Deep-learing-based reduced-order modeling for subsurface flow simulation

报告人: 常思洋

June 12, 2020

Outline

Outline

Abstract

A new deep-learning-based reduced-order modeling (ROM) framework is proposed for application in subsurface flow simulation. The reduced-order model is based on an existing embed-to-control (E2C) framework and includes an auto-encoder, which projects the system to a low-dimensional subspace, and a linear transition model, which approximates the evolution of the system states in low dimension. We introduce a physics-based loss function that penalizes predictions that are inconsistent with the governing flow equations. The loss function is also modified to emphasize accuracy in key well quantities of interest. The E2C ROM is shown to be very analogous to an existing ROM, POD-TPWL, which has been extensively developed for subsurface flow simulation. The new ROM is applied to oil-water flow in a heterogeneous reservoir, with flow driven by nine wells operating under time-varying control specifications.

Outline

2.1 Governing equations

The governing equations for immiscible oil-water flow derive from mass conservation for each component combined with Darcy's law for each phase. The resulting equations, with capillary pressure effects neglected, are

$$\frac{\partial}{\partial t}\left(\phi S_j \rho_j\right) - \nabla \cdot \left(\lambda_j \rho_j \mathbf{k} \nabla p\right) + \sum_w \rho_j q_j^w = 0$$

This oil-water model is completed by enforcing the saturation constraint $S_o + S_w = 1$. Because the system considered in this work is horizontal (in the $x - y$ plane), gravity effects are neglected.

2.1 Governing equations

The oil and water flow equations are discretized using a standard finite-volume formulation, and their solutions are computed for each grid block. In this work, we use Stanford's Automatic Differentiation-based General Purpose Research Simulator, AD-GPRS, for all flow simulations. Let n_b denote the number of grid blocks in the model. The flow system is fully defined through the use of two primary variables, p and S_w, in each grid block, so the total number of variables in the system is $2n_b$. We define $\mathbf{x}_t = \left[\mathbf{p}_t^{\mathrm{T}}, \mathbf{S}_t^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^{2n_b}$ to be the state vector for the flow variables at a specific time step t, where $\mathbf{p}_t \in \mathbb{R}^{2n_b}$ and $\mathbf{S}_t \in \mathbb{R}^{2n_b}$ denote the pressure and saturation in every grid block at time step t.

2.1 Governing equations

The set of nonlinear algebraic equations representing the discretized fully-implicit system can be expressed as:

$$\mathbf{g}\left(\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{u}_{t+1}\right) = \mathbf{0}$$

where $\mathbf{g} \in \mathbb{R}^{2n_b}$ is the residual vector (set of nonlinear algebraic equations) we seek to drive to zero.and $\mathbf{u}_{t+1} \in \mathbb{R}^{n_w}$ designates the well control variables, which can be any combination of bottom-hole pressures (BHPs) or well rates. Here n_w denotes the number of wells in the system.

2.1 Governing equations

Newton's method is typically used to solve the full-order discretized nonlinear system. This requires constructing the sparse Jacobian matrix of dimension $2n_b \times 2n_b$, and then solving a linear system of dimension $2n_b$, at each iteration for every time step. Solution of the linear system is often the most time-consuming part of the simulation.

As will be explained later, both POD-TPWL and the deep-learning-based E2C ROM avoid the test-time construction and solution of this high-dimensional system.

2.2 POD-TPWL formulation

POD-TPWL and other POD-based ROMs involve an offline (train-time) stage and an online (test-time) stage. During the offline stage, a number of training simulation runs are performed using a full-order simulator (AD-GPRS in this work). The goal here is to predict test-time results with varying well control sequences.

During training runs, we apply different well control sequences $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_{N_{\text{ctrl}}}] \in \mathbb{R}^{n_w \times N_{\text{ctrl}}}$,where $\mathbf{u}_k \in \mathbb{R}^{n_w}$, $k = 1, \ldots, N_{\text{ctrl}}$, contains the settings for all wells at control step k,and N_{ctrl} denotes the total number of control steps in a training run. In our examples we have 20 control steps and around 100 time steps. State variables in all grid blocks (referred to as snapshots) and derivative matrices are saved at each time step in the training runs. Information saved from the training runs is used to (very efficiently) approximate test solutions.

Outline

E2C overview

The embed-to-control framework entails three processing steps: an encoder or inference model that projects the system variables from a high-dimensional space to a low-dimensional subspace (referred to here as the latent space), a linear transition model that approximates system dynamics in low-dimension, and a decoder or generative model that projects solutions back to high-dimensional (full-order) space.

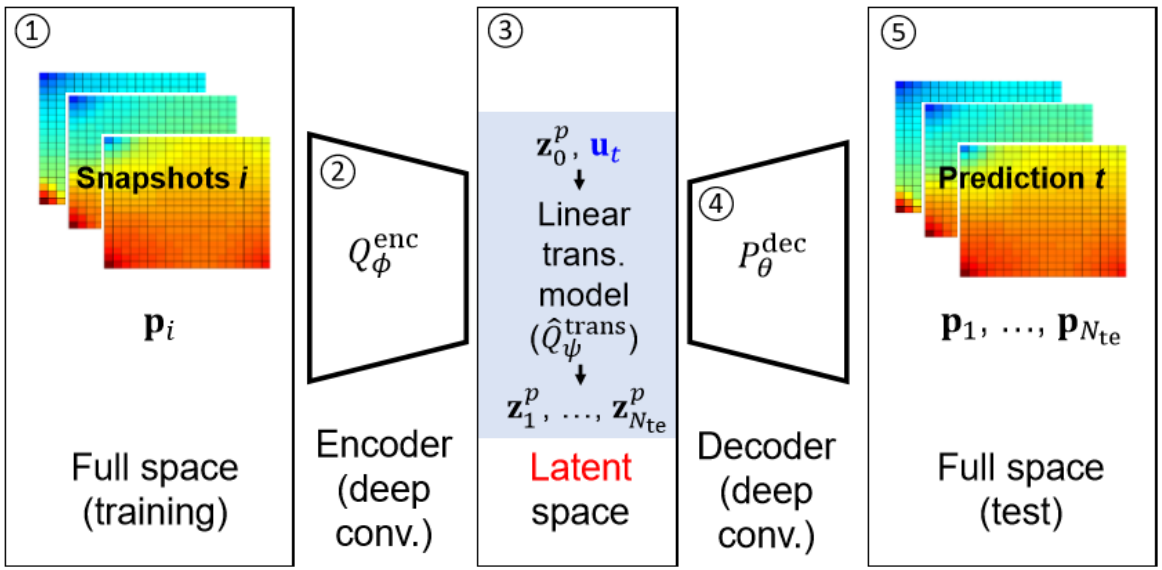


Figure 1: Embed-to control(E2C) overview

3.1 Encoder component

During training, sequences of pressure and saturation snapshots are fed through the encoder network, and sequences of latent state variables $\mathbf{z}_t \in \mathbb{R}^{l_z}$ are generated.

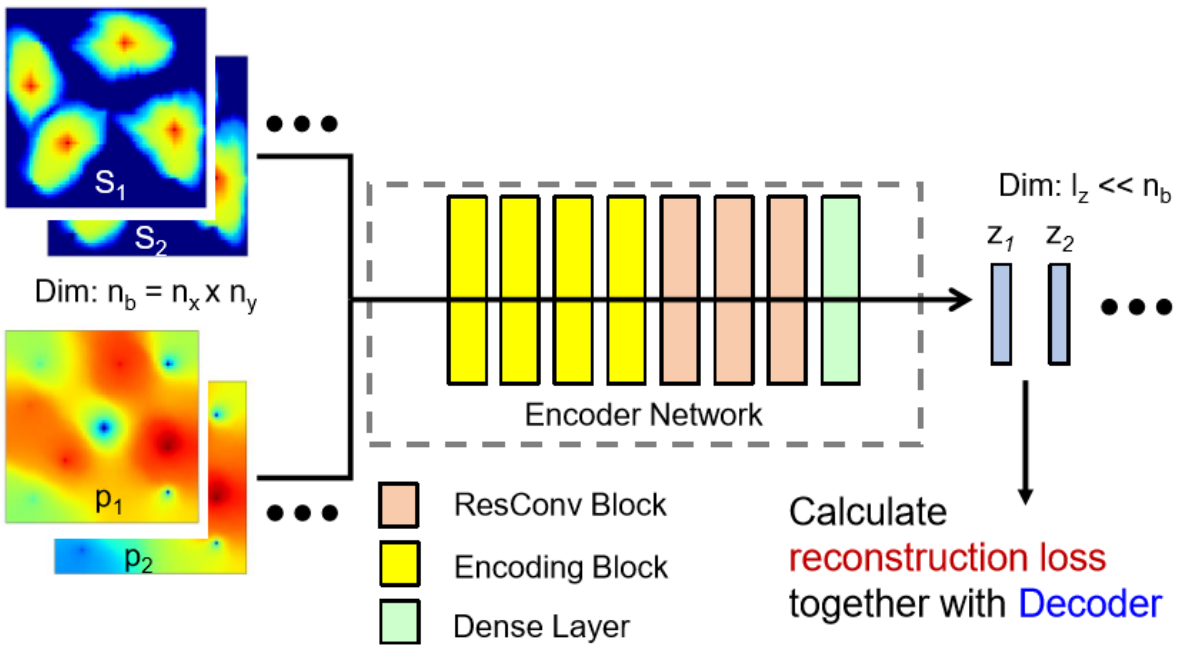
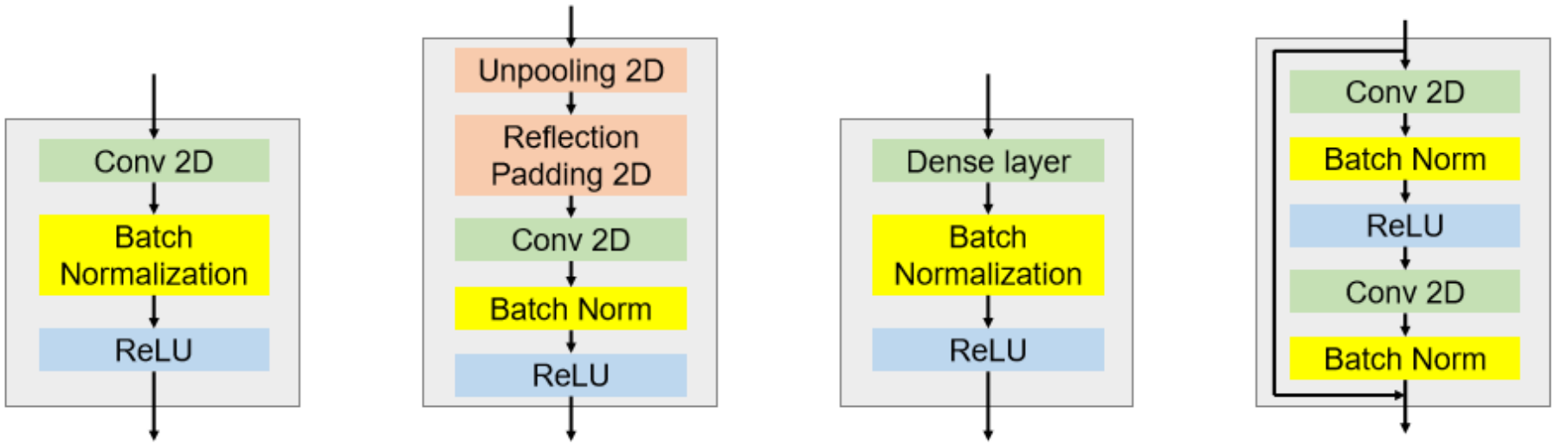


Figure 2: Encoder layout

Appendix A



(a) Encoding block (b) Decoding block (c) Trans. block (d) Res conv block

Figure 3: Encoder layout

Appendix A

Layer	Filter number, size and stride	Output size
Input		$(N_x, N_y, 2)$
Encoding block	16 of $3 \times 3 \times 2$, stride 2	$(N_x/2, N_y/2, 16)$
Encoding block	32 of $3 \times 3 \times 16$, stride 1	$(N_x/2, N_y/2, 64)$
Encoding block	64 of $3 \times 3 \times 32$, stride 2	$(N_x/4, N_y/4, 128)$
Encoding block	128 of $3 \times 3 \times 64$, stride 1	$(N_x/4, N_y/4, 128)$
ResConv block	128 of $3 \times 3 \times 128$, stride 1	$(N_x/4, N_y/4, 128)$
ResConv block	128 of $3 \times 3 \times 128$, stride 1	$(N_x/4, N_y/4, 128)$
ResConv block	128 of $3 \times 3 \times 128$, stride 1	$(N_x/4, N_y/4, 128)$
Dense		$(l_z, 1)$

Figure 4: Network architecture for encoder

3.2 Encoder component

The input to an encoding block is first fed through a convolution operation, which can also be viewed as a linear filter. The mathematical formulation of linear filtering is $F_{i,j}(\mathbf{x}) = \sum_{p=-n}^{n} \sum_{q=-n}^{n} \mathbf{w}_{p,q} \mathbf{x}_{i+p,j+q} + b$. A batchNorm operation is a crucial step, since it renders the learning process less sensitive to parameter initialization, which means a larger initial learning rate can be used. ReLU is referred to as the 'activation' of the encoding block. The conv2D-batchNorm-ReLU architecture (with variation in ordering) is a standard processing step in CNNs.Following the idea of resNet, we add a stack of resConv blocks to the encoder network to deepen the network while mitigating the vanishing-gradient issue. A dense (fully-connected) layer is simply a linear projection that maps a high-dimensional vector to a low-dimensional vector.

3.3 Linear transition model

The inputs to the linear transition model include the latent variable for the current state $\mathbf{z}_t \in \mathbb{R}^{l_z}$, the current step control $\mathbf{u}_{t+1} \in \mathbb{R}^{n_w}$, and time step size Δt. The model outputs the predicted latent state for the next time step $\hat{\mathbf{z}}_{t+1}$.

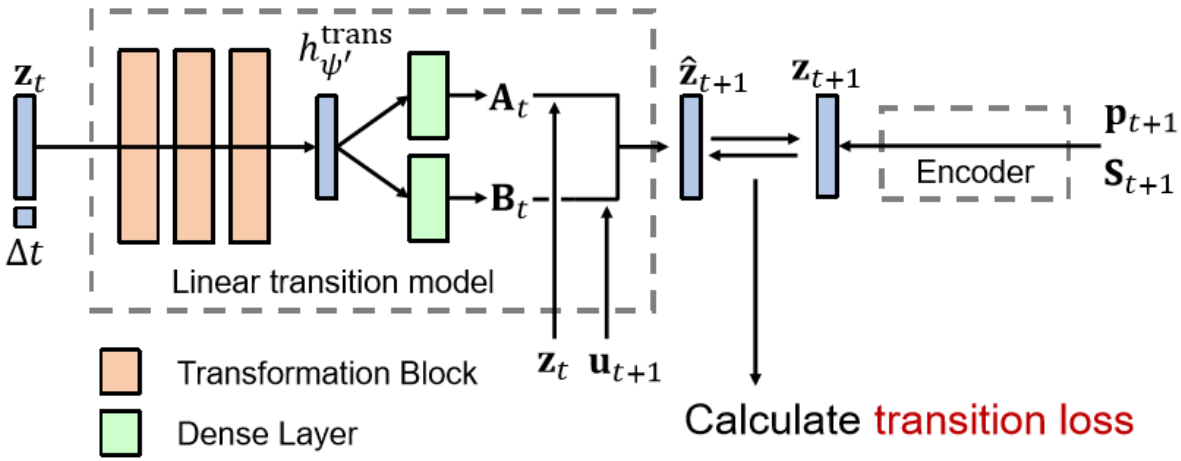


Figure 5: Liner transition model layout

3.4 Decoder component

Latent variables predicted by the linear transition model (at time step t + 1) are fed to the decoder network as input, 15 and the predicted high-dimensional states are output. The architecture of the decoder is analogous to that of the encoder except it is in reversed order (which is not surprising since the decoder is conducting the inverse operation).

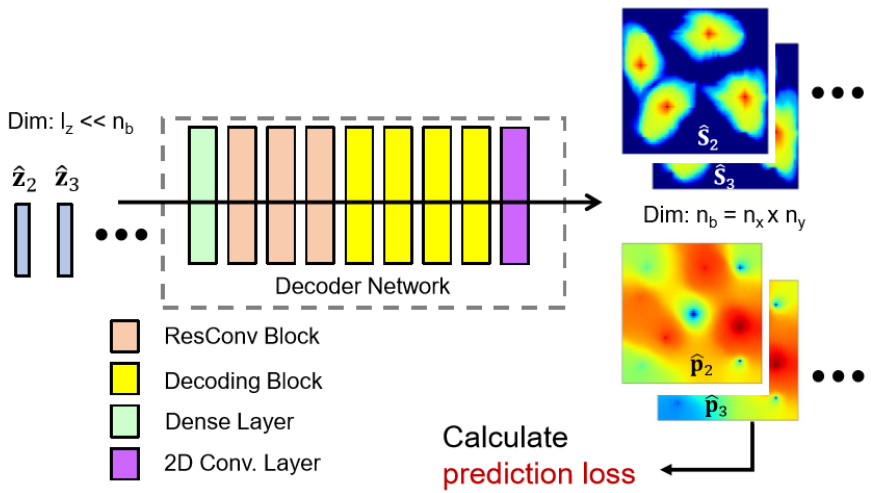


Figure 6: Decoder layout

3.5 Loss function with physical constraints

We have briefly introduced the reconstruction loss L_R, the linear transition loss L_T, and the prediction loss L_{PD}, which comprise major components of the total loss function.

The reconstruction loss for a training data point i can be expressed as :

$$(\mathcal{L}_{\mathrm{R}})_i = \left\{ \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2^2 \right\}_i \tag{3.1}$$

The variable x_t is the state variable at time step t from a training simulation, and$\hat{\mathbf{x}}_t = P_\theta^{\mathrm{dec}}(\mathbf{z}_t) = P_\theta^{\mathrm{dec}}\left(Q_\phi^{\mathrm{enc}}(\mathrm{x}_t)\right)$ denotes the states reconstructed by the encoder and decoder.

The linear transition loss for training point i is similarly defined as

$$(\mathcal{L}_{\mathrm{T}})_i = \left\{ \|\mathbf{z}_{t+1} - \hat{\mathbf{z}}_{t+1}\|_2^2 \right\}_i \tag{3.2}$$

where $\mathbf{z}_{t+1} = Q_\phi^{\mathrm{enc}}(\mathbf{x}_{t+1})$ is the latent variable encoded from the full-order state variable at $t+1$, and the variable $\hat{\mathbf{z}}_{t+1} = Q_\psi^{\mathrm{trans}}(\mathbf{z}_t, \mathbf{u}_{t+1}, \Delta t)$ denotes the latent variable predicted by the linear transition model.

3.5 Loss function with physical constraints

Finally, the prediction loss for training point i is defined as

$$(\mathcal{L}_{\mathrm{PD}})_i = \left\{ \|\mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1}\|_2^2 \right\}_i \tag{3.3}$$

where $\mathbf{x}_{t+1}$ designates the state variable at time step t + 1 from the training simulations, and $\hat{\mathbf{x}}_{t+1} = P_\theta^{\mathrm{dec}}(\hat{\mathbf{z}}_{t+1})$ represents the full-order state variable predicted by the ROM.

The data mismatch loss is the sum of these losses averaged over all training data points,

$$\mathcal{L}_d = \frac{1}{N_t} \sum_{i=1}^{N_t} (\mathcal{L}_{\mathrm{R}})_i + (\mathcal{L}_{\mathrm{PD}})_i + \lambda (\mathcal{L}_{\mathrm{T}})_i \tag{3.4}$$

We define the physics-based loss for each data point,as:

$$\begin{aligned}(\mathcal{L}_p)_i = &\left\{ \left\| \mathbf{k} \cdot \left[(\nabla \mathbf{p}_t - \nabla \hat{\mathbf{p}}_t)_{\mathrm{recon}} + (\nabla \mathbf{p}_{t+1} - \nabla \hat{\mathbf{p}}_{t+1})_{\mathrm{pred}} \right] \right\|_2^2 \right\}_i \\ &+ \gamma \left\{ \left\| (\mathbf{q}_t^w - \hat{\mathbf{q}}_t^w)_{\mathrm{recon}} + (\mathbf{q}_{t+1}^w - \hat{\mathbf{q}}_{t+1}^w)_{\mathrm{pred}} \right\|_2^2 \right\}\end{aligned} \tag{3.5}$$

3.5 Loss function with physical constraints

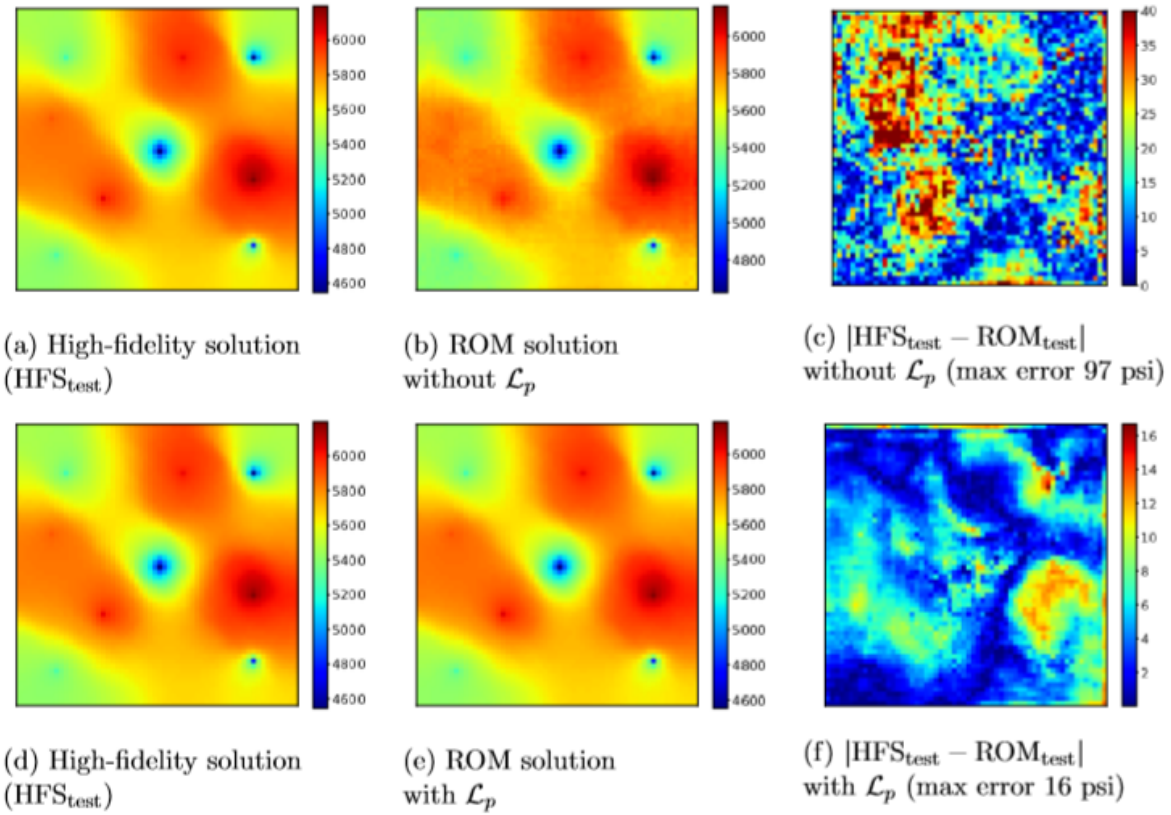
The physics-based loss function is computed by averaging $(\mathcal{L}_p)_i$ over all data points, i.e

$$\mathcal{L}_p = \frac{1}{N_t} \sum_{i=1}^{N_t} (\mathcal{L}_p)_i \tag{3.6}$$

Combining the loss for data mismatch with this physics-based loss, the total loss function becomes

$$\mathcal{L} = \mathcal{L}_d + \alpha \mathcal{L}_p \tag{3.7}$$

3.5 Loss function with physical constraints



(a) High-fidelity solution (HFS_{test})

(b) ROM solution without $\mathcal{L}_p$

(c) $|\text{HFS}_{\text{test}} - \text{ROM}_{\text{test}}|$ without $\mathcal{L}_p$ (max error 97 psi)

(d) High-fidelity solution (HFS_{test})

(e) ROM solution with $\mathcal{L}_p$

(f) $|\text{HFS}_{\text{test}} - \text{ROM}_{\text{test}}|$ with $\mathcal{L}_p$ (max error 16 psi)

Figure 7: Pressure field predictions with and without $\mathcal{L}_p$

Outline

model setup

The geological model, in terms of the log-permeability field, is shown in Fig. 8. The locations of the four injection wells and five production wells are also displayed.

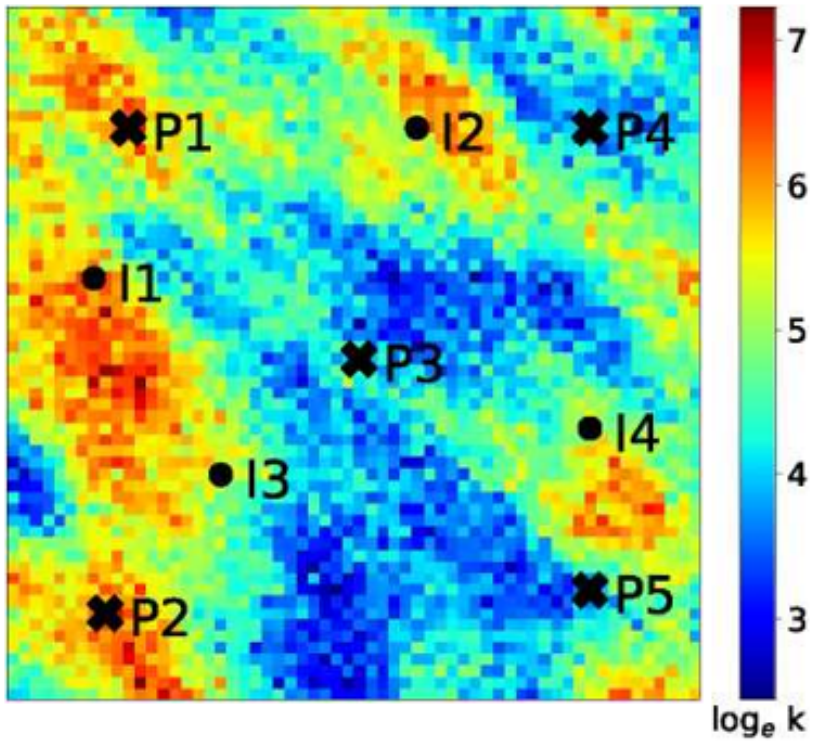


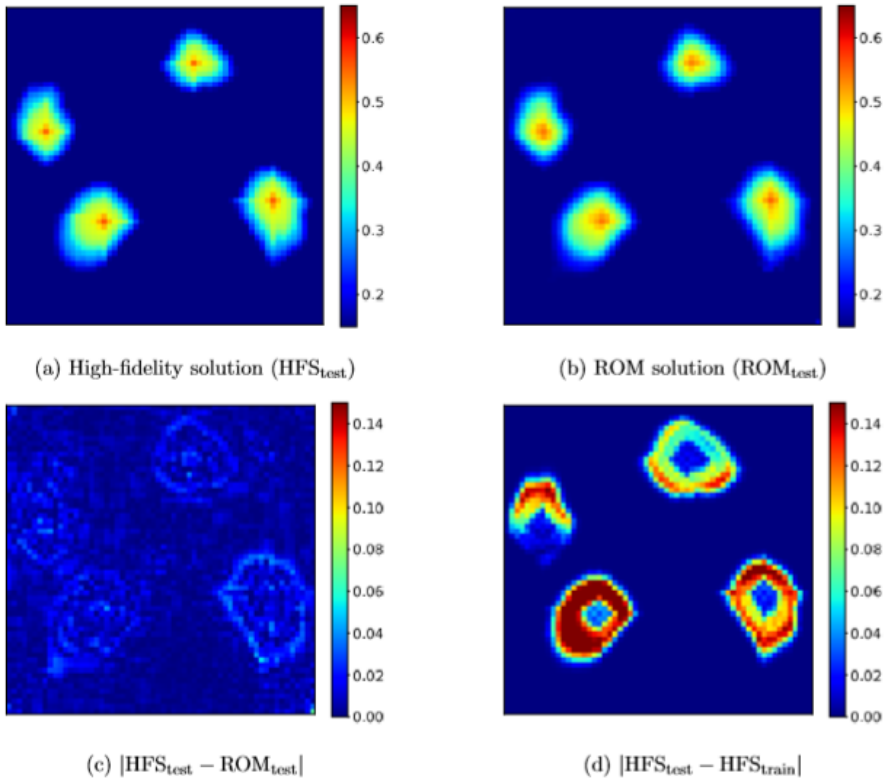
Figure 8: Log-permeability field and well locations

model setup

In the examples in this paper, we simulate a total of 300 training runs.Rather than train over all snapshots, here we set $N_{\mathrm{ctrl}} = N_{\mathrm{tr}} = N_{\mathrm{te}} = 20$. This accelerates training and focuses ROM predictions on quantities of interest at time steps when the controls are changing. This results in a total number of data points of $N_{\mathrm{t}} = 300 \times 20 = 6000$.

The gradient of the total loss function with respect to the model parameters (ϕ, ψ, θ) is calculated via back-propagation through the embed-to-control framework. The adaptive moment estimation (ADAM) algorithm is used for this optimization, as it has been proven to be effective for optimizing deep neural networks.

results



(a) High-fidelity solution ($\mathrm{HFS}_{\mathrm{test}}$)

(b) ROM solution ($\mathrm{ROM}_{\mathrm{test}}$)

(c) $|\mathrm{HFS}_{\mathrm{test}} - \mathrm{ROM}_{\mathrm{test}}|$

(d) $|\mathrm{HFS}_{\mathrm{test}} - \mathrm{HFS}_{\mathrm{train}}|$

Figure 9: Test case 1:saturation field at 200 days