POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Homework report

PoreWorld: an upscaling strategy in porous media

Scientific computing tools for advanced mathematical modelling

Authors: Carlo Laurenti Argento, Davide Galbiati and Alessandro Grignani

Academic year: 2023-2024

---

1. Small abstract

Porous media are materials with voids that can be filled with fluids like air, water, or oil, and are found in both natural and technological contexts. This study focuses on geological porous media, such as sedimentary layers and porous rocks, but the concepts and techniques discussed are also applicable to industrial and biological porous media. A porous medium consists of a solid matrix and pores that may contain one or more fluid phases. The subsequent sections will concentrate on single-phase flows within these media.

2. Mathematical formulation of the problem

Pore widths in porous media typically range in the scale of nanometers (nm), but industrial applications span much larger scales, usually kilometers (km). Solving problems at the pore scale is often impractical, so a macroscopic representation using the concept of Representative Elementary Volume (REV) is more appropriate, still preserving main physical features. The REV method assigns properties like porosity or permeability to a mathematical point, based on the average properties of a surrounding volume. This approach uses macroscopic equations to describe physical processes, relying on averaged properties rather than detailed microscopic configurations.
Let Ω_0 be a spherical sub-domain of Ω centred in x_0 with radius r. We plot the averaged property as a function of the radius of Ω_0, the correct radius for a REV is found in the flat part of the graph between l and L. Smaller radius would collocate the method as a microscopic scheme, which we do not want. Instead, larger radius would not capture the heterogeneity of the material. The void space indicator function at the microscopic level is given by:

$$i(x,t) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ belongs to the non-solid part at time } \mathbf{t} \\ 0 & \text{if } \mathbf{x} \text{ belongs to the solid part at time } \mathbf{t} \end{cases}$$

Moreover, we define porosity of a REV centered in $\mathbf{x_0}$ as

$$\phi(\mathbf{x_0}, \mathbf{t}) = \frac{1}{|\Omega_0|} \int_{\Omega_0} i(\mathbf{x}, t) d\mathbf{x} \tag{1}$$

To treat this kind of problem we necessarily have to refer to the Darcy law which states:

$$q_D = \kappa A \frac{h_2 - h_1}{l} \tag{2}$$

where q_D is the volumetric flow, the length difference is the variation of altitude between inlet and outflow of a pipe, A is the section, and κ is denoted as hydraulic conductivity. Additionally, linking Darcy law (in its extended n = 1,2 version) to pressure, which is a paramount quantity in this problem, is viable by means of Stevin's law, thus entailing:

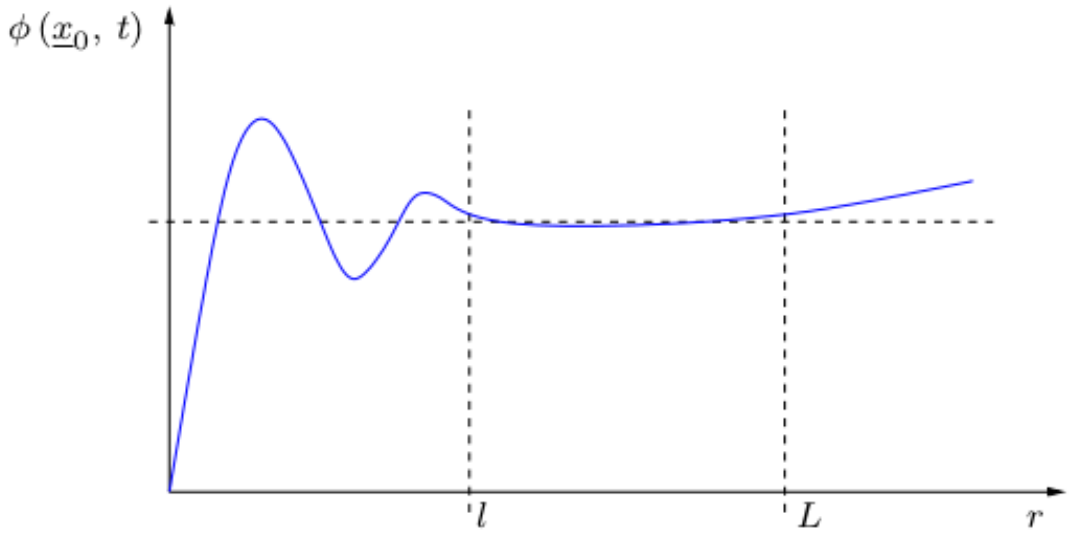


Figure 1: Porosity behaviour

$$\mathbf{q} = -\kappa \nabla h = -\kappa \nabla (\frac{p}{\rho g} + z) \tag{3}$$

It's obvious that now the flow has a direction, consisting of more components with different magnitudes, while hydraulic conductivity, that embeds the properties of material and of the phase is, becomes a tensor.
Rewriting hydraulic conductivity in terms of properties of the matrix, through permeability matrix K, and phase viscosity μ we get finally:

$$\mathbf{q} = -\frac{K}{\mu}(\nabla p + \rho g \nabla z) \tag{4}$$

Finally we resort Darcy equation and, by imposing mass conservation on a REV, we can assure the well position of the following problem which we will try to solve:

$$-\nabla \cdot (1/\mu K \nabla p) = \psi \qquad \text{in } \Omega \tag{5}$$
$$-1/\mu K \nabla p \cdot \mathbf{n} | \partial_{e\Omega} = u_e t \qquad \text{on } \partial_e \Omega \tag{6}$$
$$p | \partial_n \Omega = p_n \qquad \text{on } \partial_n \Omega \tag{7}$$
$$-1/\mu K \nabla p \cdot \mathbf{n} | \partial_{m\Omega} - k_m p | \partial_{m\Omega} = -k_m p_m \qquad \text{on } \partial_m \Omega \tag{8}$$

Among the many methods that could be used to solve the equation, we make use of a cell-centered finite volume method called as Multi-Point Flux Approximation (MPFA-0). The approach resembles the Two-Point Flux Approximation (TPFA) [1], but is suitable for more general K's.
Typical porous media simulators handle millions of computational cells, but geological characterizations require finer meshes. Due to parameter uncertainty, multiple geological realizations are necessary, making fine grid simulations computationally impractical; thus, reliable upscaled models are needed for comprehensive risk and uncertainty assessment.
Let K be the fine scale permeability discussed up to now and K* the upscaled permeability on a coarser mesh.For V being a subdomain, corresponding to a coarse cell, assumed to be a 2D quadrilateral $(0, L_1) \times (0, L_2)$, we can discretize by a finer grid. We split now the problem introduced above in two sub-systems (horizontal and vertical) enforcing an horizontal and vertical pressure gradient and null source/sink as:

$$\nabla \cdot (-K \nabla p_H) = 0 \qquad \text{in V} \tag{9}$$
$$p_H(0, y) = L_1 \tag{10}$$
$$p_H(L_1, y) = 0 \tag{11}$$
$$-K \nabla p_H \cdot \mathbf{n} = 0 \qquad \text{on } (x, 0) \cup (x, L_2) \tag{12}$$

$$\nabla \cdot (-K\nabla p_V) = 0 \qquad \text{in V} \tag{13}$$
$$p_V(x, 0) = L_2 \tag{14}$$
$$p_h(x, L_2) = 0 \tag{15}$$
$$-K\nabla p_V \cdot \mathbf{n} = 0 \qquad \text{on } (0, y) \cup (L_1, y) \tag{16}$$

Assuming the availability of coarse velocity and pressure gradient vector obtained by integral averaging finer data we could estimate the value of K^*:

$$\langle q_H \rangle = -K^* \langle \nabla p_H \rangle \tag{17}$$
$$\langle q_V \rangle = -K^* \langle \nabla p_V \rangle \tag{18}$$

The solution of this equations gives the components of the upscaled tensor, which, however, is not guaranteed to be symmetric. The upscaled tensor is constant in each coarse cell and describes fine scale effects in an averaged way. For this reason it can be non-isotropic, with an alignment of the principal axes that reflects the underlying distribution of high/low permeability fine cells.

Checkpoint 1: the first checkpoint follows the very general formulation of the problem, namely finding a field of upscaled permeability field, starting from pressure gradient and velocity. After averaging this quantities a simple solution of a linear system is necessary to achieve the goal.

Checkpoint 2: the second checkpoint is an optimization problem. In particular, four injection wells are disposed at the corners of the domain, where the function `solve_fine()` returns the maximum pressure at the four wells on the basis of an inside-domain extraction well. Finding the optimal position such that the maximum of the pressures is minimized (thus minimizing the pressure at every well) was the request.
The pressure field is found by solving the following Poisson problem for p:

$$\begin{cases} \nabla \cdot (-k\nabla p) = f + b & \text{in } \Omega \\ \nabla p \cdot \nu = 0 & \text{on } \partial\Omega \\ p(x_e, y_e) = 0 & \text{at } (x_e, y_e) \text{ position of the extraction wells} \end{cases}$$

where:

- k is the permeability matrix;
- Ω is the domain of interest with boundary $\partial\Omega$;
- f is the source term (we set $f = 0$);
- b encodes a unitary injection rate located in the four corners of the domain.

The problem of finding the optimal location of the extraction well can be formulated as:

$$\min_{(x,y)\in\Omega} \left\{ \max_{i=1,\dots,4} p(x_i, y_i) \right\}$$

where (x_i, y_i) represents the position of the i-th injecting well.
Unfortunately the original grid is too fine to achieve the minimization in a reasonable time. As a matter of fact, the geological layers available are 84 and finding the position for one single layer takes more that 4 h in parallel. For this reason, a more clever strategy to achieve a similar result, yet approximated, should be followed. Our idea, which will be deeply explained in the next section, is based on solving the problem with two coarse grids of 100 and 400 quads. Then some points are properly chosen to identify an area on which apply the precise minimization. Additionally, other areas in proximity of the corners are checked.

Checkpoint 3: the third checkpoint focused on developing a surrogate model to predict the outflow concentration of a specific substance at the well position, based on the initial concentration location. The extraction well is positioned at a fixed location, which is identified as the optimal position in **Checkpoint 2**. To ensure accuracy, we conduct a grid search over the entire domain using the solve fine function. The forward problem involves a transport problem influenced by the advective field derived from the Darcy problem addressed in checkpoint 2.
The problem is defined as follows:

$$\begin{cases} \frac{\partial c}{\partial t} + \nabla \cdot (\mathbf{q}c) = 0 & \text{in } \Omega \\ c(x, y, 0) = 1 & \text{in } I \\ c(x, y, 0) = 0 & \text{in } \Omega \setminus I \end{cases} \tag{19}$$

with $I \subseteq \Omega$ being a small squared subset located in the domain and $\mathbf{q} = -K\nabla p$ being the flux computed with the Darcy problem of **Checkpoint 2**.

The objective of this checkpoint was therefore to create a surrogate model that, given a position of the "initial concentration" as input, could approximate the outflow profile (a 2D function) of the concentration extraction at the well point. The algorithm should generate, for layers [51, 71, 42, 20, 12, 2, 82] the outflow profile based on the initial concentration location and the selected layer.

3. Methods

To address the following challenges, we strongly relied on a python package called porepy. Full documentation can be found at [1].

Checkpoint 1: First of all we completed the lacking structures of the upscaling:

```
def upscale(sd, perm, direction, export_folder=None):
```

In particular, we construct the matrix and the right hand side to solve the linear system using the Mpfa method and solving the system allows us to find the pressure. The structures to return the averaged vectorial fields are already implemented.

```
pfa = pp.Mpfa(key)
pfa.discretize(sd, data)
A_pfa, b_pfa = pfa.assemble_matrix_rhs(sd, data)
p = sps.linalg.spsolve(A_pfa, b_pfa)
```

Then we implement the computation of the upscaled vectors by assembling the average vector and matrix. It's important here to enforce the symmetry condition by adding a fifth line to the right tensor. For this reason, a method that does not need to invert the left matrix is needed. We chose least square method.

```
def compute_tensor(grad_h, grad_v, q_h, q_v):

## Assembling spd matrix for least squares
A = np.array([[grad_h[0], grad_h[1], 0, 0],
              [0, 0, grad_h[0], grad_h[1]],
              [grad_v[0], grad_v[1], 0, 0],
              [0, 0, grad_v[0], grad_v[1]],
              [0, 1, -1,0]])
b = np.array([q_h[0], q_h[1], q_v[0], q_v[1],0])

# Solve the linear system
K_avg = np.linalg.lstsq(A, b,rcond = None)[0]
```

Since this steps could be quite time consuming, especially if we want to compute a very refined coarse grid, efficiency has been preserved thanks to pool.multiprocessing parallelization on both upscaling and computing tensor.

```
...
with Pool() as pool:
    for local_perm, mask in pool.starmap(to_parallel, args, chunksize = 2):
...
result.append(local_perm)
```

Checkpoint 2: Before speaking about our method, it's worth noticing that the package used for parallelization is different from Checkpoint 1. Indeed we used ray which provides a very high speed-up [2] compared to multiprocessing. With a speedup that increases a lot with the number of cores.
First of all we created with a ray-parallelized code of Checkpoint 1 two grids of 100 and 400 quads calling:

```
sd_coarse_100, result_100 = Checkpoint1_solution(selected_layers, ..., 100)
sd_coarse_400, result_400 = Checkpoint1_solution(selected_layers, ..., 400)
```

Then, we call the function solve coarse to find the pressure field on the coarse grids. The coarse function has been developed exactly resembling the the already provided fine solver with 100, 400 upscaled permeabilites. The most relevant part of the code is

```
def solve_coarse(sd, pos_well, kk, ..., dimension = 100):
# Extract the grid for simplicity
    kxx = np.zeros([dimension])
    kxy = np.zeros([dimension])
    kyy = np.zeros([dimension])
    i = 0
    for elem in kk:
        kxx[i] = elem[0]
        kxy[i] = elem[1]
        kyy[i] = elem[3]
        i = i + 1
    # Permeability
    perm_tensor = pp.SecondOrderTensor(kxx=kxx, kxy=kxy, kyy=kyy)
```

We invoked the function choosing a position well chosen in the centre of every coarse REV and order them from the lowest to the greatest. It's important to notice that this procedure is quite empirical and it's a trade-off between accuracy and efficiency. Our choice was strictly contingent, and persecuted on the basis of the result we obtained. Moreover we found out that executing the coarse solver with around 900 quads would result is a much higher resolution time, spoiling efficiency and nevertheless being still very far from accurate results.

We then proceeded in setting up a clustering to identify the zones where the minimum could be found most likely, choosing the first 12 pressures in the 100-coarsening and the first 35 pressures in the 400-coarsening. We then created two clusters using DB-Scan Py-package setting the minimum points distance to be included in the cluster just below the vertical pace of the coarse grids, i.e. 67 and 33. Eventually we overlapped the clusters, that is to say we identify as "hot zones" for our quest only the coordinates (associated to their pressure) of the finer grid that are contained in the coarser one. The other points are discarded. This procedure is graphically described in the following pictures. We include also a small chunk of code that performs the first clustering.

```
def solve_coarse(sd, pos_well, kk, ..., dimension = 100):
    # Set the minimum distance threshold
    min_distance = 67
    # DBSCAN clustering
    dbscan = DBSCAN(eps=min_distance, min_samples=1)
    cluster_labels = dbscan.fit_predict(coordinates)
    # Creating dictionary to store clusters
    clusters = {}
    for label in set(cluster_labels):
    clusters[label] = [coarse_pressures_100[i] for i, l in enumerate(cluster_labels) if l
        == label]
```

The following step consists of performing a fine solve on the coordinates of the cluster(s) in order to identify the real pressures associated to our positions, so as to identify the variable `best_coarse`. From this position we generate a square area that empirically extends 6-horizontal, vertical spacings right, left, up, down where, for every point `solve_fine` is called. The point of minimum pressure is the solution of our problem, yet not considering angle inspection.
Since we observed that sometimes the minimum could be hardly identified because it was very near some of the corners we made a further angle inspections. Since we thought this would be quite challenging on the side of efficiency we made a physical assumption to discard three of the four injection point. We checked only the well with maximum pressure, believing that if the minimum had to be in such zones it should have been near a zone with high injection pressure. As a matter of fact, having the well near this zone would result in little effort for the pump to inject the fluid. Nonetheless, this hypothesis cannot be proved at all, and it's empirically corroborated by higher obtained grade. Explicitly we generated a roughly squared area that covered an area of

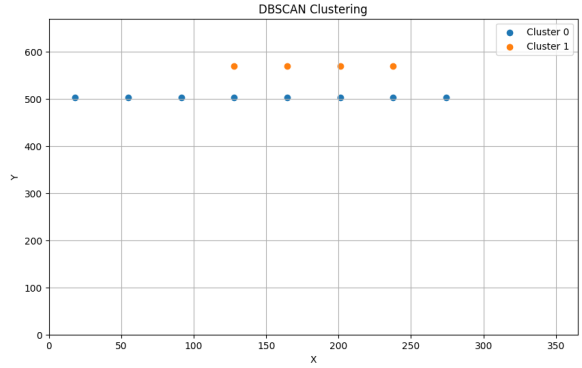


Figure 2: Clustering on sd_{100}

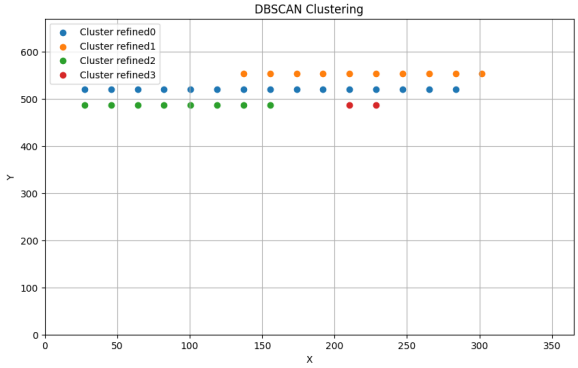


Figure 3: Clustering on sd_{400}

Figure 4: Overlap

```
1.5*sd_coarse_100.spacing[0], 1.2*sd_coarse_100.spacing[1])]
```

Checkpoint 3: for what concerns the third checkpoint, two different mathematical methods were put to test.

3.1. Training an MLP

The first approach consisted in training an MLP. The goal is to train an MLP to learn how to map a given initial configuration of the problem (a layer, the position of the well for that layer and then the initial concentration position) into a small numbers of hyper-parameters through which to rebuild the complete outflow functions. To achieve this, one would first have to build a training set.
The first step was to create the dataset over which to train the NN. This was done by using the forward configuration of the problem, which first solves the Darcy problem for the given configuration and then computes the final outflow. By iterating this process over different configurations, we generated a collection of pairs of configuration conditions and their corresponding outflows.
The inputs of the MLP itself where built in the following way:

- Initial Inputs: Start with the initial concentration coordinates of x and y.
- Neighborhood Points: Create a grid of points around the concentration position using a combination of small and big radii to ensure coverage of the neighborhood. Points are generated using polar coordinates with specified angles to determine their placement.
- Permeability Values: For each generated point, extract the corresponding permeability values and add these values to the neural network inputs list.

Lastly, for every input of the MLP training set, one had to compute the parameters that will map the function describing the final outflow. To fetch these parameters, the real outflow function was transformed in one of two ways, according to the nature of the function itself:

- Normal Distribution: estimate mean, standard deviation, and amplitude
- Exponential Distribution: estimate mean, and the two scale parameters

To sum up, this workflow prepares the input data and target values necessary for training an MLP by considering the spatial positions of concentrations and wells, calculating permeability values from surrounding areas, computing outflow characteristics, and estimating parameters for model outputs.
Unfortunately, at testing time, the MLP didn't live up to expectations. As a matter of fact, during training, the

MLP appeared to perform well. However, when the model was evaluated on the test dataset, its performance was poor. This indicates a significant drop in accuracy and an increase in error when predicting on new, unseen data.
The most important factor that contributed to this discrepancy is that the function that generates the outputs by fitting the outflow data to either normal or exponential distributions assumed that all possible outflow patterns can be accurately represented by these two types of distributions, which may not be the case.
Given this poor performance, a more flexible approach to modeling the outputs that better captures the variability and complexity of the real-world phenomena was necessary. This leads us to our second approach.

3.2. Weighting the neighbouring outflow

This new solution uses a different approach to predict the outflow, leveraging spatial proximity and a weighted combination of outflows from nearby coordinates. This method proved to be more effective and provided significantly better results compared to the previous neural network approach.
This method's implementation is pretty straightforward:

- Start by computing and loading known initial problem configurations with their corresponding outflow data for a selected layer.
- Receive as input a new set of pairs of x and y coordinates
- For each coordinate pair, compute the Euclidean distance from the 4 nearest coordinates which are stored in the dataset computed at the first point. This step identifies how far each precomputed outflow location is from the point of interest.
- To give more importance to closer points, weights representing the proximity to the given point are computed (for each coordinate pair)
- Finally, the results is a composite outflow that is a weighted average of the nearest precomputed outflows, with closer points having a greater influence.

4. Numerical results

Checkpoint 1: The result were satisfying since we achieved the maximum points. Hence the reconstruction of upscaled permeability meets the expected results. We can observe here the original field and the reconstructed one. Also from the point of view of efficiency, results were very good. Partially, some enhancements has been earned by the use of pathos.multiprocessing package that has some advantage over multiprocessing library in Python in the sense that can serialize wider range of functions more efficiently.

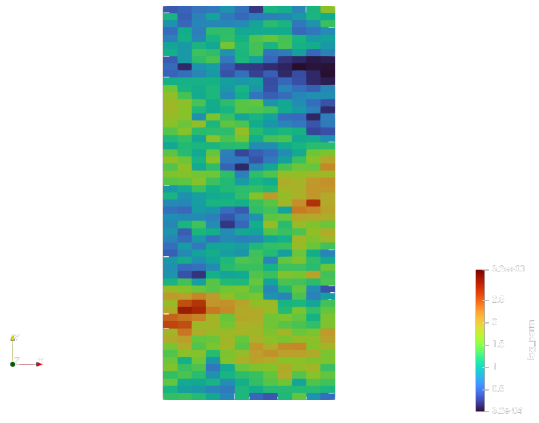


Figure 5: Original kxx-permeability component of layer 10

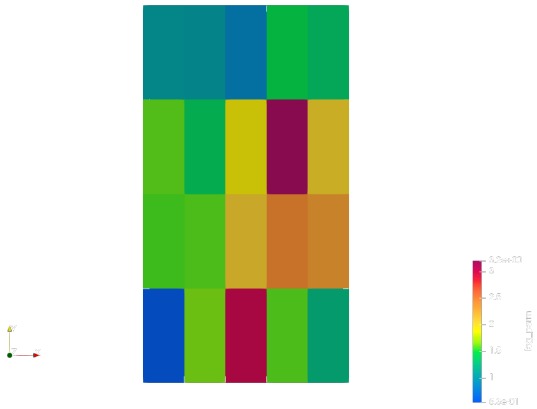


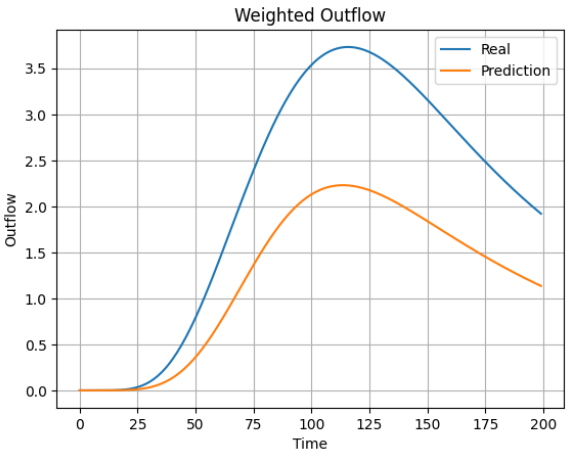
Figure 6: 20-Upscaled kxx-permeability component of layer 10

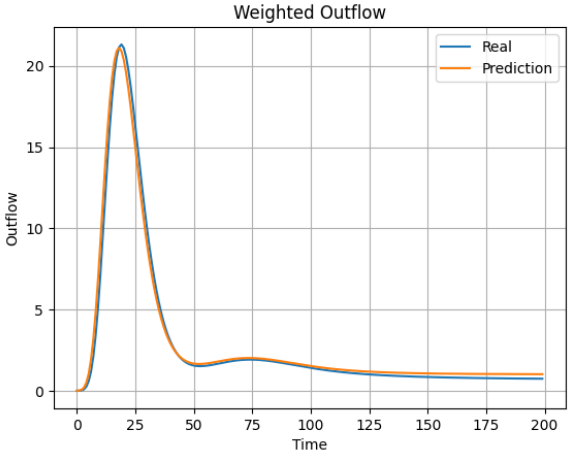
Checkpoint 2: As for Checkpoint 2, the efficiency result was very good. Thanks to ray, the parallelization has a great speed-up, especially due to velocity in accessing the permeability dictionary in the `solve_fine()` function. As for accuracy, results were again satisfying, gaining an accuracy of 0.95/1. We considered this result sufficient even if still improvable. Two main flaws should be taken into account. First of all, if the true minimum is not included neither in the last refinement square nor in the angle-area, the last minimization would be a waste of time and potentially could result in a even less accurate result withe respect to the best coarse result. Moreover, our code does not consider permeability values. With the assumption, that indeed should be verified, that the minimum stands in a point with high porosity some points could be relinquished, thus allowing to

extend the final search. We considered doing so, but unfortunately we did not have time. Moreover, the results of other groups that followed this more involved idea are similar to ours, thus gaining very little more in terms of accuracy. We also show some result on known layers:

Layers	Estimated Coordinates	Real Coordinates
Layer 27	[216.408, 510.540]	[209.216, 544.068]
Layer 47	[210.312, 522.732]	[210.312, 522.732]
Layer 51	[313.944, 597.713]	[313.944, 598.932]
Layer 67	[167.640, 275.844]	[179.832, 272.796]

For what concerns **Checkpoint 3**, the 2nd approach (weighting the neighbouring outflow) was selected. By focusing on the nearest coordinates, this method ensures that the outflow prediction is influenced by spatially relevant data. But most importantly, unlike the previous method (the MLP), which forced outflows into specific distributional forms (normal or exponential), this approach directly utilizes actual outflow data. It doesn't assume any particular distribution, making it more flexible and better suited to capture the true variability of the data.
This second approach provides a more accurate and reliable prediction of outflows, resulting in more robust predictions compared to the previously used neural network model.





5. Conclusions

Overall, we achieved satisfying results in every checkpoint. The only challenge in which we could have spent more effort was Checkpoint 3. The NN approach seemed very promising and we were not willing to put it on the side. Unfortunately, given the particular context (the solutions were pretty fast to compute) and the tight time schedule, we had to turn to a less complex but still very efficient solution.
A lot of effort was also put into finding the best packages and configurations for the parallelization of our algorithms. The final methods we chose were selected after testing multiple different solutions.

6. Bibliography

References

[1] Eirik Keilegavlen, Alessio Fumagalli, Runar Berge, Ivar Stefansson, and Inga Berre. Porepy: An open-source simulation tool for flow and transport in deformable fractured rocks. *CoRR*, abs/1712.00460, 2017.

[2] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. *CoRR*, abs/1712.05889, 2017.