Estimating injection rate from ground displacement in coupled flow and geomechanics problems using deep learning enabled Bayesian inference

Saumik Dana[a]

[a] *University of Southern California,*

Abstract

In this work, we use a combination of Bayesian inference, Markov chain Monte Carlo and deep learning in the form of LSTM autoencoders to build and test a framework to provide robust estimates of injection rate from ground surface data in coupled flow and geomechanics problems. We use LSTM autoencoders to reconstruct the displacement time series for grid points on the top surface of a faulting due to water injection problem. We then deploy this LSTM autoencoder based model instead of the high fidelity model in the Bayesian inference framework to estimate injection rate from displacement input.

Keywords: LSTM autoencoder, Bayesian inference, Markov chain Monte Carlo, coupled flow and geomechanics

1. Introduction

The high fidelity forward model for coupled flow and geomechanics [1–10] in the absence of inertia is a piece of the puzzle in which forward simulations are used to arrive at the seismic impact of fault slip on earthquake activity. Typically, the earthquakes are measured with seismograms/geophones on the surface as P-waves and S-waves, and then these readings are used to calibrate the seismic activity for constant monitoring. The acceleration/displacement field around the fault slip activity translates to these waves recorded on the surface, and forward simulations with wave propagation bridge that gap. That being said, a seismic recording on the surface cannot easily be backtraced to the source of the fault slip. That is precisely inverse modeling, and the Bayesian framework coupled with Markov chain Monte Carlo [11] allows us to put such a framework in place. The accelerations/displacements are field quantities, and the inverse estimation of the field around the fault from the time series at the seismogram/geophone is not a trivial task. With that in mind, we test the robustness of the Bayesian/MCMC framework to inversely estimate a value instead of a field. To run Bayesian though, we need multiple (sometimes millions of) high fidelity simulation runs, which will become infeasible if the model is sophisticated. Hence, reduced order models are needed, and we deploy LSTM autoencoders for that purpose. In this work, we go from the forward model to the simulations to the time series data for the ground surface data, then LSTM autoencoder based reconstruction for an archetypal faulting due to injection problem, and then eventually Bayesian inference estimation of injection rate using the constructed reduced order model. The big picture scenario is that the recording at the

seismogram/geophone would be fed into the Bayesian/MCMC framework as an input, and the framework would provide an estimate of whichever model parameters are critical. This document is structured as follows: In Section 2, we give details of the forward model and the specific problem under consideration, in Section 3, we give details of the LSTM autoencoder based time series reconstruction, Section 4 provides a formalism of the Bayesian inference framework with Markov chain Monte Carlo, in Section 5 we present the estimate results and in Section 6, we provide conclusions and outlook. The procedural framework is elucidated in the following steps

- Run high fidelity simulations for a bunch of injection rates
- For each injection rate, construct a displacement time series at a chosen grid point on the ground surface
- Add noise to the time series to eventually serve as noisy data for the Bayesian inference framework
- Train the LSTM autoencoder with time stamp and injection rate as input and displacement time series (without the noise) as the target
- Run the Bayesian inference framework with the LSTM autoencoder based reduced order model
- Test the robustness of the framework by comparing the estimates of the injection rate with the ground truth injection rate

2. The high fidelity forward model

The governing PDE for displacement $\mathbf{u}$ is the linear momentum balance given by

$$\nabla \cdot \boldsymbol{\sigma} + \rho_b \mathbf{g} = \mathbf{0} \tag{1}$$

with the constitutive laws relating poroelastic stress tensor $\boldsymbol{\sigma}$, effective stress tensor $\boldsymbol{\sigma'}$, strain tensor $\boldsymbol{\epsilon}$, volumetric strain $\epsilon = tr(\boldsymbol{\epsilon})$ and pore pressure p given by

$$\begin{aligned} \boldsymbol{\sigma} &= \boldsymbol{\sigma'} - bp\mathbf{I} \\ \boldsymbol{\sigma'} &= \lambda\epsilon\mathbf{I} + 2G\boldsymbol{\epsilon} = \boldsymbol{D\epsilon} \end{aligned} \tag{2}$$

with boundary and initial conditions given by

$$\begin{aligned} \mathbf{u} = \overline{\mathbf{u}} \text{ on } \Gamma_D,\ \dot{\mathbf{u}} = \dot{\overline{\mathbf{u}}} \text{ on } \Gamma_D,\ \boldsymbol{\sigma}^T\mathbf{n} = \overline{\mathbf{t}} \text{ on } \Gamma_N \\ \mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x}),\ \dot{\mathbf{u}}(\mathbf{x}, 0) = \dot{\mathbf{u}}_0(\mathbf{x}) \end{aligned}$$

As shown in Fig. 1, slip on the fault is the displacement of the positive side relative to the negative side:

$$(\boldsymbol{u}_+ - \boldsymbol{u}_-) - \boldsymbol{d} = \mathbf{0} \text{ on } \Gamma_f, \tag{3}$$

Recognizing that fault tractions are analogous to the boundary tractions, we add in the contributions from integrating the Lagrange multipliers $\boldsymbol{l} \equiv \boldsymbol{\sigma'}\boldsymbol{n}$ over the fault surface in the conventional finite element formulation to get

$$\begin{aligned} &\int_\Omega \nabla\boldsymbol{\eta} : (\boldsymbol{\sigma'} - bp\mathbf{I})\, d\Omega - \int_\Omega \boldsymbol{\eta} \cdot \rho_b\boldsymbol{g}\, d\Omega - \int_{\Gamma_N} \boldsymbol{\eta} \cdot \overline{\boldsymbol{t}}\, d\Gamma \\ &+ \int_{\Gamma_{f_+}} \boldsymbol{\eta} \cdot (\boldsymbol{l} - bp_+\boldsymbol{n})\, d\Gamma - \int_{\Gamma_{f_-}} \boldsymbol{\eta} \cdot (\boldsymbol{l} - bp_-\boldsymbol{n})\, d\Gamma = 0 \end{aligned} \tag{4}$$

We use the Mohr-Coulomb theory [12] to define the stability criterion for the fault and

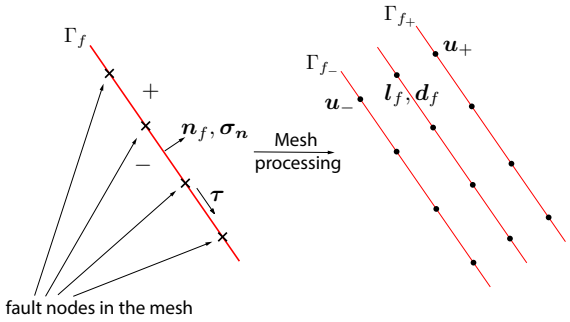

Figure 1: A fault surface Γ_f is processed to create three surfaces: the positive side surface Γ_{f+} containing $\boldsymbol{u}_+$, the negative side surface Γ_{f-} containing $\boldsymbol{u}_-$, and the slip surface containing the fault effective traction vector $\boldsymbol{l}$ and the slip vector $\boldsymbol{d}$

define a fault pressure $p_f = \frac{p_- + p_+}{2}$. The shear stress and frictional stresses on the fault are

$$\tau = |\boldsymbol{l} - \sigma_n' \boldsymbol{n}| \equiv |\boldsymbol{l} - (\boldsymbol{l} \cdot \boldsymbol{n})\boldsymbol{n}|$$

$$\tau_f = \begin{cases} \tau_c - \mu_f \boldsymbol{l} \cdot \mathbf{n}, & \boldsymbol{l} \cdot \mathbf{n} < 0, \\ \tau_c, & \boldsymbol{l} \cdot \mathbf{n} \geq 0 \end{cases}$$

where τ_c is the cohesive strength of the fault, μ_f is the coefficient of friction which evolves as

$$\mu_f = \begin{cases} \mu_s - (\mu_s - \mu_d)\frac{|\boldsymbol{d}|}{d_c}, & |\boldsymbol{d}| \leq d_c, \\ \mu_d, & |\boldsymbol{d}| > d_c \end{cases} \tag{5}$$

where d_c is a critical slip distance. The fields are approximated as follows:

$$\boldsymbol{u} \approx \boldsymbol{u}_h = \sum_{b=1}^{n_{\text{node}}} \eta_b \boldsymbol{U}_b, \quad \boldsymbol{l} \approx \boldsymbol{l}_h = \sum_{b=1}^{n_{f,\text{node}}} \eta_b \boldsymbol{L}_b, \quad \boldsymbol{d} \approx \boldsymbol{d}_h = \sum_{b=1}^{n_{f,\text{node}}} \eta_b \boldsymbol{D}_b,$$

where n_{node} is the total number of nodes and $n_{f,\text{node}}$ is the number of Lagrange nodes. After substitution of the finite element approximations into the weak form of the problem, we obtain the fully discrete aligns in residual form for all nodes a and lagrange nodes $\bar{a}$:

$$\begin{aligned} \boldsymbol{R}_{u,a}^{Stat} = &\int_\Omega \boldsymbol{B}_a^T (\boldsymbol{\sigma}_h'^{n+1} - b p_h^{n+1} \mathbf{1}) d\Omega - \int_\Omega \boldsymbol{\eta}_a^T \rho_{b,h}^{n+1} \boldsymbol{g} d\Omega - \int_{\Gamma_N} \boldsymbol{\eta}_a^T \bar{\boldsymbol{t}} d\Gamma \\ &+ \int_{\Gamma_{f_+}} \boldsymbol{\eta}_a^T (\boldsymbol{l}_h^{n+1} - b p_{f,h}^{n+1} \boldsymbol{n}) d\Gamma - \int_{\Gamma_{f_-}} \boldsymbol{\eta}_a^T (\boldsymbol{l}_h^{n+1} - b p_{f,h}^{n+1} \boldsymbol{n}) d\Gamma = \mathbf{0} \end{aligned} \tag{6}$$

$$\boldsymbol{R}_{l,\bar{a}}^{Stat} = \int_{\Gamma_{f_+}} \boldsymbol{\eta}_{\bar{a}}^T \boldsymbol{u}_{h_+}^{n+1} d\Gamma - \int_{\Gamma_{f_-}} \boldsymbol{\eta}_{\bar{a}}^T \boldsymbol{u}_{h_-}^{n+1} \, d\Gamma - \int_{\Gamma_f} \boldsymbol{\eta}_{\bar{a}}^T \boldsymbol{d}_h^{n+1} d\Gamma = \mathbf{0} \tag{7}$$

We find the system Jacobian matrix by isolating the term for the increments in displacements and Lagrange multipliers at time step $n+1$. The system of linear aligns is:

$$\left[\begin{array}{ccc:c} \boldsymbol{K}_{rr} & \boldsymbol{K}_{r+} & \boldsymbol{K}_{r-} & \mathbf{0} \\ \boldsymbol{K}_{+r} & \boldsymbol{K}_{++} & \mathbf{0} & \boldsymbol{C}_+^T \\ \boldsymbol{K}_{-r} & \mathbf{0} & \boldsymbol{K}_{--} & -\boldsymbol{C}_-^T \\ \hdashline \mathbf{0} & \boldsymbol{C}_+ & -\boldsymbol{C}_- & \mathbf{0} \end{array}\right] \left[\begin{array}{c} d\boldsymbol{U}_r \\ d\boldsymbol{U}_+ \\ d\boldsymbol{U}_- \\ \hline d\boldsymbol{L} \end{array}\right] = - \left[\begin{array}{c} \boldsymbol{R}_{u,r}^{Stat} \\ \boldsymbol{R}_{u,+}^{Stat} \\ \boldsymbol{R}_{u,-}^{Stat} \\ \hline \boldsymbol{R}_l^{Stat} \end{array}\right] \tag{8}$$

Algorithm 1: Time-marching in poroelastostatics

$n \leftarrow 0$; $t \leftarrow 0$;
$\boldsymbol{d}_h^0 \leftarrow \boldsymbol{0}$; // Initialize fault slip at virgin state
$\boldsymbol{u}_h^0 \leftarrow \boldsymbol{u}_h^{Prestep}$; // Initial condition based on a elastic prestep solve
while $t < T$ **do**
  /* time marching till final time */
  while *Not converged* **do**
    /* Staggered solution algorithm loop */
    Solve flow problem for pressures;
    $\boldsymbol{d}_h^{n+1} \leftarrow \boldsymbol{d}_h^n$; // Initialize fault slip for next time step
    Solve system of aligns using GMRES; // Krylov subspace solver
    $\boldsymbol{L} \leftarrow \boldsymbol{L} + d\boldsymbol{L}$; // Update lagrange multipliers
    $\boldsymbol{U} \leftarrow \boldsymbol{U} + d\boldsymbol{U}$; // Update displacements
    for *Loop over lagrange nodes* **do**
      $\tau \leftarrow |\boldsymbol{l}_h^{n+1} - (\boldsymbol{l}_h^{n+1} \cdot \boldsymbol{n})\boldsymbol{n}|$; // Obtain shear stress on fault
      $\tau_f \leftarrow \tau_f|_{\boldsymbol{d}_h^n}$; // Compute fault friction based on the slip value at previous time step
      while $\tau > \tau_f$ **do**
        /* Satisfy fault constitutive law */
        $\boldsymbol{U}_+ \leftarrow \boldsymbol{U}_+ - \boldsymbol{K}_{++}^{-1} \frac{\tau - \tau_f}{\tau} \boldsymbol{L}$;
        $\boldsymbol{U}_- \leftarrow \boldsymbol{U}_- + \boldsymbol{K}_{--}^{-1} \frac{\tau - \tau_f}{\tau} \boldsymbol{L}$;
        $\boldsymbol{d}_h^{n+1} \leftarrow \boldsymbol{u}_{h_+}^{n+1} - \boldsymbol{u}_{h_-}^{n+1}$; // Update fault slip
        $\tau_f \leftarrow \tau_f|_{\boldsymbol{d}_h^{n+1}}$; // Update fault friction
      $\boldsymbol{d}_h^{n+1} \leftarrow \boldsymbol{u}_{h_+}^{n+1} - \boldsymbol{u}_{h_-}^{n+1}$; // Update fault slip
  $n \leftarrow n + 1$;
  $t \leftarrow t + \Delta t$;

where the top row corresponds to displacement nodes excluding the fault positive and negative side nodes. In many quasi-static simulations it is convenient to compute a static problem with elastic deformation prior to computing a transient response. The heavy lifting for the time marching is done at the Krylov solver [13] stage to solve the system of Eqs. (8) as shown in Algorithm 1.

2.1. Water injection in the presence of fault

As shown in Fig. 2, we consider a two-dimensional plane-strain model with the fault under normal faulting conditions, that is, the vertical principal stress due to gravity is the largest among the three principal stresses. The aquifer is hydraulically compartmentalized with a sealing fault that cuts across it. The storage capacity of the aquifer is limited by overpressurization and slip on the fault. The initial fluid pressure at $500\,m$ depth is $5\,MPa$ and $24.63\,MPa$ at $2500\,m$, considering a hydrostatic gradient of $9.81\,MPa/km$ and an atmospheric pressure of $0.1\,MPa$ at the ground surface. The rock density is $2260\,kg/m^3$, so the lithostatic gradient is $22.17\,MPa/km$. Assuming a porosity of 0.1, the initial vertical

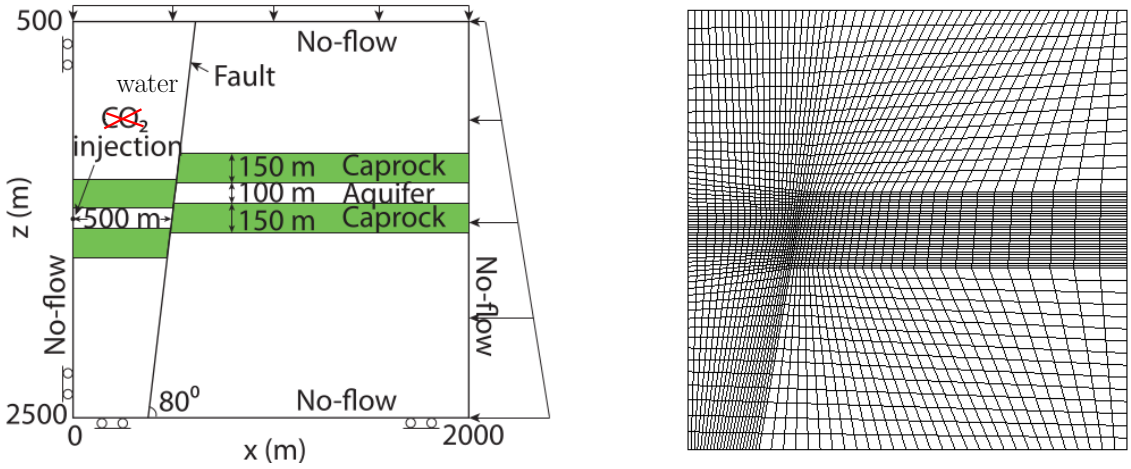

Figure 2: Model of the water injection plane strain case [14]) and the mesh of 3127 nodes and 3016 elements with more refinement around the fault

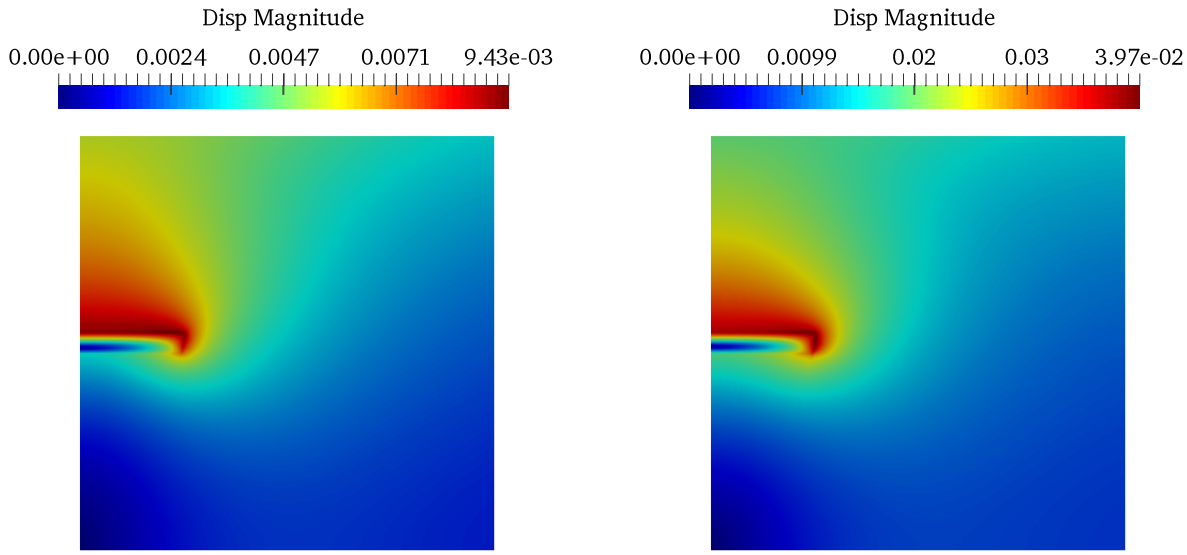

Figure 3: Snapshots of displacement at t=60 days for injection rates of 100 MSCF/day and 400 MSCF/day respectively

stress is $11.085 \times 0.9 + 5 \times 0.1 = 10.4765\, MPa$ at $500\, m$, and $22.17 \times 2.5 \times 0.9 + 24.63 \times 0.1 = 52.3455\, MPa$ at $2500\, m$. We choose a value of 0.7 for the ratio of horizontal to vertical initial total stress. The average bulk density is $\rho_b = 2260 \times 0.9 + 1000 \times 0.1 = 2134 kg/m^3$, the average sonic compressional and shear velocities are $V_p = 730$ m/s and $V_s = 420$ m/s respectively. The Biot coefficient is assumed to be $b = 1.0$. The friction coefficient drops linearly from static friction $\mu_s = 0.5$ to dynamic friction $\mu_d = 0.2$ over $d_c = 5\, mm$. Snapshots of the displacement evolution are given in Fig. 3.

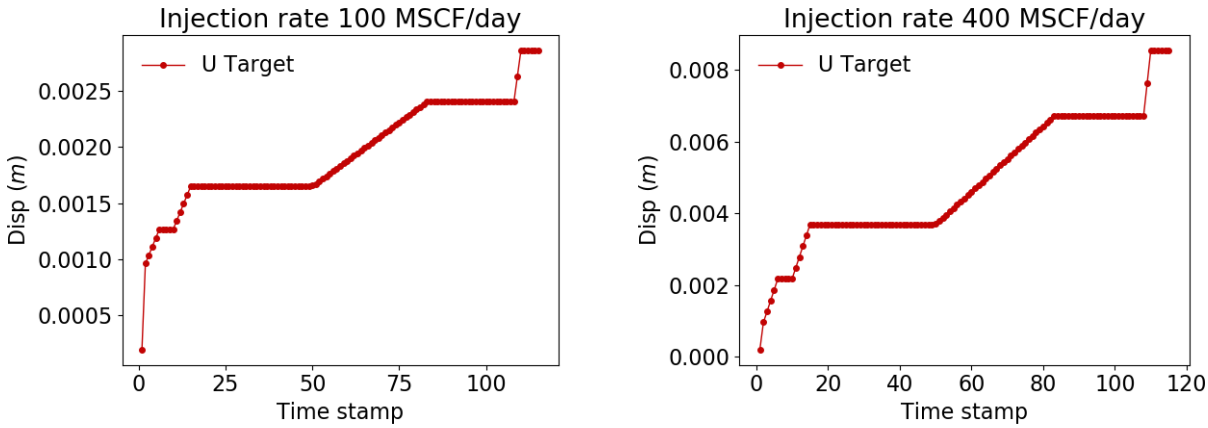

Figure 4: Displacement time series ground truth for u at $(0, 500)$ for injection rates on either end of the data spectrum.

3. Reconstruction using LSTM autoencoders

The simulator spits out vtk files for each time stamp in the simulation. Our job is to extract the displacement values corresponding to grid points on the top surface, and construct a time series as the ground truth, as shown in Fig. 4. A code snippet for processing vtk files is provided in Listing 1.

```
class parse_vtk:
# Base class for parsing vtk files

    def get_surface_information(self, vector_name):
        """
        :vector_name: the vector you want to process
        :return: displacement components
        """
        reader = vtk.vtkDataSetReader()
        reader.SetFileName(self.infile)
        reader.Update()
        data = reader.GetOutput()
        npoints = data.GetNumberOfPoints()
        d = data.GetPointData()
        array = d.GetArray(vector_name)

        u, v, w, x, y, z = np.zeros(npoints),np.zeros(npoints),np.zeros
    (npoints),np.zeros(npoints),np.zeros(npoints),np.zeros(npoints)

        for n in range(npoints):
            x[n], y[n], z[n] = data.GetPoint(n)
            u[n], v[n], w[n] = array.GetTuple(n)

        # Surface information at min x and max y
        u = u[np.where((x==min(x)) & (y==max(y)))[0]]
```

```
        v = v[np.where((x==min(x)) & (y==max(y)))[0]]

        del x, y, z
        return np.sqrt(u**2+v**2)
```

Listing 1: Code for processing vtk files

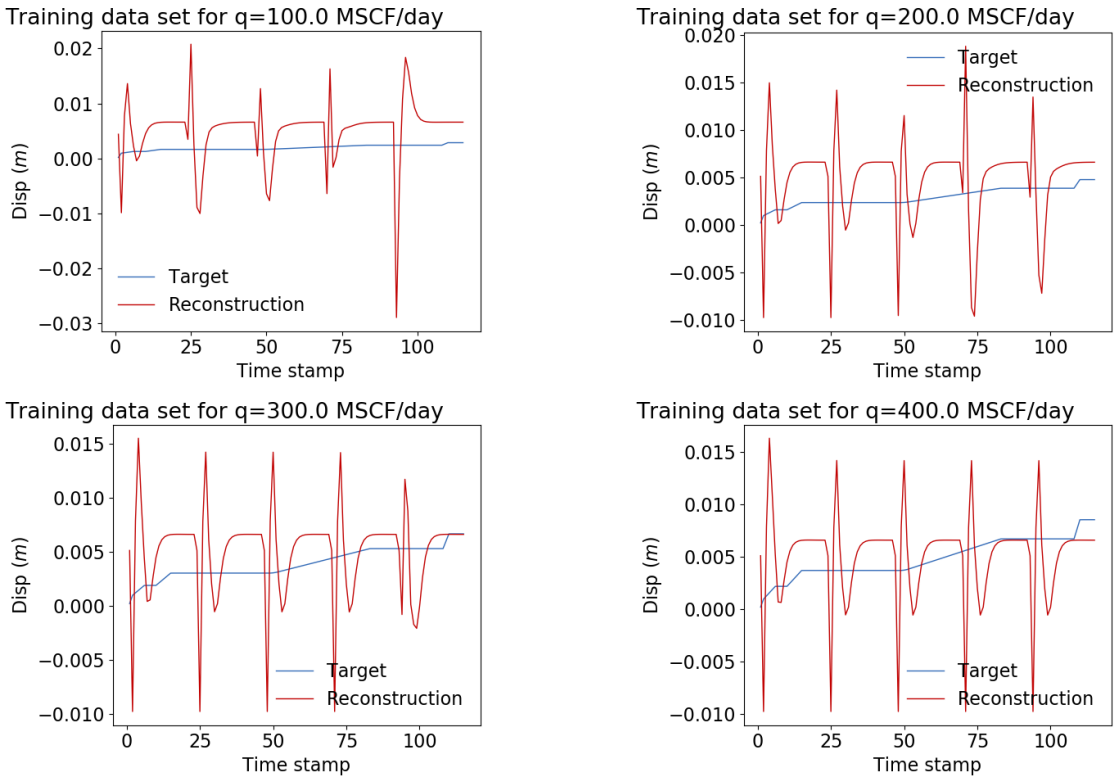

Figure 5: Reconstruction of displacement time series ground truth for u at $(0, 500)$ for different injection rates

A reduced order model would effectively mean reconstructing this time series using LSTM autoencoder, and the optimal deep learning parameters to best reconstruct the time series. The deep learning piece is built on the PyTorch framework, and all simulations are run on a basic AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx $\times$ 4 processor. We use a window size of 23 for 115 time steps, hidden state size half the window size, number of LSTM layers per encoder and decoder is 1, and we deploy the Adam optimizer to train the model using only 10 epochs to avoid overfitting. The ratio of the window to the hidden state size is a measure of the amount of compression that is imposed while encoding the information using the encoder. We observe from Fig. 5 that LSTM autoencoders are meant for oscillating time series, because they have this compulsive behavior to want to capture oscillations. Nevertheless, the mean behavior is captured correctly, and that is good enough for inference in the robust Bayesian framework. We have to keep in mind though, that this is a toy problem, and the ground displacement time series due to seismic activity is typically always oscillatory, as is evident from seismogram data. So we stick to this deep

learning architecture for problems in this realm going forward. A code snippet is provided in Listing 2.

```
class lstm_encoder(nn.Module):
# Encodes time-series sequence

    def __init__(self, input_size, hidden_size, num_layers):
        super(lstm_encoder, self).__init__()
        self.lstm = nn.LSTM(input_size = input_size, hidden_size=
    hidden_size, num_layers = num_layers)

    def forward(self, x_input):
    # called internally by PyTorch
        lstm_out, self.hidden = self.lstm(x_input.view(x_input.shape
    [0], x_input.shape[1], self.input_size))
        return lstm_out, self.hidden

class lstm_decoder(nn.Module):
# Decodes hidden state output by encoder

    def __init__(self, input_size, hidden_size, num_layers):
        super(lstm_decoder, self).__init__()
        self.lstm = nn.LSTM(input_size = input_size, hidden_size =
    hidden_size, num_layers = num_layers)
        self.linear = nn.Linear(hidden_size, input_size)

    def forward(self, x_input, encoder_hidden_states):
    # called internally by PyTorch
        lstm_out, self.hidden = self.lstm(x_input.unsqueeze(0),
    encoder_hidden_states)
        output = self.linear(lstm_out.squeeze(0))
        return output, self.hidden
```

Listing 2: LSTM autoencoder code snippet

4. The formalism of Bayesian inference with Markov chain Monte Carlo sampling

The Bayesian inference framework works on the basic tent of uncovering a distribution centered around the true value and starts off with an initial guess for the distribution also called "prior" $\mathcal{D}$ to eventually get to the most accurate distribution possible also called "posterior" $\mathcal{P}$ through a likelihood $\mathcal{L}$. The Bayes theorem in a nutshell is:

$$\mathcal{P} = \frac{\mathcal{D} \times \mathcal{L}}{\int \mathcal{D} \times \mathcal{L}} \tag{9}$$

The prior is typically taken to be a Gaussian distribution and the likelihood carries information about the forward model. The quantity that makes evaluation of the posterior difficult is the integral term in the denominator. Since direct evaluation of the integral using quadrature rules is expensive, sampling methods like Markov chain Monte Carlo (MCMC) [15–18]

are used. To put it mathematically, if we were evaluating an integral, then the sampling would apply to points at which we know the value of integrand, and then proceed to evaluate the integral. But if the integrand at each of those points is a distribution rather than a value, it makes the sampling and subsequent averaging significantly more complicated. By constructing a Markov chain that has the desired distribution as its equilibrium distribution, one can obtain a sample of the desired distribution by recording states from the chain. The more steps are included, the more closely the distribution of the sample matches the actual desired distribution.

4.1. Applied to our problem

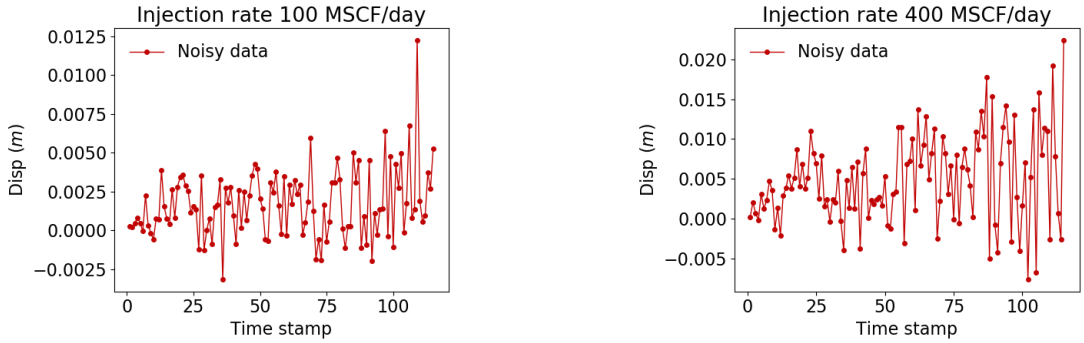

Figure 6: Displacement time series noisy data for u at $(2000, 500)$ for injection rates on either end of the data spectrum. The noisy data is generated from the ground truth using a Gaussian distribution

In this particular inverse problem, the displacement response of the model is known and the goal is to estimate the injection rate q. To formalize the problem, consider the relationship between displacement $u(t)$ and the forward model $\mathcal{F}(\theta)$ with model parameters, constants and variables θ by the following statistical model

$$u(t) = \mathcal{F}(\theta) + \epsilon \tag{10}$$

where ϵ is the noise. Assuming the $\epsilon \sim N(0, \sigma^2)$ as unbiased, independent and identical normal distribution with standard deviation σ allows us to conveniently generate the synthetic data as shown in Fig. 6. The goal of the inverse problem is to determine the model parameter distribution as follows

$$\pi(q|u(t_1), ..., u(t_n)) = \frac{\pi(u(t_1), ..., u(t_n)|q)\pi_0(q)}{\int_q \pi(u(t_1), ..., u(t_n)|q)\pi_0(q)dq} \tag{11}$$

where $\pi_0(q)$ is the prior distribution and $\pi(u(t_1), ..., u(t_n)|q)$ is the likelihood given by

$$\pi(u(t_1), ..., u(t_n)|q) = \prod_{i=1}^{n} \pi(u(t_i)|q) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{u(t_i)-\mathcal{F}(\theta)}{\sigma}\right)^2} \tag{12}$$

4.2. Adaptive metropolis algorithm

The adaptive Metropolis algorithm [17] explores the parameter space with specified limits ranging from a high of q^u to a low of q^l and starts from a random initial guess of the model parameter q^m, where m is the iteration number. The initial covariance matrix in the adaptive Metropolis algorithm is constructed using the initial parameter $q^{m=0}$. At each iteration, the steps are

1. A random parameter sample q^* is generated from the proposal distribution
2. If q^* is not within the specified limits, $q^* \notin (q^l, q^u)$, the iteration is passed without moving to the next steps, and the previous sample is considered as the new sample, $q^{m+1} = q^m$
3. If q^* is within the specified limits, $q^* \in (q^l, q^u)$, a new value of standard deviation associated with q^* is generated using the inverse-gamma distribution
4. q^* is accepted as the new sample $q^{m+1} = q^*$ if a criterion which involves the standard devation is met
5. The covariance matrix is updated if the iteration number is an exact multiple of m_0 using the previous m_0 model parameters

The simulation is repeated for n iterations and the parameter samples resulting from all these iterations represent the parameter posterior distribution. The construction of the q^* is only based on the current parameter, q^m, which is the Markov process. The computational time of the MCMC sampling method is proportional to the number of generated samples n. To put it more succinctly, the algorithm is elucidated in Algorithm 2.

5. Results

Figs. 7- 10 are results of the Bayesian/MCMC inference framework for the different injection rates in the spectrum for different number of samples. We start with an initial guess of 1 MSCF/day for all Bayesian/MCMC simulations, which is way off the desired estimated value. In reality, this tests the robustness of the framework, as the initial guess in realistic scenarios is expected to be way off the desired estimated value because we do not know the desired estimated value. The interval of adapting the covariance matrix is 100, which means the matrix is modified every 100 samples. Also, since the initial guess is more often than not way off the desired value, the initial half of the number of samples are burnt-in, which is common practice in MCMC simulations. We observe from the results that the estimation is evidently impacted by how good the reduced order model is in the first place, and we know from Fig. 5 that LSTM autoencoders do a lot better when the time series is oscillatory more than any other feature. We also observe that the estimation is not always monotonically converging to the ground truth with increasing number of samples, but is expected to beyond a certain number of samples, which is a function of the value itself, the reduced ordel model, and how well the reduced order model works for the value in and around the ground truth. The reality is that the time series across the injection rates in the spectrum of the generated data spans an order of magnitude, and to fit all that into a LSTM autoencoder in a one size fits all manner is not a trivial task.

Algorithm 2: Metropolis-Hastings algorithm

```
m ← 0;
q^m ← q^0;                                          // q^0 is an initial guess
V ~ q^m;                                            // Construct covariance matrix
for j=1,2,...,n do
    q* ~ q^m, V;                                    // Randomly sample from a distribution with covariance V
    if q* ∉ (q^l, q^u) then
        continue;                                   // Move on if the guess is off the specified limits
    else
        γ ~ q*;                                     // Get standard deviation
        if f(γ, q*) then
            q^{m+1} ← q*;                           // Accept sample based on a condition f(γ, q*) being satisfied
        else
            q^{m+1} ← q^m;                          // Reject sample
    m ← m + 1;
    if m%m_0 then
        V ~ {q^m, q^{m-1}, .., q^{m-m_0}};          // Update covariance matrix every m_0 iterations
```

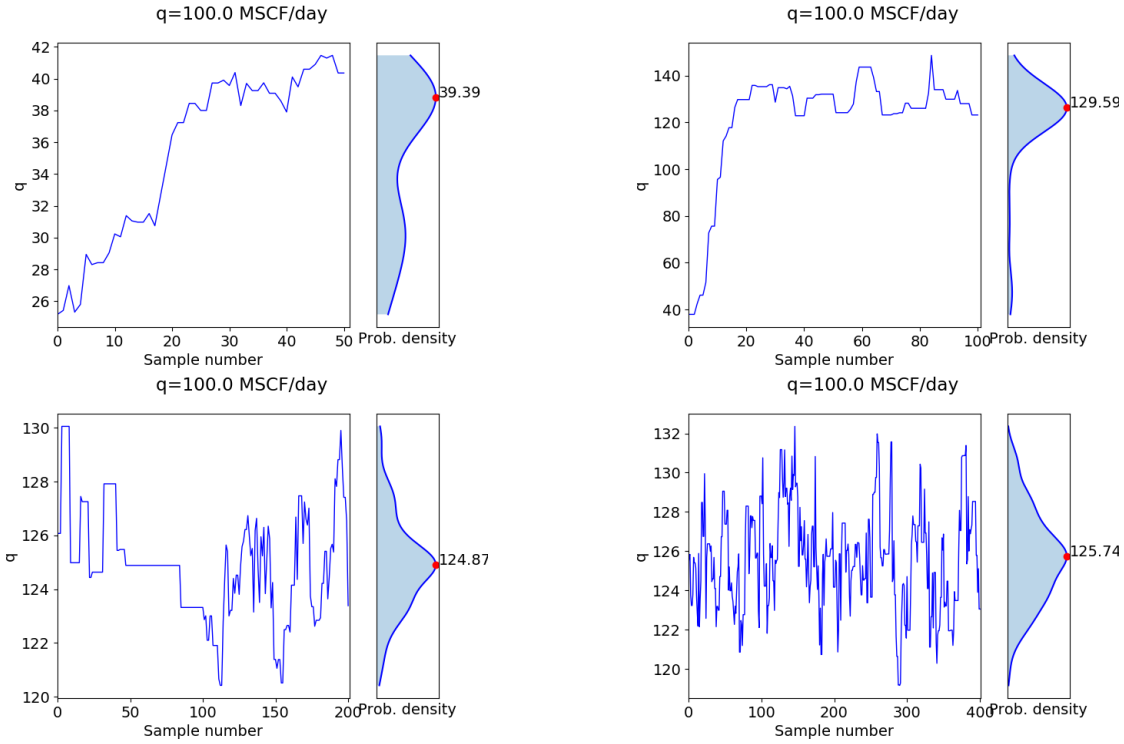

Figure 7: Inversion results for injection rate of 100 MSCF/day

6. Conclusions and outlook

To summarise the procedural framework in this work again, the steps we followed were:

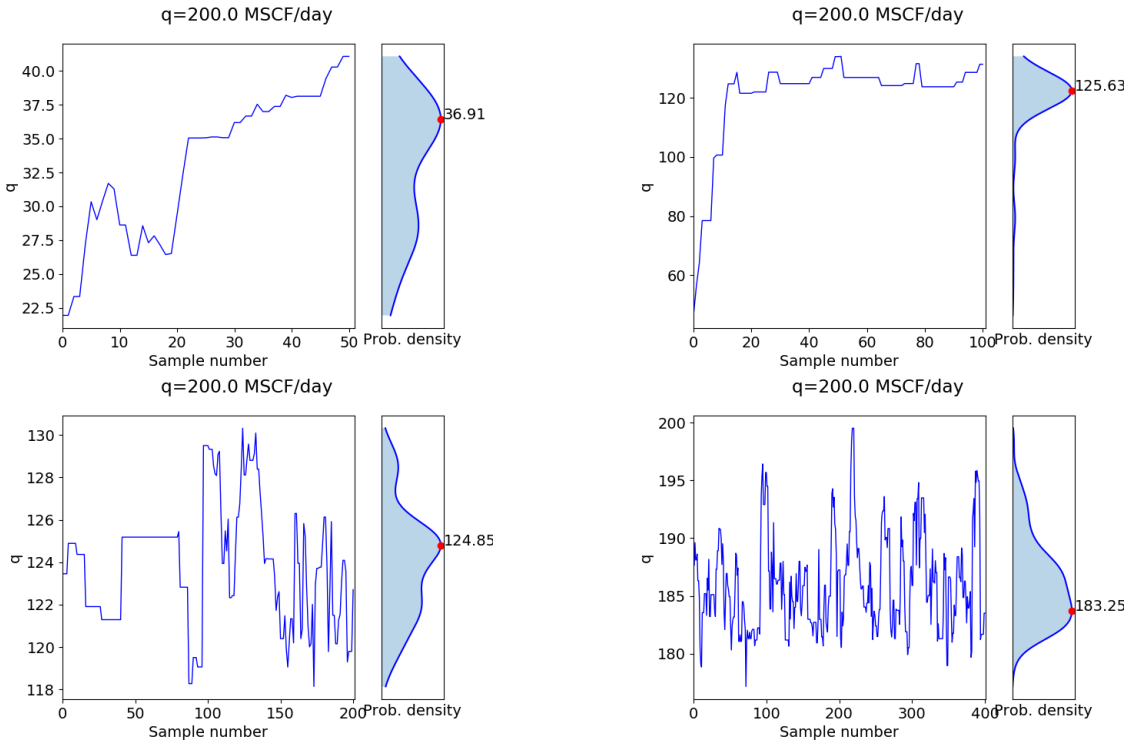

Figure 8: Inversion results for injection rate of 200 MSCF/day

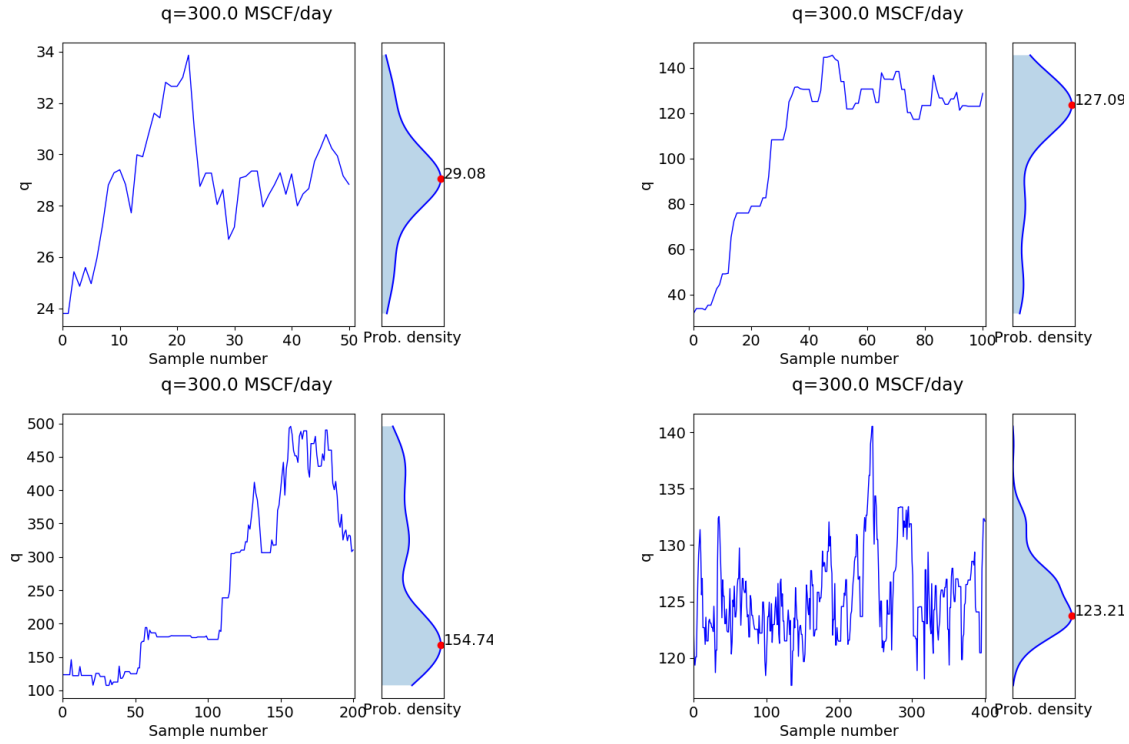

Figure 9: Inversion results for injection rate of 300 MSCF/day

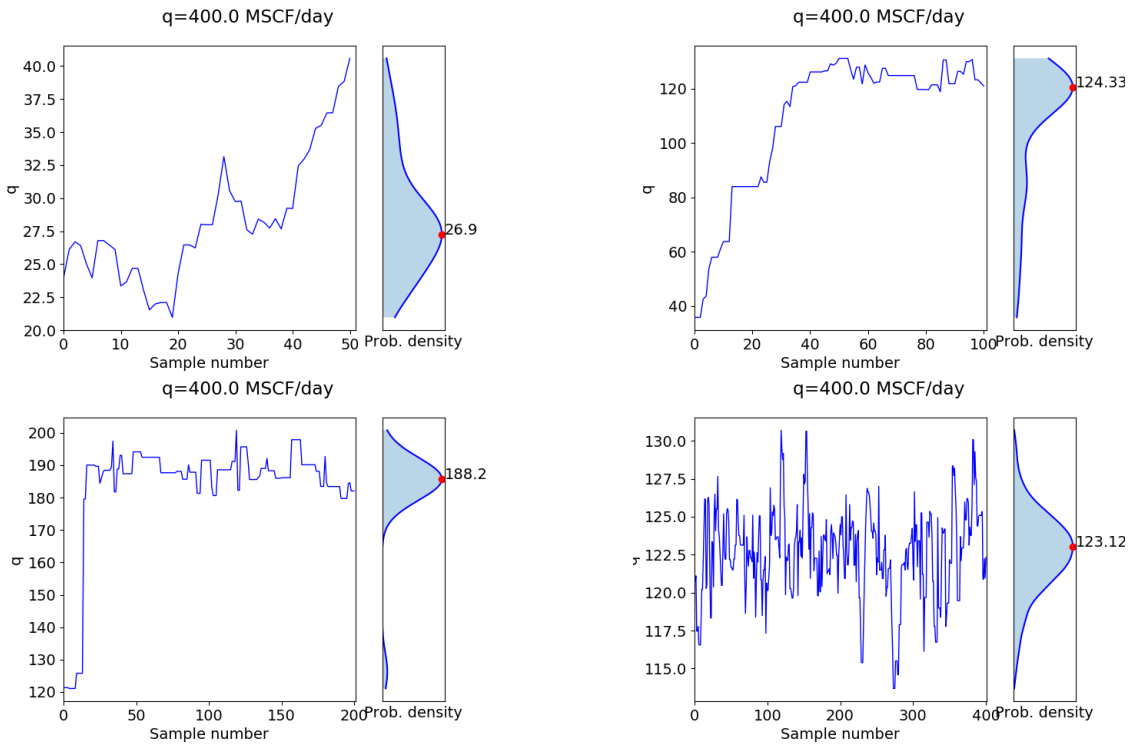

Figure 10: Inversion results for injection rate of 400 MSCF/day

- Run high fidelity simulations for a bunch of injection rates
- For each injection rate, construct a displacement time series at a chosen grid point on the ground surface
- Add noise to the time series to eventually serve as noisy data for the Bayesian inference framework
- Train the LSTM autoencoder with time stamp and injection rate as input and displacement time series (without the noise) as the target
- Run the Bayesian inference framework with the LSTM autoencoder based reduced order model
- Test the robustness of the framework by comparing the estimates of the injection rate with the ground truth injection rate

We observed that the Bayesian/MCMC performance is squarely a function of how well the reduced order model replicates the high fidelity model, and while this is a good sign as the Bayesian/MCMC piece is robust, it lends to more future work in the realm of doing a good job of model order reduction. In this realm, decompositions play a role in the form of principal component analysis of the time stamps of the solution vector from the high fidelity, and it remains to be seen how to tie in that analysis into a sophisticated forward model using a framework like PyTorch. The author is aware of such frameworks being increasingly developed, and that is the ballpark in terms of future work of developing the software package.

References

[1] S. Dana, K. Reddy, Bayesian inference and markov chain monte carlo based estimation of a geoscience model parameter, arXiv preprint arXiv:2201.01868 (2021).

[2] S. Dana, K. R. Lyathakula, Uncertainty quantification in friction model for earthquakes using bayesian inference, arXiv preprint arXiv:2104.11156 (2021).

[3] S. Dana, M. F. Wheeler, Convergence analysis of fixed stress split iterative scheme for anisotropic poroelasticity with tensor biot parameter, Computational Geosciences 22 (2018) 1219–1230.

[4] S. Dana, M. F. Wheeler, Convergence analysis of two-grid fixed stress split iterative scheme for coupled flow and deformation in heterogeneous poroelastic media, Computer Methods in Applied Mechanics and Engineering 341 (2018) 788–806.

[5] S. Dana, B. Ganis, M. F. Wheeler, A multiscale fixed stress split iterative scheme for coupled flow and poromechanics in deep subsurface reservoirs, Journal of Computational Physics 352 (2018) 1–22.

[6] S. Dana, S. Srinivasan, S. Karra, N. Makedonska, J. D. Hyman, D. O'Malley, H. Viswanathan, G. Srinivasan, Towards real-time forecasting of natural gas production by harnessing graph theory for stochastic discrete fracture networks, Journal of Petroleum Science and Engineering 195 (2020) 107791.

[7] S. Dana, M. Jammoul, M. F. Wheeler, Performance studies of the fixed stress split algorithm for immiscible two-phase flow coupled with linear poromechanics, Computational Geosciences (2021) 1–15.

[8] S. Dana, J. Ita, M. F. Wheeler, The correspondence between voigt and reuss bounds and the decoupling constraint in a two-grid staggered algorithm for consolidation in heterogeneous porous media, Multiscale Modeling & Simulation 18 (2020) 221–239.

[9] S. Dana, Addressing challenges in modeling of coupled flow and poromechanics in deep subsurface reservoirs, Ph.D. thesis, The University of Texas at Austin, 2018.

[10] L. Gasparini, J. R. Rodrigues, D. A. Augusto, L. M. Carvalho, C. Conopoima, P. Goldfeld, J. Panetta, J. P. Ramirez, M. Souza, M. O. Figueiredo, V. M. Leite, Hybrid parallel iterative sparse linear solver framework for reservoir geomechanical and flow simulation, Journal of Computational Science 51 (2021) 101330.

[11] A. Olivier, D. G. Giovanis, B. Aakash, M. Chauhan, L. Vandanapu, M. D. Shields, Uqpy: A general purpose python package and development environment for uncertainty quantification, Journal of Computational Science 47 (2020) 101204.

[12] J. C. Jaeger, N. G. W. Cook, Fundamentals of Rock Mechanics, Chapman and Hall, London, 1979.

[13] H. A. Van Der Vorst, Krylov subspace iteration, Computing in science & engineering 2 (2000) 32–37.

[14] F. Cappa, J. Rutqvist, Impact of CO_2 geological sequestration on the nucleation of earthquakes, Geophys. Res. Lett. 38 (2011) L17313.

[15] A. Shapiro, Monte carlo sampling methods, Handbooks in operations research and management science 10 (2003) 353–425.

[16] W. K. Hastings, Monte carlo sampling methods using markov chains and their applications (1970).

[17] H. Haario, E. Saksman, J. Tamminen, An adaptive metropolis algorithm, Bernoulli (2001) 223–242.

[18] C. L. Mueller, Exploring the common concepts of adaptive mcmc and covariance matrix adaptation schemes, in: Dagstuhl Seminar Proceedings, Schloss Dagstuhl-Leibniz-Zentrum fÃ1/4r Informatik, 2010.