-dimensionality of the state.

The implementation of the PCE method is done using the Python library Chaospy [58].

D.3. Deep Bayesian inversion

Deep Bayesian Inversion (DBI) makes use of conditional GANs (CGANs) to learn the posterior distribution [27]. A CGAN is trained to learn to sample from the posterior distribution, $P_{u|y}$, directly. The general setup is the same as in Section 2.4, but with some minor differences. The generator is now a map that takes a latent vector, $\mathbf{z}$, and observations, $\mathbf{y}$, and outputs a sample $\mathbf{u}$:

$$G(Z, \mathbf{y}) = P_{u|y}^g(U|\mathbf{y}) \approx P_{u|y}^r(U|\mathbf{y}), \quad Z \sim P_z^g. \tag{D.5}$$

Similarly, the discriminator takes $\mathbf{y}$ and $\mathbf{u}$ and outputs a real number.

The training is performed by solving the inf-sup problem:

$$\begin{aligned} \inf_\theta \sup_\omega \quad & \mathbb{E}_{X \sim P_u^r}\left[D_\omega(X, \mathbf{h}(X))\right] - \mathbb{E}_{Z \sim P_z^g}\left[D_\omega(G_\theta(Z), \mathbf{h}(G_\theta(Z)))\right] \\ & -\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim P_{\hat{X}}}\left[\left(||\nabla_{\hat{X}} D_\omega(\hat{X}, \mathbf{h}(\hat{X}))|| - 1\right)^2\right]. \end{aligned} \tag{D.6}$$

Hence, the training is performed with the same data as for the MCGAN training, with the difference that the observation operator is used to create observations for training.

In the online stage, when observations become available, one samples several latent vectors for the same observations and uses those to obtain posterior samples:

$$\left\{G(\mathbf{z}_1, \mathbf{y}), G(\mathbf{z}_2, \mathbf{y}), \ldots, G(\mathbf{z}_N, \mathbf{y})\right\} = \left\{\mathbf{u}_1|\mathbf{y}, \mathbf{u}_2|\mathbf{y}, \ldots, \mathbf{u}_N|\mathbf{y}\right\}, \quad \mathbf{z}_i \sim P_z^g, \quad \forall i. \tag{D.7}$$

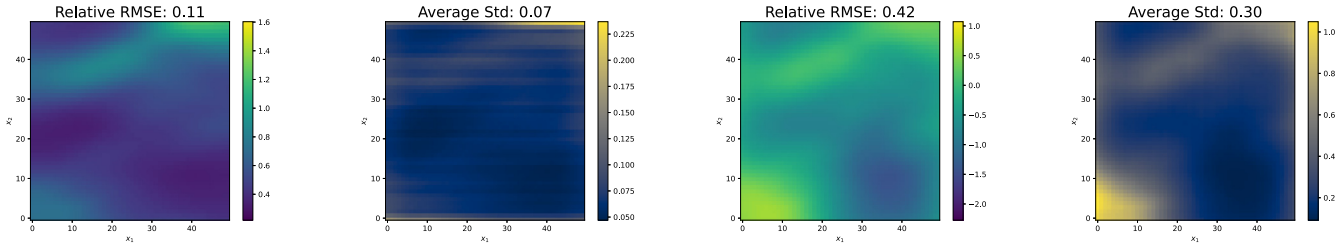
The generator is now tied to the observation operator that was used for training.

D.4. Darcy flow

For the Darcy flow, we compared the MCGAN results with EKI, since it is infeasible to compute high-dimensional distributions with PCE. We compute ensembles consisting of 4000 forward computations and use 25 iterations. Note that the EKI method is parallel since each member of the ensemble can be computed independently from the other members. Therefore, we run the EKI using 20 CPU cores. For computing the permeability field, we use $n = 1089$, which is the total number of degrees of freedom. See Fig. D.9 for the results.

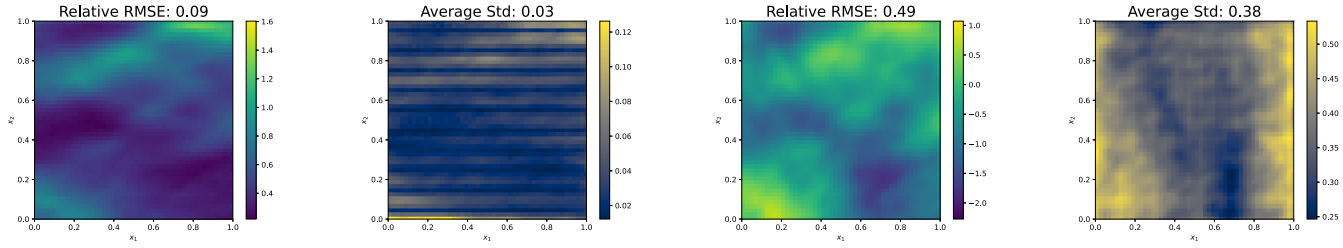
D.5. Leakage detection in pipe flow

For the leakage detection in the pipe, we compare our method with the PCE and the EnKF approaches. The PCE model is trained to map the leakage location, x_l, and discharge coefficient, C_d, to the observations. We achieved the highest precision with fourth-order polynomials. We performed 50,000 MCMC posterior samples and discarded the first 40,000. The state reconstruction is performed after the sampling by computing



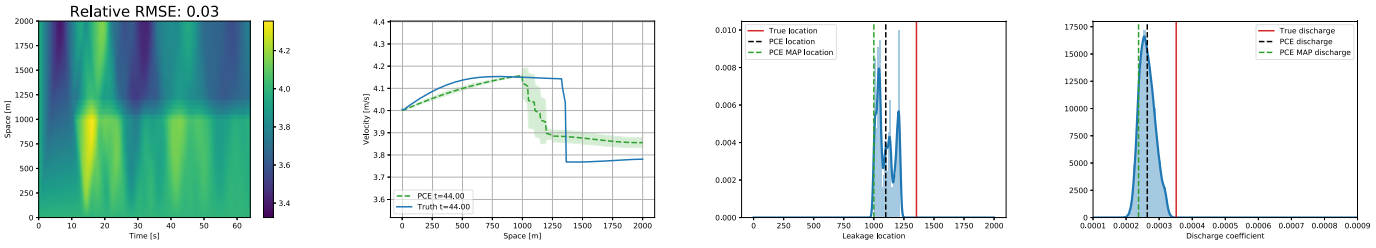
(a) EKI approximated v_1. (b) EKI standard deviation v_1. (c) EKI approximated $\log(k)$. (d) EKI standard deviation $\log(k)$.

Fig. D.9. In (a)-(b) we see the reconstruction of v_1 and the standard deviation of the reconstruction, respectively. In (c)-(d) we see the reconstruction of $\log(k)$ and the standard deviation of the reconstruction, respectively.



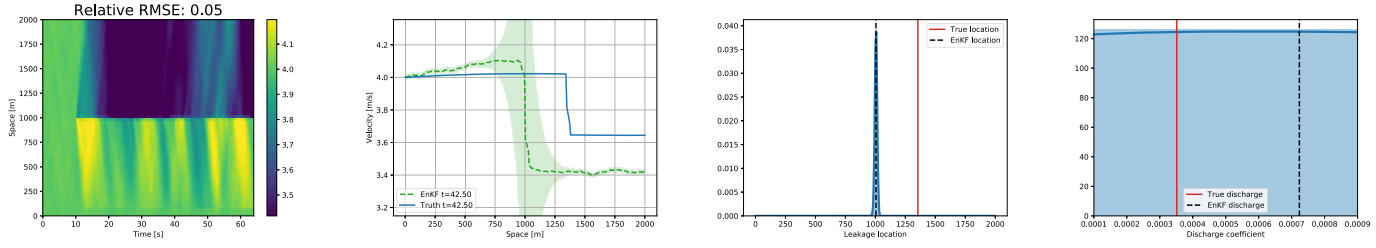
(a) DBI approximated v_1. (b) DBI standard deviation v_1. (c) DBI approximated $\log(k)$. (d) DBI standard deviation $\log(k)$.

Fig. D.10. In (a)-(b) we see the reconstruction of v_1 and the standard deviation of the reconstruction, respectively. In (c)-(d) we see the reconstruction of $\log(k)$ and the standard deviation of the reconstruction, respectively.



(a) PCE approximated velocity. (b) Reconstruction at $t = 44$ (c) Leakage location. (d) Discharge coefficient.

Fig. D.11. Results for the MCMC sampling with a PCE surrogate model applied to the pipe flow with a leakage, Eq. (26). In (a) we see the space-time contour plots of the reconstructed velocity. In (b) we see the velocity reconstruction at $t = 44$ with the shaded area denoting the standard deviation. In (c)-(d) we see the posterior distributions of the leakage location and discharge coefficient.

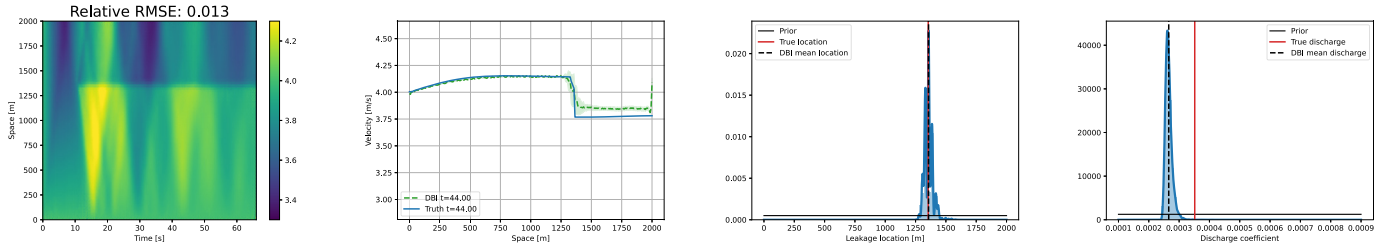


(a) EnKF approximated velocity. (b) Reconstruction at $t = 44$ (c) Leakage location. (d) Discharge coefficient.

Fig. D.12. Results for EnKF method applied to the pipe flow with a leakage, Eq. (26). In (a) we see the space-time contour plots of the reconstructed velocity. In (b) we see the velocity reconstruction at $t = 44$ with the shaded area denoting the standard deviation. In (c)-(d) we see the posterior distributions of the leakage location and discharge coefficient.

the state using the parameters samples from the MCMC sampling. The state reconstructions are computed in parallel using 20 cores. See Fig. D.11 for results.

In the EnKF method we used an ensemble size of 2000. Γ_q and Γ_m are chosen to be identity matrices and $Q_q = \text{diag}(0.01, 0.001)^2$ and $Q_m = \text{diag}(100, 1 \cdot 10^5)^2$. The ensemble is computed in parallel on 20 CPU cores. See Fig. D.12 for results.



(a) DBI approximated velocity. (b) Reconstruction at $t = 44$ (c) Leakage location. (d) Discharge coefficient.

Fig. D.13. Results for DBI method applied to the pipe flow with a leakage, Eq. (26). In (a) we see the space-time contour plots of the reconstructed velocity. In (b) we see the velocity reconstruction at $t = 44$ with the shaded area denoting the standard deviation. In (c)-(d) we see the posterior distributions of the leakage location and discharge coefficient.

References

[1] M. Asch, M. Bocquet, M. Nodet, Data Assimilation: Methods, Algorithms, and Applications, SIAM, 2016.
[2] J. Harlim, Data-Driven Computational Methods: Parameter and Operator Estimations, Cambridge University Press, 2018.
[3] A.M. Stuart, Inverse problems: a Bayesian perspective, Acta Numer. 19 (2010) 451–559.
[4] J. Kaipio, E. Somersalo, Statistical and Computational Inverse Problems, vol. 160, Springer Science & Business Media, 2006.
[5] M.G. Kapteyn, J.V. Pretorius, K.E. Willcox, A probabilistic graphical model foundation for enabling predictive digital twins at scale, Nat. Comput. Sci. 1 (5) (2021) 337–347.
[6] S. Brooks, A. Gelman, G. Jones, X.-L. Meng, Handbook of Markov Chain Monte Carlo, CRC Press, 2011.
[7] D. Gamerman, H.F. Lopes, Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference, CRC Press, 2006.
[8] A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations: an Introduction, vol. 92, Springer, 2015.
[9] D. Xiu, Numerical Methods for Stochastic Computations, Princeton University Press, 2010.
[10] H. Wang, J. Li, Adaptive Gaussian process approximation for Bayesian inference with expensive likelihood functions, Neural Comput. 30 (11) (2018) 3072–3094.
[11] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (2020) 477–508.
[12] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, et al., Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence, Tech. Rep., USDOE Office of Science (SC), Washington, DC (United States), 2019.
[13] R. Gribonval, G. Kutyniok, M. Nielsen, F. Voigtlaender, Approximation spaces of deep neural networks, Constr. Approx. (2021) 1–109.
[14] N.T. Mücke, S.M. Bohté, C.W. Oosterlee, Reduced order modeling for parameterized time-dependent pdes using spatially and memory aware deep learning, J. Comput. Sci. (2021) 101408.
[15] J.S. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. 363 (2018) 55–78.
[16] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint, arXiv:2010.08895, 2020.
[17] T. Kadeethum, D. O'Malley, J.N. Fuhg, Y. Choi, J. Lee, H.S. Viswanathan, N. Bouklas, A framework for data-driven solution and parameter estimation of pdes using conditional generative adversarial networks, arXiv preprint, arXiv:2105.13136, 2021.
[18] L. Ruthotto, E. Haber, An introduction to deep generative modeling, GAMM-Mitt. (2021) e202100008.
[19] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, arXiv preprint, arXiv:1406.2661, 2014.
[20] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, arXiv preprint, arXiv:1312.6114, 2013.
[21] P. Dhariwal, A. Nichol, Diffusion models beat gans on image synthesis, arXiv preprint, arXiv:2105.05233, 2021.
[22] D. Rezende, S. Mohamed, Variational inference with normalizing flows, in: International Conference on Machine Learning, PMLR, 2015, pp. 1530–1538.
[23] H. Goh, S. Sheriffdeen, J. Wittmer, T. Bui-Thanh, Solving Bayesian inverse problems via variational autoencoders, arXiv preprint, arXiv:1912.04212, 2019.
[24] J. Whang, E. Lindgren, A. Dimakis, Composing normalizing flows for inverse problems, in: International Conference on Machine Learning, PMLR, 2021, pp. 11158–11169.
[25] D.V. Patel, D. Ray, H. Ramaswamy, A. Oberai, Bayesian inference in physics-driven problems with adversarial priors, in: NeurIPS 2020 Workshop on Deep Learning and Inverse Problems, 2020.
[26] Y. Xia, N. Zabaras, Bayesian multiscale deep generative model for the solution of high-dimensional inverse problems, J. Comput. Phys. 455 (2022) 111008.
[27] J. Adler, O. Öktem, Deep Bayesian inversion, arXiv preprint, arXiv:1811.05910, 2018.
[28] Z. Xiao, K. Kreis, A. Vahdat, Tackling the generative learning trilemma with denoising diffusion gans, arXiv preprint, arXiv:2112.07804, 2021.
[29] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, S. Ganguli, Deep unsupervised learning using nonequilibrium thermodynamics, in: International Conference on Machine Learning, PMLR, 2015, pp. 2256–2265.
[30] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein generative adversarial networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 214–223.
[31] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092.
[32] W.K. Hastings, Monte Carlo sampling methods using Markov chains and their applications, 1970.
[33] M.D. Hoffman, A. Gelman, The no-u-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo, J. Mach. Learn. Res. 15 (1) (2014) 1593–1623.
[34] A. Stuart, A. Teckentrup, Posterior consistency for Gaussian process approximations of Bayesian posterior distributions, Math. Comput. 87 (310) (2018) 721–753.
[35] A. Jabbar, X. Li, B. Omar, A survey on generative adversarial networks: variants, applications, and training, arXiv preprint, arXiv:2006.05132, 2020.
[36] P. Brémaud, Probability Theory and Stochastic Processes, Springer Nature, 2020.
[37] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville, Improved training of Wasserstein gans, arXiv preprint, arXiv:1704.00028, 2017.
[38] S. Liu, O. Bousquet, K. Chaudhuri, Approximation and convergence properties of generative adversarial learning, arXiv preprint, arXiv:1705.08991, 2017.
[39] B. Sanderse, V.V. Dighe, K. Boorsma, G. Schepers, Efficient Bayesian calibration of aerodynamic wind turbine models using surrogate modeling, Wind Energy Sci. Discuss. (2021) 1–34.
[40] F. Lu, M. Morzfeld, X. Tu, A.J. Chorin, Limitations of polynomial chaos expansions in the Bayesian solution of inverse problems, J. Comput. Phys. 282 (2015) 138–147.
[41] V.I. Bogachev, Measure Theory, vol. 1, Springer Science & Business Media, 2007.
[42] B. Sprungk, On the local Lipschitz stability of Bayesian inverse problems, Inverse Probl. 36 (5) (2020) 055015.
[43] V.M. Panaretos, Y. Zemel, An Invitation to Statistics in Wasserstein Space, Springer Nature, 2020.
[44] X. Han, H. Gao, T. Pffaf, J.-X. Wang, L.-P. Liu, Predicting physics in mesh-reduced space with temporal attention, arXiv preprint, arXiv:2201.09113, 2022.
[45] D.L. Colton, R. Kress, R. Kress, Inverse Acoustic and Electromagnetic Scattering Theory, vol. 93, Springer, 1998.
[46] S. Domesová, M. Beres, Solution of inverse problems using Bayesian approach with application to estimation of material parameters in Darcy flow, Adv. Electr. Electron. Eng. 15 (2) (2017).
[47] S. Ruchi, S. Dubinkina, M. Iglesias, Transform-based particle filtering for elliptic Bayesian inverse problems, Inverse Probl. 35 (11) (2019) 115005.
[48] P. Kumar, P. Luo, F.J. Gaspar, C.W. Oosterlee, A multigrid multilevel Monte Carlo method for transport in the Darcy–Stokes system, J. Comput. Phys. 371 (2018) 382–408.
[49] B. Cockburn, J. Guzmán, H. Wang, Superconvergent discontinuous Galerkin methods for second-order elliptic problems, Math. Comput. 78 (265) (2009) 1–24.
[50] A. Logg, K.-A. Mardal, G. Wells, Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book, vol. 84, Springer Science & Business Media, 2012.
[51] Z. Ding, Q. Li, Ensemble Kalman inversion: mean-field limit and convergence analysis, Stat. Comput. 31 (1) (2021) 1–21.
[52] P.K. Kundu, I.M. Cohen, Fluid Mechanics, 2002.

[53] E. Hauge, O.M. Aamo, J.-M. Godhavn, Model based pipeline monitoring with leak detection, IFAC Proc. Vol. 40 (12) (2007) 318–323.
[54] J.A. Schetz, A.E. Fuhs, Handbook of Fluid Dynamics and Fluid Machinery, vol. 1, Wiley, New York, 1996.
[55] J.S. Hesthaven, T. Warburton, Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications, Springer Science & Business Media, 2007.
[56] R.J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems, SIAM, 2007.
[57] A. Hørsholt, L.H. Christiansen, K. Meyer, J.K. Huusom, J.B. Jørgensen, Spatial discretization and Kalman filtering for ideal packed-bed chromatography, in: 2019 18th European Control Conference (ECC), IEEE, 2019, pp. 2356–2361.
[58] J. Feinberg, H.P. Langtangen, Chaospy: an open source tool for designing methods of uncertainty quantification, J. Comput. Sci. 11 (2015) 46–57.