Journal of Computational Science 75 (2024) 102205

Contents lists available at ScienceDirect

Journal of Computational Science

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

Automated parameter tuning with accuracy control for efficient reservoir simulations

Erik Hide Sæternes [a,*], Andreas Thune [a], Alf Birger Rustad [c], Tor Skeie [b,a], Xing Cai [a,b]

[a] Simula Research Laboratory, Kristian Augusts gate 23, Oslo, 0164, Norway
[b] University of Oslo, Department of Informatics, Gaustadalléen 23B, Oslo, 0373, Norway
[c] Equinor Research Centre, Arkitekt Ebbells veg 10, Ranheim, 7053, Norway

A R T I C L E I N F O

Keywords:
Petroleum reservoirs
Reservoir simulation
Parameter tuning
Accuracy control
High performance computing

A B S T R A C T

Computer simulations of complex physical processes typically require sophisticated numerical schemes that internally involve many parameters. Different choices of such internal numerical parameters may lead to considerably different levels of computational efficiency, some may even result in wrong simulation results. The task of finding an optimal set of the numerical parameters (e.g. for the purpose of minimising the simulation time), while ensuring an accepted level of numerical accuracy, is therefore extremely important but challenging. In this paper, we propose a new automated search algorithm that is based on constrained stochastic searches within the parameter space. This iterative search scheme is also equipped with an accuracy check, which adopts several complementary measures for quantifying the similarities between time series from different simulations, such that parameter choices that lead to insufficiently accurate results will be automatically rejected. As a concrete scenario of usage, we have applied the automated parameter search scheme to the open-source reservoir simulation framework OPM. An empirical study shows that a suitable design of the optimisation objective function, together with an appropriate choice of the number of trials per search iteration and the perturbation scale per trial, can produce fast and convergent improvements with respect to the optimisation objective. For example, for a set of 12 numerical parameters, 30 trials from five search iterations are sufficient for reducing the objective function by 30% for the open Norne black-oil reservoir model. The robustness of the automated search scheme is also demonstrated for two other open reservoir models. Moreover, it is found that the parameter values automatically identified for the Norne model can also greatly improve the simulation efficiency of another proprietary reservoir model that has drastically different scale, resolution and geological properties.

1. Introduction

Multi-physics processes are often modelled mathematically by systems of nonlinear partial differential equations (PDEs). In general, analytical solutions to these nonlinear PDEs do not exist, thus approximate solutions have to be sought by some computer code that implements a suitable numerical method. One of the complexities is that a complete numerical algorithm for solving a system of nonlinear PDEs can involve many *numerical parameters* that are not easily chosen but can affect both the resulting accuracy and the needed computing effort. Typical examples of such numerical parameters include the spatial mesh resolution (i.e. the characteristic size of the individual grid cells), the time step size as well as the parameters used to adjust it in the case of adaptive time stepping, the threshold value for checking the convergence of the Newton iterations within each time step that are used to linearise the time-discretised equations, and the convergence threshold value for the linear-system solver iterations within each Newton iteration.

An ideal goal will be to find an optimal set of values for the numerical parameters, such that a numerical solution can be computed using the least amount of effort, while ensuring that the numerical errors are within an acceptable level — e.g. smaller than the errors due to inaccurate physical parameters and computer round-offs. This goal is however still beyond today's knowledge of numerical computing. Instead, we can formulate a more realistic goal as the context of the work to be presented in this paper. That is, we want to develop an *automated* parameter search strategy that can find "optimal" values of the most important numerical parameters, while ensuring that the deviation from a reference numerical solution is within a prescribed

* Corresponding author.
E-mail addresses: erikhs@simula.no (E.H. Sæternes), andreast@simula.no (A. Thune), abir@equinor.com (A.B. Rustad), tskeie@simula.no (T. Skeie), xingca@simula.no (X. Cai).

https://doi.org/10.1016/j.jocs.2023.102205
Received 18 July 2023; Received in revised form 24 October 2023; Accepted 21 December 2023
Available online 27 December 2023

level. Two important remarks are in order here. First, optimality is understood as being able to minimise a prescribed real-valued objective function, which for instance reflects the computing time needed. Second, the reference numerical solution has been computed by using some "conservative" values of the numerical parameters that may require excessive time steps, Newton iterations, and linear iterations. The accuracy of this reference solution is either trusted or validated in some way beforehand.

Although there exists extensive work on automated parameter tuning and the general subject of optimisation in high-dimensional spaces, our work is novel in the following aspects:

- Our automated search algorithm, to be presented in Section 3, is different from the state of the art.
- We have added accuracy control to the individual trials executed within the automated search algorithm, so that any sets of parameter values that lead to insufficiently accurate numerical solutions are automatically discarded.
- We have combined several similarity measures, found in Section 2, to provide extra security in the accuracy control. Several of the adopted similarity measures are not known to have been used in optimisation contexts.

Apart from the above-mentioned novelties, our work will also make contributions to the particular subject of reservoir simulation. We have implemented our automated parameter search strategy and applied it to the *Flow* simulator of multi-phase porous media flows in reservoirs, which belongs to the open-source OPM software framework [1,2]. Extensive experimental results, most of which are based on three open reservoir models, can be found in Section 4. It will be shown that our automated search algorithm can quickly identify close-to-optimal sets of the numerical parameter values of the *Flow* simulator, and the resulting savings in the computing time are substantial. Moreover, the auto-identified sets of numerical parameter values, which are obtained using a small-scale open reservoir model, also work out-of-the-box for one industrial-scale proprietary reservoir model. The resulting time savings for this real-life case are also substantial. We remark that the capability of cheaply identifying economic sets of numerical parameter values is particularly important for large-scale realistic reservoir models, because running simulation ensembles for the uncertainty analysis of the physical parameters is a hugely resource-demanding task. Time saving guarded by accuracy control is therefore highly desirable.

The remaining sections are organised as follows: Section 2 introduces and discusses the similarity measures used to ascertain the accuracy of simulation results, when comparing them to a reference simulation result. Section 3 presents our automated search algorithm used to iteratively find improved sets of parameter values for a target numerical algorithm. Section 4 details the experiments carried out to test the automated search algorithm equipped with the similarity measures as accuracy control. Section 5 gives an overview of the earlier work relevant for our present work. Finally, Section 6 provides a summary of the findings in this paper, as well as a discussion about the limitations and possible future work.

2. Similarity measures for accuracy control

As mentioned in the preceding section, we want to be able to check the accuracy of a new numerical solution by comparing it against an existing reference solution. Therefore, quantification of the difference between two simulation results is needed. In this work, we will make use of several transformations, called *measures*, all of which give out a single number that gauges the degree of similarity between two simulation results. A too high or low value (depending on the measure) will then indicate that a new simulation result is too inaccurate (compared against the reference result) and should be discarded.

In this work, we assume that the simulation results will be in the form of time series. The application scenario chosen for this paper addresses reservoir simulations, and hence an explanation of the relevant time series data is provided here. For reservoir simulations, an important quantity is the pressure at the bottom of the wells that are used to extract oil and gas, or inject water (or CO_2). This is referred to as *well bottom-hole pressure* (WBHP), which is widely used in industrial practice for checking the accuracy of a simulation. Since using the WBHP to identify dissimilarities between different simulations can reveal small inaccuracies, we will make use of the time series of WBHP for each individual well in a reservoir model. Hence, for each well in the model and each similarity measure, we will get one number gauging the similarity between two simulation results. That is, the total number of values that can be used for accuracy control equals the number of wells times the number of similarity measures chosen.

2.1. Resolving non-coinciding time steps

In general, the time series of WBHP for a well is simply an array of positive real numbers, with each value corresponding to one time step of the simulation. For each time series array $\boldsymbol{x} = (x_1, \ldots, x_n)$, there is a corresponding array $\boldsymbol{t} = (t_1, \ldots, t_n)$ giving the time value for each value of $\boldsymbol{x}$. Often, we want to calculate the distance between two time series – one from the new simulation, the other from the reference simulation – which can be represented by two arrays $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$. Note that $\boldsymbol{x}$ and $\boldsymbol{y}$ are not necessarily of equal length, due to adaptive time stepping adopted in the simulations.

The similarity measures to be detailed later in this section, however, all assume that $\boldsymbol{x}$ and $\boldsymbol{y}$ have the coinciding time steps, i.e., the same position in $\boldsymbol{x}$ and $\boldsymbol{y}$ corresponds to the same time point. Although some reservoir models do dictate certain time points which all simulations need to include — that is, at which they are forced to make a stop, these dictated time points are often too coarse to be suitable for a detailed comparison between two simulations. In order to utilise all the available information from the simulations and handle the situation of not having coinciding time steps, we need to perform some sort of interpolation to fill in the missing time steps in each simulation.

The interpolation is performed by starting at the initial time point and identifying the array which is closest forward in time (say, without loss of generality, that it is $\boldsymbol{x}$). We then insert the closest forward time point into the other time series (in this case $\boldsymbol{y}$) by performing an interpolation (see below). We continue this process until there are no more time points left. In the end, we will have two longer arrays representing two refined time series that have coinciding time steps.

The most obvious choice is a standard linear interpolation [3], which is both easy to implement and easy to visualise. Assume that we have two time points t_i and t_{i+1} with the corresponding values x_i, x_{i+1} contained in $\boldsymbol{x}$. If we add an extra time point $\tilde{t}$, satisfying $t_i < \tilde{t} < t_{i+1}$, our corresponding new value $\tilde{x}$ becomes

$$\tilde{x} = x_i + \frac{\tilde{t} - t_i}{t_{i+1} - t_i}(x_{i+1} - x_i). \tag{1}$$

The above linear interpolation might not make sense for all quantities, though. In particular, a more suitable approach when dealing with WBHP is to use a step (piecewise constant) function. More concretely, we choose between two time points the value corresponding to the later time point. For $t_i < \tilde{t} < t_{i+1}$ as above, we therefore simply get

$$\tilde{x} = x_{i+1}. \tag{2}$$

2.2. The similarity measures

In this work, we will use the following five measures: correlation, cosine similarity, an L^2-inspired measure, a relative L^2 measure, and the Jensen–Shannon divergence. It should be noted that these five measures by no means constitute an exhaustive list of relevant measures, and that several other measures could potentially be substituted for our choice here and yield good results. Throughout this section, we assume that we have two arrays $\boldsymbol{x} = (x_1, \dots, x_{N_t})$ and $\boldsymbol{y} = (y_1, \dots, y_{N_t})$ with coinciding time steps of size N_t.

The correlation [4] between them is given by

$$\rho_{\boldsymbol{xy}} = \frac{\sum_{i=1}^{N_t}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N_t}(x_i - \bar{x})^2 \sum_{i=1}^{N_t}(y_i - \bar{y})^2}}, \tag{3}$$

with $\bar{x}$ being the arithmetic mean of $\boldsymbol{x}$ (and similarly for $\bar{y}$). The value of $\rho_{\boldsymbol{xy}}$ will lie somewhere between -1 (when the arrays are perfectly negatively correlated) and 1 (when we have perfect correlation). A value of 0 means no correlation. As we would like our arrays to be similar, we want them to be highly correlated. In other words, we would like $\rho_{\boldsymbol{xy}}$ to be close to 1. A value below some threshold will then indicate that the deviation for the new simulation result is too large.

The cosine similarity [5] is given by

$$\cos_{\boldsymbol{xy}} = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} = \frac{\sum_{i=1}^{N_t} x_i y_i}{\sqrt{\sum_{i=1}^{N_t} x_i^2}\sqrt{\sum_{i=1}^{N_t} y_i^2}}. \tag{4}$$

If we view $\boldsymbol{x}$ and $\boldsymbol{y}$ as vectors in an N_t-dimensional space, the cosine similarity is the cosine of the (smallest) angle between them. As for the correlation, the cosine similarity takes values from -1 to 1, taking the value 1 when the arrays are identical, and taking the value -1 when the arrays are parallel but pointing in opposite directions. We want the value to be as close to 1 as possible — or rather, we do not want the value to be smaller than 1 minus a threshold value.

The L^2-inspired measure is given by

$$L^2(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\frac{1}{t_{N_t} - t_0} \sum_{i=1}^{N_t} (t_i - t_{i-1}) \left(x_i - y_i\right)^2}, \tag{5}$$

with the array $(t_1, \dots, t_{N_t})$ giving the time steps used in the simulations, and t_0 the time at which the simulation starts. The measure takes values from the positive real line, giving 0 when the arrays are identical. Hence, the higher the value for this measure, the more dissimilar the arrays are.

The relative L^2 norm is given by

$$L^2_{\text{rel}}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\frac{\sum_{i=1}^{N_t}(t_i - t_{i-1})\left(x_i - y_i\right)^2}{\sum_{i=1}^{N_t}(t_i - t_{i-1})x_i^2}}. \tag{6}$$

The last measure we will use is taken from probability theory, where it is used to calculate the distance between two probability distributions. Hence, for this measure we assume positive values for x_i and y_i, $i = 1, \dots, N_t$, in order to avoid running into difficulties. Luckily, this is the case for our application when using arrays of pressure values.

The Jensen–Shannon divergence [6] is given by

$$D_{JS}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}\left(D_{KL}\left(\boldsymbol{x}, \frac{\boldsymbol{x}+\boldsymbol{y}}{2}\right) + D_{KL}\left(\boldsymbol{y}, \frac{\boldsymbol{x}+\boldsymbol{y}}{2}\right)\right) \tag{7}$$

with the Kullback–Leibler divergence [7] defined as

$$D_{KL}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{N_t}\sum_{i=1}^{N_t} x_i \ln\left(\frac{x_i}{y_i}\right). \tag{8}$$

The Jensen–Shannon divergence is mainly used to quantify the difference between probability distributions. Here, however, we will simply use the arrays $\boldsymbol{x}$ and $\boldsymbol{y}$ as input, but we will normalise both arrays. The measure takes values on the interval $[0, \ln(2)]$, with 0 if and only if the arrays are identical [8]. A higher value thus signifies a higher degree of dissimilarity.

Table 1 summarises the five similarity measures.

Table 1
A summary of the five similarity measures used in this paper.

Symbol	Name	Range of values	$\boldsymbol{x}$ identical with $\boldsymbol{y}$
$\rho_{\boldsymbol{xy}}$	Correlation	$[-1, 1]$	1
$\cos_{\boldsymbol{xy}}$	Cosine similarity	$[-1, 1]$	1
$L^2(\boldsymbol{x}, \boldsymbol{y})$	L^2-norm measure	$[0, \infty)$	0
$L^2_{\text{rel}}(\boldsymbol{x}, \boldsymbol{y})$	Relative L^2-norm measure	$[0, \infty)$	0
$D_{JS}(\boldsymbol{x}, \boldsymbol{y})$	Jensen–Shannon divergence	$[0, \ln(2)]$	0

2.3. Choosing threshold values

Having defined the similarity measures in Section 2.2, we need to find a suitable threshold value for each of them. The threshold values should be such that simulation results that diverge too much from the reference simulation are filtered out. At the same time, we do not want to discard results that are accurate enough. In other words we need to balance the two goals in such a way that we limit the number of inaccurate simulations that are accepted, without rejecting too many sufficiently accurate (and possibly quite efficient) simulations. Striking a good balance will greatly affect the outcome of the optimisation procedure.

However, the most suitable threshold values may depend on the reservoir model used for the simulation, and hence we might not have a one-size-fits-all situation. Also, what is considered accurate enough, or which simulation results are diverging too much is a question without an easy, straightforward answer. An experienced engineer who is used to working with such models and simulations might be able to apply expert knowledge to decide whether a specific result is accurate enough and arrive at suitable threshold values to be used. Later on we will provide empirically chosen threshold values which we have found to work well in practice (see Table 2 in Section 4).

2.4. The need for using more than one measure

Suppose we use only one similarity measure to distinguish accurate and inaccurate simulation results. If our chosen measure, at a given threshold, lets through too many solutions which are deemed too inaccurate, we would need to make the threshold stricter. By doing so, we might disregard many solutions which are actually accurate enough, thus losing out on potential improvements in efficiency (in connection with parameter searching to be presented in Section 3). In essence, the similarity measure functions as a filter, and making the threshold stricter will lead to more solutions (accurate and inaccurate) being filtered out.

If we introduce a second similarity measure, we will add a second layer to the filtering mechanism. The idea, then, is that these two layers will filter out different kinds of inaccurate solutions by focusing on different aspects of the solutions. Through this feature, the thresholds can be set slightly less strict in order to accept more of the sufficiently accurate solutions, while still rejecting the inaccurate solutions. Hence, the idea is to introduce several measures with different properties in order to create a more effective filter.

As a concrete example, we can show a case in which only one measure (in this case Jensen–Shannon) deems the simulation result as too inaccurate, while the others stay within their respective thresholds when using the threshold values in Table 2. Fig. 1 shows the WBHP for one well (named D-1H) from the open Norne black-oil model [9], for which only the Jensen–Shannon divergence indicates too much inaccuracy. The largest deviations in pressure are around 15–20 bar, which are higher than what would normally be accepted. Hence, it makes sense to define this new simulation result as being too inaccurate. (We remark that a difference in the time steps used for the two simulation runs is not the reason for these pressure deviations.) In case the Jensen–Shannon divergence was the only similarity measure used, then a stricter threshold would have been chosen to filter out inaccurate simulations that are more easily detected by the other measures, thus risking rejection of many accurate enough simulations as the consequence.

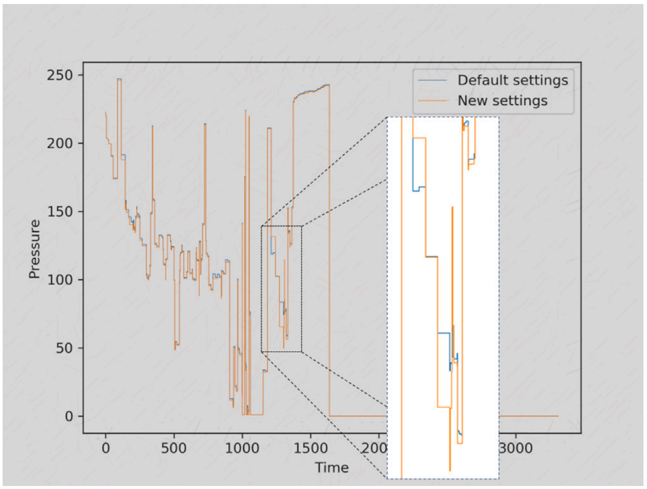


Fig. 1. WBHP for two simulation runs — one being the reference simulation result. The WBHP is from the well named D-1H from the Norne black-oil model [9]. The largest deviations are highlighted in the white rectangle, showing pressure deviations of around 15–20 bar.

3. Automated searches in the parameter space

Our objective is to tune the values of a selected set of numerical parameters of a reservoir simulator, in order to improve (i.e. decrease) the computing time needed to complete the simulation. While doing this, we also want to ensure that the simulation result does not become too different from a reference simulation, guarded by the similarity measures as described in Section 2. An automated search algorithm will be used to find better values for the parameters — i.e. to find values which result in faster simulations. This algorithm works by perturbing the current best parameter set several times, and then checking whether any improvement has been made. Each such step of performing several perturbations of the parameter set and combining them to form an improved set of parameter values amounts to one iteration of the search algorithm.

Before providing a detailed mathematical description of the way in which the parameter search is performed, a brief overview will be given: We start out with a set of parameters (the choice of which will be discussed later), each having an initial value (which might be the default value, or some other value deemed suitable). We then perform an iterative search for more efficient parameters. In each iteration, a certain number of the parameters are perturbed based on the best set of parameter values from the previous iteration. The performance of these new sets of parameters is then gauged by running the simulator with these values — one simulation per new parameter set. Having done so, one further set of parameters is created based on the information available from these simulations and their performance. The preferable set of parameters is then chosen to be the set of parameters which has the best performance. Note that the set of parameters from the previous iteration step might prove to be better than all the new sets of parameters, and if so, we keep this as our current best set. Note also that if a simulation fails (i.e. the simulation does not run to completion), or the solution from a successful simulation is deemed too inaccurate, we discard it.

3.1. Mathematical description

Suppose we have selected for a simulator a set of n parameters with values $\boldsymbol{p} = (p_1, \ldots, p_n)$. The values p_i can be integers or real numbers, and be with or without minimum and maximum values. Let $\boldsymbol{P}$ be the space of all possible values of $\boldsymbol{p}$. We then create a mapping $\mu : \boldsymbol{P} \to \mathbb{R}^+ \cup \infty$ which takes the parameter values and gives out a positive real number (or positive infinity). The mapping μ should be such that a set of parameter values giving a more efficient simulation receives a lower value. The goal, then, is to minimise μ over all $\boldsymbol{p} \in \boldsymbol{P}$. That is, we want to find

$$\boldsymbol{p}^* = \underset{\boldsymbol{p}\in\boldsymbol{P}}{\arg\min}\, \mu(\boldsymbol{p}). \tag{9}$$

Hence, the mapping μ is the *objective function* of our optimisation algorithm. Note that if the simulation does not run to completion for a parameter set $\boldsymbol{q}$, or if the simulation result is deemed too inaccurate (by the similarity measures), we define $\mu(\boldsymbol{q}) = \infty$.

Ideally, as Eq. (9) states, we want to find $\boldsymbol{p}^*$ such that $\mu(\boldsymbol{p}^*) \le \mu(\boldsymbol{p})\,\forall \boldsymbol{p} \in P$. However, finding such an optimal $\boldsymbol{p}^*$, even if one does exist, is practically infeasible. Hence we limit our iterative search to finding an improved parameter set. Starting out with an array of initial values $\boldsymbol{p}^{(0)}$, we randomly perturb the parameters in order to find a better set of values, and we do this iteratively until a certain criterion is met. For each iteration, we get a new set of values $\boldsymbol{p}^{(j)}$ (possibly equal to that from the previous iteration) satisfying $\mu(\boldsymbol{p}^{(j)}) \le \mu(\boldsymbol{p}^{(j-1)})$. To get a more concise notation, we write $\mu^j := \mu(\boldsymbol{p}^j)$.

Each iteration, from $j-1$ to j, is performed using the following steps:

1. We create in total m new parameter sets $\boldsymbol{p}^{(j)_1}, \ldots, \boldsymbol{p}^{(j)_m}$ by perturbing $\boldsymbol{p}^{(j-1)}$ using the following formula:
$$\boldsymbol{p}^{(j)_k} = \boldsymbol{p}^{(j-1)} + \boldsymbol{b}_k \cdot \boldsymbol{u}_k \cdot \boldsymbol{p}^{(j-1)} \tag{10}$$
for $k = 1, \ldots, m$. The perturbation array $\boldsymbol{u}_k = (u_{k1}, \ldots, u_{kn})$ is such that $u_{ki} \sim \mathcal{D}$ for $i = 1, \ldots, n$, where $\mathcal{D}$ is some probability distribution (see Section 4.1.1 for the choice of $\mathcal{D}$ in this work), and $\boldsymbol{b}_k = (b_{k1}, \ldots, b_{kn})$ is such that $b_{ki} = 1$ with a pre-specified probability and $b_{ki} = 0$ otherwise. The latter array, $\boldsymbol{b}_k$, is included to make sure that not all parameter values are perturbed simultaneously.
2. For each new parameter set $\boldsymbol{p}^{(j)_k}$, we ensure that each element $p_i^{(j)_k}$ is within the bounds of that parameter. That is, if $p_i^{(j)_k}$ is higher/lower than the maximum/minimum value, we simply set $p_i^{(j)_k}$ to the maximum/minimum value. Also, we round off values that should be integers to the nearest whole number.
3. For each new parameter set, the value $\mu_k^j := \mu(\boldsymbol{p}^{(j)_k})$ is found by running the simulation and evaluating the objective function.
4. To potentially speed up the optimisation process, we use the μ_k^j values for the newly created parameter sets to generate yet another parameter set $\boldsymbol{p}^{(j)_{m+1}}$. We do this by first calculating a help array $\boldsymbol{s}$ that mimics a gradient-based search direction:
$$\boldsymbol{s} = \frac{\sum_{k=1}^{m}\left[\mu_k^j < \infty\right]\left(\mu^{j-1} - \mu_k^j\right)\left(\boldsymbol{p}^{(j)_k} - \boldsymbol{p}^{(j-1)}\right)}{\sum_{k=1}^{m}\left[\mu_k^j < \infty\right]}, \tag{11}$$
with $[P]$ being the Iverson Bracket [10], giving 1 if P is true, and 0 if P is false. We then calculate
$$p_i^{(j)_{m+1}} = p_i^{(j-1)} + \frac{s_i}{p_i^{(j-1)} \cdot \mu^{j-1}}, \tag{12}$$
with $(s_1, \ldots, s_n)$ being the entries of $\boldsymbol{s}$. If $p_i^{(j-1)} = 0$, we instead use the formula
$$p_i^{(j)_{m+1}} = \frac{s_i}{\mu^{j-1}}. \tag{13}$$
5. Using the newly created parameter set $\boldsymbol{p}^{(j)_{m+1}}$, we run the simulation to find the value $\mu_{m+1}^j := \mu(\boldsymbol{p}^{(j)_{m+1}})$.
6. The resulting new parameter set for iteration j, $\boldsymbol{p}^{(j)}$, is then chosen such that
$$\mu^j = \min\left(\mu^{j-1}, \mu_1^j, \ldots, \mu_m^j, \mu_{m+1}^j\right) \tag{14}$$
is satisfied.

3.2. Optimisation target

For our application, we want a mapping μ that reflects the total running time of the simulation. In that way, μ will be a proxy for the computational requirements needed to complete the simulation, and minimising μ will therefore indirectly amount to a minimisation of the total running time.

Instead of simply adopting the simulation time usage as the objective function, we choose μ as a linear combination of the Newton and linear iterations needed to complete the simulation. The reason is twofold. First, the time measurement of a simulation may itself contain inaccuracies, especially when the simulator is run on a shared hardware platform. Second, adjusting the weight ratio between Newton and linear iterations can allow optimisation for other scenarios, such as a different preconditioner of the linear solver or another intended computer. In other words, we have

$$\mu(\boldsymbol{p}) = I_N(\boldsymbol{p}) + \alpha I_L(\boldsymbol{p}), \tag{15}$$

with $I_N(\cdot)$ and $I_L(\cdot)$ denoting the iteration count for the Newton and linear iterations respectively, and $\alpha \in \mathbb{R}$. Now, the remaining issue is the choice of a suitable α.

3.3. Choosing the optimisation target parameter

An important part of the optimisation procedure is choosing the value α in Eq. (15). This value will balance the weight given to the Newton iterations relative to that of the linear iterations, and hence it will likely impact the end result. In the extreme case, setting $\alpha = 0$ will force the optimisation to disregard the number of linear iterations, and hence any decrease (however large) in Newton iterations might be offset by an increase in linear iterations, which in turn makes it difficult to ensure that we get a speed-up for the simulation as a whole. Hence, it makes sense to balance the linear and Newton iterations in such a way that the total simulation time is likely to be reduced the most.

One possible approach is to choose α such that it mirrors the difference in computational cost between the Newton and linear iterations. As an example, if one was to switch from a simple, computationally cheap preconditioner to another preconditioner which improves convergence properties and reduces the number of linear iterations needed at each time step, but is more computationally costly, one might want to decrease α to retain a good balancing between the Newton and linear iterations when performing the optimisation.

Another approach might be to set α to be the ratio between the number of Newton and linear iterations in the initial (reference) simulation. When choosing this approach, a shift to a more computationally costly preconditioner with better convergence properties will again result in a change in α, since the number of linear iterations relative to that of Newton iterations will go down.

3.4. Additional settings for the optimisation procedure

In addition to the weighting of Newton iterations against linear iterations, given by α, there are several settings which might influence how well the optimisation procedure will perform. As discussed in Section 2.3, the threshold values for the similarity measures are obviously of great importance. Other potentially important settings are:

- **Number of search iterations:** It seems obvious that the higher this value is, the better the optimisation procedure will perform. While this may be true, there is a significant cost associated with increasing the number of search iterations. Hence, we would like to be able to set this value as low as possible, without loosing too much with respect to efficiency of the best simulation. A suitable number of search iterations might be found by running multiple simulations and looking at when the optimisation seems to tail off — that is, when the diminishing marginal improvement is so low that carrying out more iterations of the search is not likely to cause much improvement.
- **Number of perturbations per search iteration:** As with the number of search iterations, increasing the number of perturbations will result in increased computational requirements. However, by increasing the number of perturbations, we might be able to reduce the number of search iterations and still get the same improvement from our optimisation procedure. On the other hand, increasing the number of perturbations by too much might result in slower improvements when taking the total computational expenditure (that is, the total number of simulations that need to be carried out) into account. Hence, a goal might be to find the number of perturbations that will result in the lowest possible overall computational cost.
- **Probability of perturbing each individual parameter:** This setting is less straightforward as the two previous ones. Intuitively it might seem reasonable that each new perturbation should involve as much change as possible to the current best parameter settings, in order to better explore the parameter space. However, it turns out that limiting the number of parameters that are perturbed simultaneously might improve the optimisation.

The above settings will be empirically studied in Section 4.2.1.

4. Experiments

The purpose of this section is to carry out an empirical study of the automated search algorithm introduced in Section 3. More specifically, we are interested in investigating the following questions:

1. Is the automated search algorithm able to iteratively find better and better parameter sets that lead to improved values of the objective function (15) defined in Section 3.2?
2. Do the improved objective function evaluations (due to reduced Newton and linear iterations) translate to a corresponding reduction in the simulation time?
3. Is it possible to use a small-scale reservoir model to produce improved parameter sets, and then apply these to a different and much larger reservoir model for the same purpose of substantially improving the simulation time?
4. Lastly, are the similarity measures described in Section 2.2 capable of acting as accuracy control when integrated inside the automated search algorithm? How does this mechanism of quality check compare with the standard metrics, such as monitoring the mass balance error within a reasonable bound?

In the following, we will start with a detailed description of the experimental setup in Section 4.1, including the chosen threshold values for the similarity measures (Section 4.1.1), the Flow reservoir simulator and hardware used (Section 4.1.2), and the list of numerical parameters of the reservoir simulator to be tuned by the automated search algorithm (Section 4.1.3). We will then continue with presenting three open reservoir models and one proprietary reservoir model to be studied (Section 4.2). Thereafter, we will present an extensive study of the automated search algorithm by experimenting with one of the open reservoir models (Section 4.3), where both the robustness and effectiveness are carefully investigated. Moreover, we will further test the automated search algorithm on the other two open reservoir models (Section 4.4). In Section 4.5, we will examine the effect of applying improved parameter sets, which are obtained from running the automated search algorithm for the Norne black-oil model, to an industrial-scale proprietary reservoir model. Finally, we will verify in Section 4.6 that the similarity measures have indeed ensured sufficient accuracy of the simulation results, with respect to maintaining the mass balance.

Table 2
The threshold values (for the five similarity measures) used throughout the experimental part of this paper. See Table 1 for more information about each similarity measure.

Name	Threshold value
Correlation	0.995
Cosine similarity	0.995
L^2 measure	10^{-4}
Relative L^2 measure	0.1
Jensen–Shannon divergence	10^{-7}

4.1. *The experimental setup*

4.1.1. *Threshold values for the similarity measures*

As discussed in Section 2.3, we need to decide which threshold values should be used for the similarity measures. Table 2 lists the threshold values used in the remainder of this work. These values have been chosen through a process of trial and error, and seem to work well for all the reservoir models used in this article.

In addition to the threshold values for the similarity measures, there are a few choices and implementation details which should be mentioned:

- Some numerical parameters of a reservoir simulator (as listed in Table 3) have open bounds — e.g. $(0, \infty)$. For these, we introduce a small value ϵ and let the bounds of the parameter be $[\epsilon, \infty)$. Specifically, we have used $\epsilon = 10^{-10}$, which was chosen based on a limited amount of trial and error.
- During the first step of each iteration of the search algorithm described in Section 3.1, the entries of $\boldsymbol{u}_k$ from Eq. (10) are distributed according to the probability distribution $\mathcal{D}$. Specifically, we have used $U[-1, 1]$ for $\mathcal{D}$.
- To keep any single simulation from using a huge amount of time to complete, a cut-off is implemented. This means that any simulation taking more than a prescribed amount of time will be interrupted and considered unsuccessful. The current cut-off time is set to 900 s (i.e. 15 min). This cut-off time works well for the small open-source reservoir models used in this work, because a simulation running for more than 900 s has a tendency to run for much longer, and hence interrupting it will save a lot of time in the optimisation procedure. For larger models, a longer cut-off time may be needed.

4.1.2. *The OPM flow simulator and hardware*

Before delving into the actual experiments and results, however, let us give a quick overview of the reservoir simulator software, and the hardware used in this work. The simulator software used throughout this work is OPM Flow, which is open-source and a part of The Open Porous Media Initiative [1,2].

All the simulations have been run on a server with dual AMD EPYC Milan 7763 64-core CPUs by using 32 MPI processes evenly spread over the two CPUs. The only exception concerns the simulations of the million-cell model described in Section 4.2.4; see Section 4.5 for information about the hardware used for this proprietary reservoir model.

4.1.3. *Numerical parameters of the reservoir simulator*

Which parameters should be targeted when searching for a better set of parameter values? Ideally, we want to choose parameters that have the greatest impact, and ignore those that do not contribute much (or anything) to the efficiency of the simulation. However, some parameters might be interdependent, meaning that changing them simultaneously does not produce the same results as changing them individually. Indeed, selecting the “correct” set of parameters is more of an art, so some trial and error, together with the use of domain expertise, might be needed.

Table 3 lists the parameters chosen for this paper.

4.2. *The reservoir models*

4.2.1. *Norne black-oil model*

The Norne model is a “real field black-oil model for an oil field in the Norwegian Sea” [9]. It comes with an open license (the Open Database License [11]), and has been widely used by many researchers for validation and benchmarking. We will extensively experiment with the automated search algorithm using this model in Section 4.3. The reservoir model has in total 44 431 active cells and a total pore volume of 673 248 728 m^3.

4.2.2. *Norne prediction model*

The Norne prediction model [12] uses the same grid as the Norne black-oil model introduced above. However, this so-called prediction reservoir model does not consider a historic period, but is rather used to simulate a future production period. This prediction model is different from its historic counterpart, in that WBHP values are fixed whereas the production rates become the solution variables. The model is open-source and can be found at the Github repository of OPM [1] under the name `NORNE_ATW2013_4A_STDW.DATA`.

4.2.3. *Smeaheia*

Smeaheia [13] is an open-source CO_2 storage model of a site located in the Norwegian North Sea. The major difference between this reservoir model and the two Norne models is that the latter two simulate three-phase flows in porous media, whereas the Smeaheia model simulates two-phase flows.

4.2.4. *Million-cell North Sea oil field model*

We also consider a reservoir model containing approximately one million active cells. Similar to the Norne models, this much larger model is a realistic representation of a North Sea oil field with a black-oil fluid system and more than one hundred production and injection wells. This is a proprietary reservoir model.

4.3. *Experimenting with the Norne black-oil model*

In this subsection, we will perform experiments with the automated search algorithm using the Norne model (see Section 4.2.1). Firstly, we will test whether the gradient-inspired step offers an improvement, and which values we should choose for the number of perturbations per search iteration, as well as the probability of perturbing each individual numerical parameter. Note that our final choices here are listed later in Table 4. Then, we will investigate how to best choose the value α from the optimisation target given in Eq. (15).

4.3.1. *The effect of a gradient-inspired search*

Before looking into the settings of the optimisation procedure, it is interesting to check whether the inclusion of the gradient-inspired step from Eqs. (11)–(13) in the search process is in fact fruitful. By removing this step and instead adding one extra perturbation per search iteration (to make sure that the same number of parameter perturbations are tested), and comparing this with the original optimisation, we can see whether there is anything to be gained from including the gradient-inspired step. Fig. 2 shows a summary of 40 independent optimisations with and without a gradient-inspired step. By independent optimisations, we simply mean that each optimisation is run separately with the same initial settings. The stochastic nature of the optimisation procedure results in slightly different optimisation results for each run of the optimisation algorithm. Hence, running several independent optimisations will allow us to get a better picture of the improvements one can expect.

Table 3
The chosen numerical parameters of the OPM Flow simulator [1] targeted by the automated search algorithm introduced in Section 3.

Name	Type	Default	Bounds	Explanation
linear-solver-max-iter	$\mathbb{Z}$	200	$[1, \infty)$	The maximum number of iterations for the linear solver.
max-strict-iter	$\mathbb{Z}$	0	$[0, \infty)$	The maximum number of Newton iterations before relaxed tolerances are used for the CNV convergence criterion. See tolerance-cnv(-relaxed).
flow-newton-max-iterations	$\mathbb{Z}$	20	$[1, \infty)$	The maximum number of Newton iterations per time step used by the simulator.
max-welleq-iter	$\mathbb{Z}$	30	$[1, \infty)$	The maximum number of iterations to determine the solution to the well equations.
newton-max-relax	$\mathbb{R}$	0.5	$[0, 1]$	The maximum relaxation factor of a Newton iteration used by the simulator.
linear-solver-reduction	$\mathbb{R}$	0.01	$(-\infty, 1]$	The tolerance for the linear solver. The linear solver convergences when the residual is reduced sufficiently.
relaxed-max-pv-fraction	$\mathbb{R}$	0.03	$[0, 1]$	The fraction of the pore volume of the reservoir where the volumetric error (CNV) may be violated during strict Newton iterations.
tolerance-cnv	$\mathbb{R}$	0.01	$(0, \infty)$	The maximum non-linear tolerance error. This is the local convergence tolerance (maximum of local saturation errors).
tolerance-cnv-relaxed	$\mathbb{R}$	1	$(0, \infty)$	The relaxed local convergence tolerance that applies for iterations after the iterations with the strict tolerance.
tolerance-mb	$\mathbb{R}$	10^{-6}	$(0, \infty)$	The maximum mass balance error, that is the tolerated mass balance error relative to total mass present.
tolerance-well-control	$\mathbb{R}$	10^{-7}	$(0, \infty)$	The maximum tolerance for the well control equations.
tolerance-wells	$\mathbb{R}$	0.0001	$(0, \infty)$	The maximum non-linear error for the well equations.

Fig. 3 shows a scatter plot of the percentage improvement in the optimisation target for each search iteration, again with 40 independent optimisations. The red dots are improvements where the gradient-inspired suggestion was the best, whereas the blue dots are the improvements where one of the perturbed parameter arrays was better.

From these figures, it is clear that it is beneficial to include a gradient-inspired step in the search algorithm. However, the benefit seems to come from rapid improvement in the first few stages of the optimisation. After this, the gradient-inspired step does not add much to the performance. As such, including a gradient-inspired step seems to allow us to run fewer search iterations to reach the same level of improvement. Also, one might imagine a modified algorithm in which the gradient-inspired step is removed after a certain number of iterations or a certain percentage improvement in the optimisation goal, in order for the algorithm to focus more exclusively on fine-grained optimisation in the vicinity of the current best solution.

4.3.2. *Number of perturbations per search iteration*

How many perturbations per search iteration – see Eq. (10) – will yield the best results? While this might depend on the model, the numerical solver, the number of processes and possibly other factors, we might be able to find some values which work reasonably well. More concretely, the goal will be to see whether we can identify lower and upper bounds at which the performance seems to drop, and through this get a sense of which values should work best.

In order to make a fair comparison, we should make sure that the total number of simulations stays constant — that is, if we increase the number of perturbations, we should at the same time decrease the number of search iterations in such a way that the number of perturbations (plus one gradient-inspired parameter configuration) times the number of search iterations remains constant.

Fig. 4 shows the performance when creating 2, 5, 8 and 11 perturbations per search iteration. Based on these plots, we can conclude that all four generally perform well. However, perturbing less than 5 or as much as 11 times seem to reduce the quality of the optimisation, and we would therefore expect that perturbing between 5 and 8 times will yield the best results. In the remainder of this article, we will use 5 perturbations.

4.3.3. *Probability of perturbing the parameters*

Up until now we have only perturbed about half of the parameters each time we created a new parameter array, because early experiments seemed to suggest that this was a good idea. In other words, the array $\boldsymbol{b}_k$ from Eq. (10) has been such that each index b_{ki} has about a fifty-fifty chance of being either 0 or 1. It is not immediately clear why perturbing many or few parameters at the same time yields worse performance than perturbing around half the parameters, but experiments do seem to suggest such behaviour. One possible reason why perturbing few parameters might not be a good idea, is the lack of progress that ensues, whereas perturbing too many parameters at the same time might run a higher risk of producing parameter settings which lead to inaccurate (and hence non-acceptable) simulation results, due to the possibility of making larger jumps in the parameter space.

Here, however, we present some empirical evidence to back up our claim that perturbing about half the parameters each time is a good idea. Fig. 5 shows the optimisation results when using different values for the perturbation probability. The overall picture seems to be that all values can be expected to result in a substantial improvement in performance. However, a closer investigation reveals that a probability closer to 0.5 than to the more extreme values of 1 and 0.1 is likely to give a sharper increase in performance during the first stages of the optimisation procedure. A more thorough experimentation with perturbation probabilities centred around 0.5 shows that 0.6 performs well — indeed slightly better than nearby values. In the remainder of this article, we will therefore use 0.6 as the probability of perturbing each parameter value.

Table 4
Summary of the choices made for the optimisation algorithm, based on the experiments discussed in Sections 4.3.1–4.3.3.

Algorithmic parameters	Choice
Use gradient-inspired step	Yes
Number of perturbations	5
Probability of perturbing	0.6

Table 4 summarises the choices made for the remainder of this work.

4.3.4. *Changing the weighting inside the objective function*

After having confirmed that a gradient-inspired step is beneficial and settled on the appropriate values for the number of perturbations per search iteration and the probability of perturbing each individual parameter; the next topic worth investigating in-depth is the weighting α inside the linear combination which acts as the optimisation

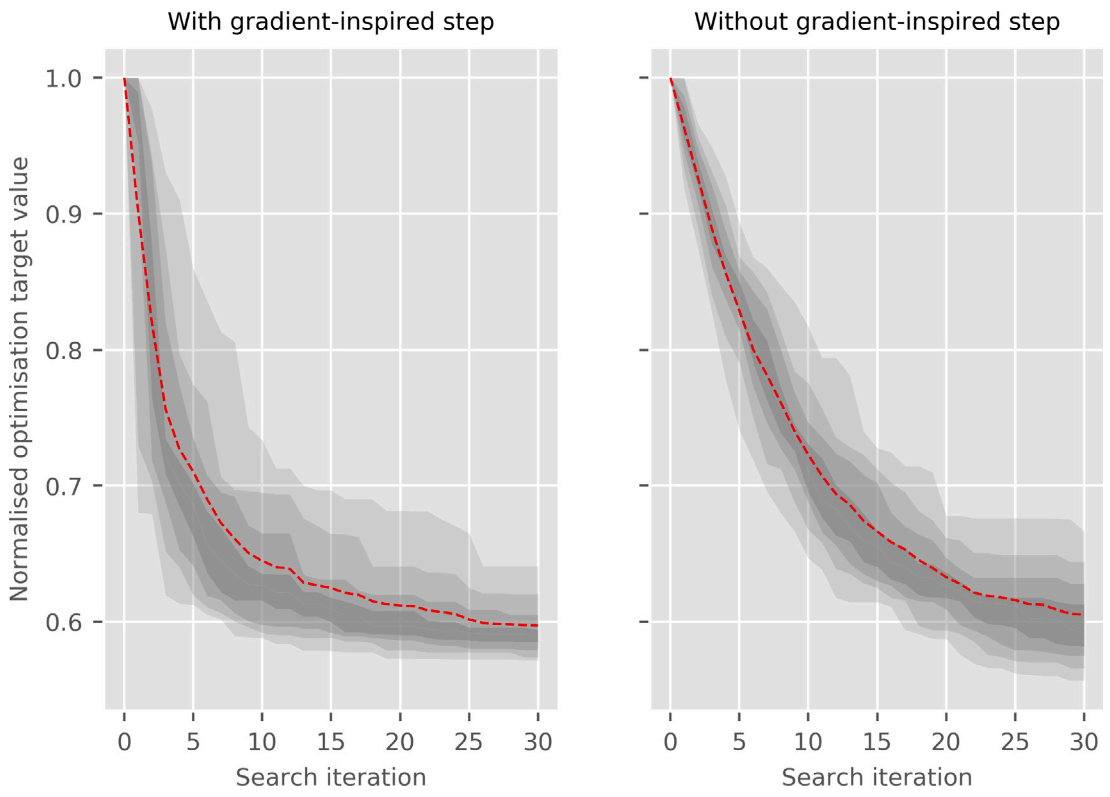


Fig. 2. A comparison between optimising the numerical parameters *with* the gradient-inspired step from Eqs. (11)–(13) and a naïve search *without* the gradient-inspired step. Each plot summarises 40 independent optimisation runs, with the red dashed line giving the mean improvement, and the different shades of grey giving the percentiles from the median and to the 10th/90th percentile (with lighter grey being farther from the median).

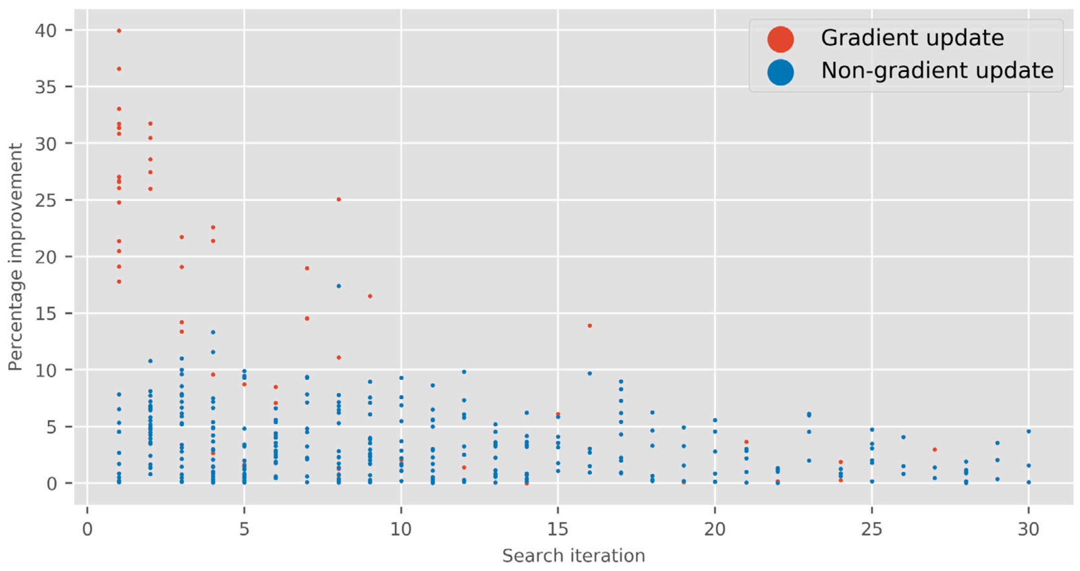


Fig. 3. Percent improvements of the objective function during the iterations (showing in total 40 independent optimisations). The blue dots indicate that the improvement came from one of the randomly perturbed parameters, whereas a red dot means the gradient-inspired step provided the improvement.

goal of our algorithm, given by Eq. (15) and discussed thoroughly in Section 3.3.

Fig. 6 shows the results from using $\alpha = 0.0620$, which is the ratio between the number of Newton and linear iterations used to complete the simulation when default (initial) parameter values are used. While there is no readily available explanation why such an approach to weighting the iterations should work, experiments seem to suggest that it is actually a fairly good way to set the value of α.

Another way to find a good value for α is by performing linear regression using several different simulation results (with time measurements, and actual counts of linear and Newton iterations). By running several simulations with different parameter settings, as well

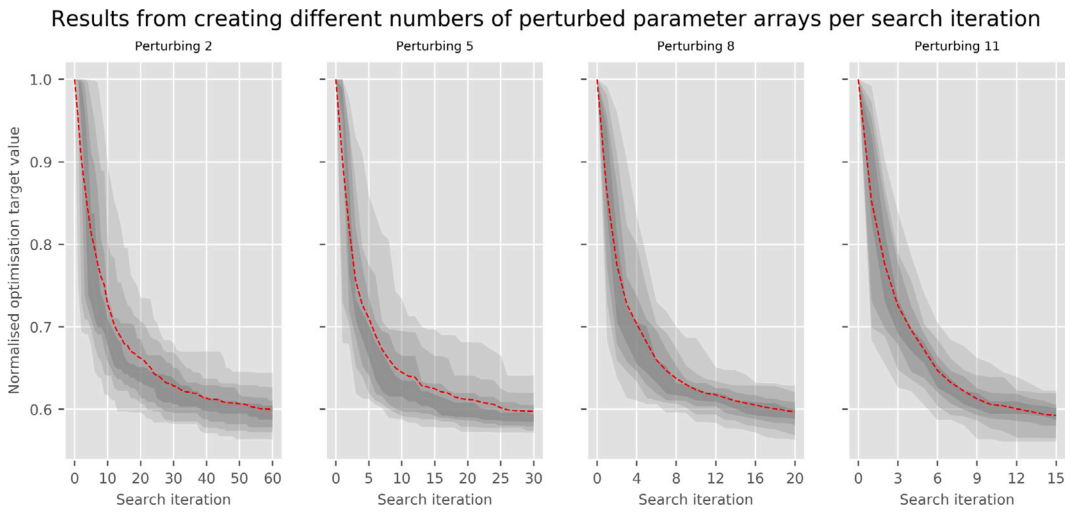


Fig. 4. Plots showing the results from optimising the numerical parameters when using 2, 5, 8 and 11 perturbations per search iteration. Each plot summarises 40 independent optimisation runs, with the red dashed line giving the mean improvement, and the different shades of grey giving the percentiles from the median and to the 10th/90th percentile (with lighter grey being farther from the median).

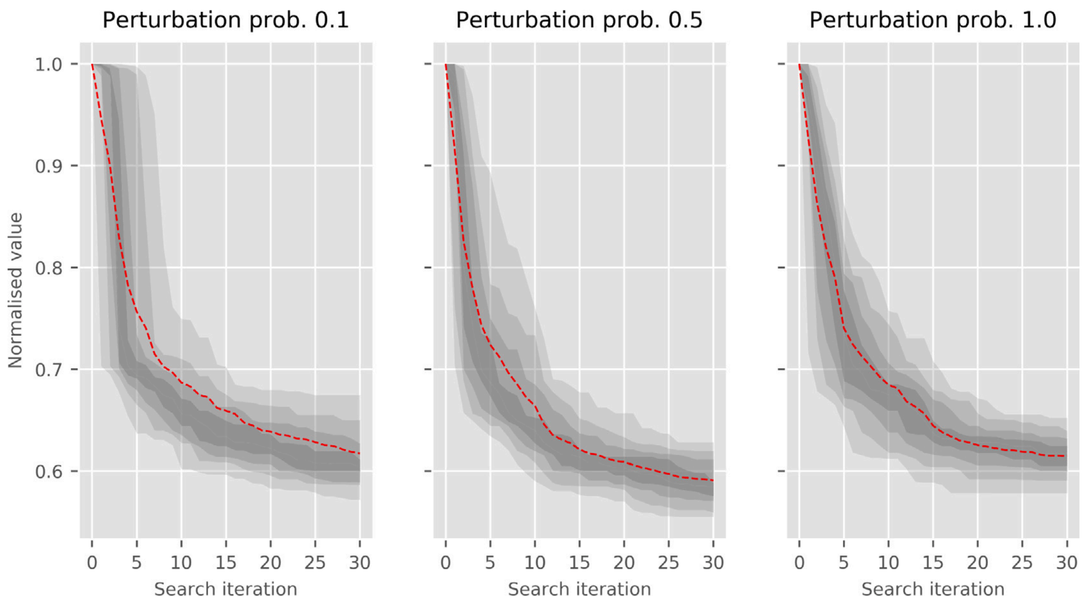


Fig. 5. Plots showing the results from optimising the numerical parameters when using 0.1, 0.5 and 1.0 as the probability of perturbing each individual parameter value. Each plot summarises 40 independent optimisation runs, with the red dashed line giving the mean improvement, and the different shades of grey giving the percentiles from the median and to the 10th/90th percentile (with lighter grey being farther from the median).

as exclusive access to the computing hardware, we can use linear regression to fit the equation

$$T(\boldsymbol{p}) = \alpha_N I_N(\boldsymbol{p}) + \alpha_L I_L(\boldsymbol{p}) + c, \tag{16}$$

that is, to determine the values α_N, α_L and c. The weighting parameter will then be given by the ratio $\alpha = \alpha_L/\alpha_N$. By running 20 simulations with different parameter settings, we get an estimate for α equal to 0.0335. The results when using this α value in the optimisation algorithm are visually similar to those seen in Fig. 6.

It is not clear from the optimisation results alone which α value performs better. In the end, we are not really concerned about the actual number of Newton/linear iterations, but rather the time needed to complete the simulation. Hence, in order to fully compare the two α values, we should estimate the reduction in time that each of these values produce.

4.3.5. *Impact on the simulation time*

In order to measure the time for each simulation, we run 10 independent simulations with exclusive access to the computing hardware and measure the time taken for each of them to complete. Then, we take the minimum of these as our time measurement.

To see how each value of α performs, we take the results we got from running 40 independent optimisations, and measure the time of each parameter setting. While this requires quite a lot of computing time, it gives us an insight into the expected performance of each choice of α. Due to the stochastic nature of the optimisation algorithm, any single optimisation might not paint a sufficiently accurate picture of the expected improvement in performance.

Fig. 7 shows the mean computing times for 40 independent parameter optimisations for α equal to 0.00625, 0.0335, 0.0620, 0.2025, and 0.330. As can be seen, all choices of α is likely to yield significant improvements in computing time. However, a very low α (in this case

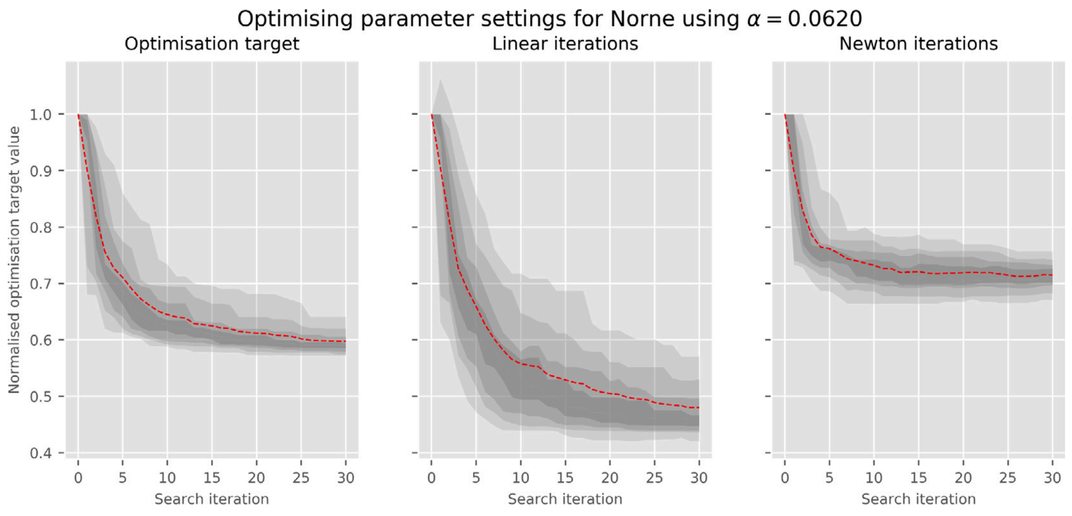


Fig. 6. Plots showing the results – the optimisation target (i.e. Eq. (15) giving the linear combination of iterations), the linear iterations and the Newton iterations – from optimising the numerical parameters when using $\alpha = 0.0620$ in Eq. (15). Each plot summarises 40 independent optimisation runs, with the red dashed line giving the mean improvement, and the different shades of grey giving the percentiles from the median and to the 10th/90th percentile (with lighter grey being farther from the median).

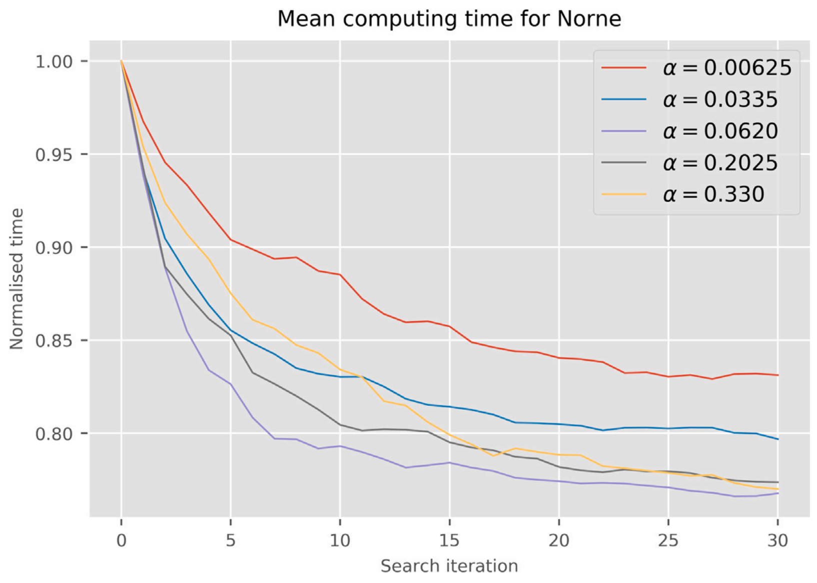


Fig. 7. The mean computing time when running the optimisation algorithm on Norne using the ILU0 (default) preconditioner. The data is based on 40 independent optimisation runs for each α value, and then 10 independent time measurements for each parameter setting, the minimum of these giving the time measurement for that set of parameters.

0.00625), is clearly performing worse than the higher values, indicating that placing too much weight on the newton iterations (which in turn gives less decrease in linear iterations) is not beneficial. Also, the opposite seems to be true, at least for search iterations 1 to 15: Placing too much weight on the linear iterations, as is the case with $\alpha = 0.330$ results in worse improvement in computing time than lower values that give more weight to the Newton iterations.

From these observations, we can hypothesise that an optimal value for α is likely to lie somewhere between 0.00625 and 0.330. Note that these exact α values are dependent upon the use of the Norne model — any other model is likely to require different α values for optimal reduction in the simulation time.

4.3.6. Running norne with CPR preconditioner

The OPM Flow simulator uses incomplete LU factorisation (ILU0) as the default preconditioner for the linear system solver. Now, we want to test the robustness of the automated search algorithm by adopting the more advanced CPR preconditioner. By changing the preconditioner to a more costly one, the most important effect will be a reduction in the number of linear iterations. By carrying out the parameter optimisation procedure in these settings, can we still get similar reductions in iteration counts and computing time?

Fig. 8 shows the mean computing time when optimising parameters for Norne with a CPR preconditioner, running 40 independent optimisations, clearly showing that the optimisation procedure also works well for this preconditioner.

4.3.7. Using different numbers of MPI processes

In order to ascertain the stability of the optimisation procedure, additional tests were performed using 4, 8 and 16 MPI processes. Fig. 9 shows the results from running 40 individual optimisations with the settings given in Table 4. The results suggest that the number of MPI processes does not play a crucial role with regards to the end result of the optimisation. In fact, using fewer MPI processes seems to give a larger improvement in the optimisation target. It should, however, be noted that using fewer MPI processes will in general result in the optimisation procedure taking more time to complete.

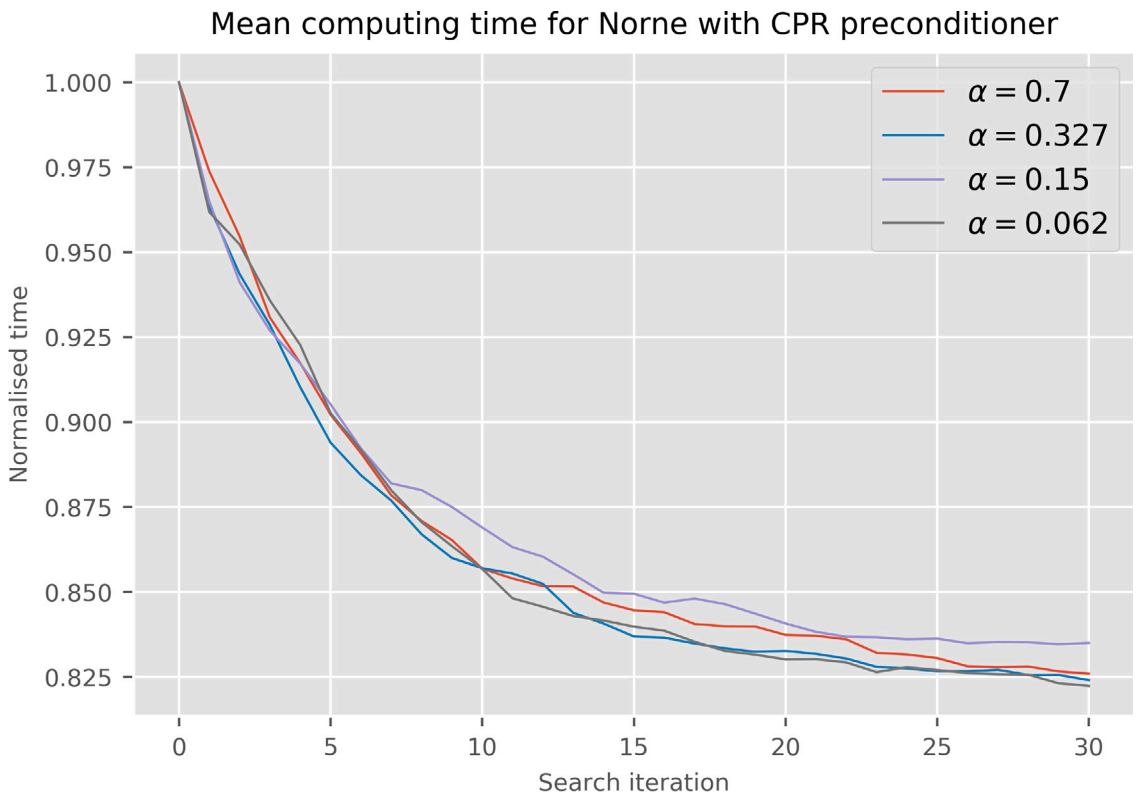


Fig. 8. The mean computing time when running the optimisation algorithm on Norne using the CPR preconditioner. The data is based on 40 independent optimisation runs for each α value, and then 10 independent time measurements for each new parameter setting, the minimum of these giving the time measurement for that set of parameters.

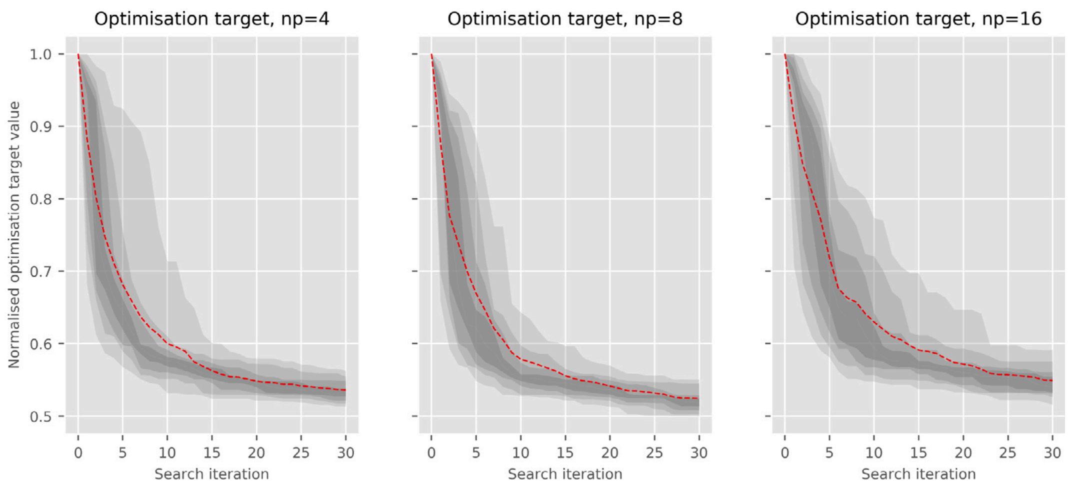


Fig. 9. The results from optimising the numerical parameters when using 4, 8 and 16 MPI processes (instead of 32, see Fig. 6). Each plot summarises 40 independent optimisation runs, with the red dashed line giving the mean improvement, and the different shades of grey giving the percentiles from the median and to the 10th/90th percentile (with lighter grey being farther from the median).

4.4. Experimenting with the other two open reservoir models

4.4.1. Running the norne prediction model

By running a completely different model (albeit on the same grid), we can get a better picture of how general is the optimisation procedure. For the prediction model, we use production rates to gauge the accuracy of new simulations, instead of the WBHP. Otherwise, the algorithm works exactly the same.

Fig. 10 shows the mean improvement in time from running 40 independent parameter optimisations, clearly displaying that the procedure works well also for the prediction model.

4.4.2. Running the Smeaheia CO_2 model

So far, all models considered here have been black-oil models with three phases. It is thus interesting to see whether a two-phase model

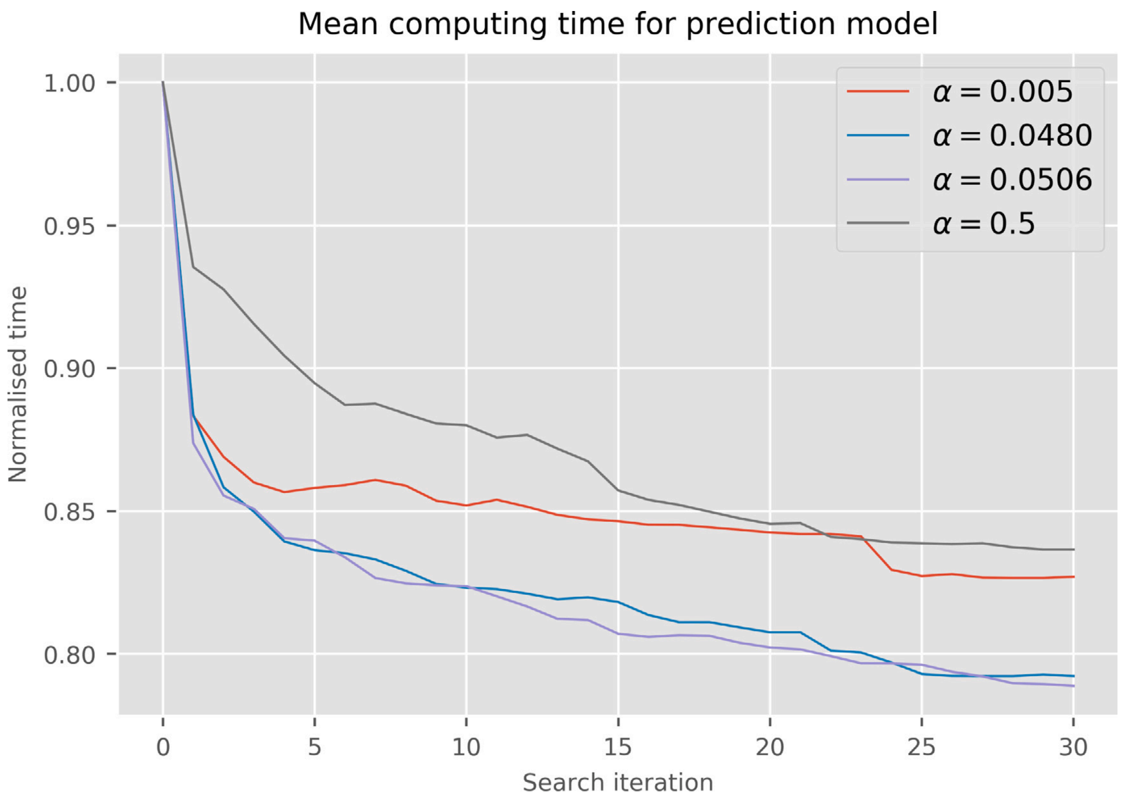


Fig. 10. The mean computing time when running the optimisation algorithm on the prediction model (see Section 4.2.2). The data is based on 40 independent optimisation runs for each α value, and then 10 independent time measurements for each new parameter setting, the minimum of these giving the time measurement for that set of parameters.

for CO_2 storage could also be input into our optimisation framework to yield fruitful results.

Fig. 11 shows the mean improvement in time from running 40 independent parameter optimisations. Again there is a significant reduction in the computing time.

4.5. Applying improved parameters to a proprietary reservoir model

An interesting question is whether the parameters found by optimising on a small reservoir model such as Norne, can be successfully applied to a different, much larger model to improve the computing time while ensuring that the simulation results remain accurate enough. To investigate this, we will use three sets of improved parameters found for the Norne model in Section 4.2.1 and apply these to the proprietary black-oil reservoir model described in Section 4.2.4.

To get the improved parameter sets for this experiment, we chose 0.161 as the α value in Eq. (15), which was based on the number of linear and Newton iterations used to solve the large black-oil model with default parameter settings.

The experiments on the million-cell reservoir model were conducted on dual ARM 64-core Kunpeng 920-6426 CPUs. All simulations of the model used 128 MPI processes. We considered four sets of parameters including the default "conservative" parameter set. The total simulation time and numbers of linear iterations and Newton iterations required to complete the simulations for the four parameter sets are displayed in Table 5.

The iteration counts and execution times displayed in Table 5 demonstrate that the improved parameter sets attained using simulations of the Norne model also yield improved performance for the much larger black-oil model, although the improvements with the million-cell model are not as smooth as those with the Norne model. For all the four parameter sets, the accumulated mass balance error for the million-cell model (see Section 4.6) remains below 0.1%.

Table 5
A summary of the simulation results from running the million-cell North Sea oil field model (see Section 4.2.4) using three improved parameter sets obtained when optimising Norne using an α value of 0.161. For completeness, the corresponding measurements for the Norne black-oil model are also included.

Model name	Optimisation target	Linear iterations	Newton iterations	Time (s)
Million-cell	3937.2	12374	1945	612.73
	2759.2	5399	1890	485.45
	2957.9	5198	2121	515.61
	2521.1	3808	1908	458.56
Norne	4993.8	22390	1389	50.60
	3571.9	15055	1148	44.17
	3217.4	13369	1065	42.02
	3013.4	12251	1041	40.52

4.6. A look at the mass balance error

Instead of looking at the WBHP for each well in the model, previous work has used the mass balance equation (MBE) error to ascertain the quality of the simulation results [14,15]. Hence, it is appropriate that we check that the improved simulation results found in this article has an MBE error that stays close to zero throughout the simulation (i.e. for each time step). We here look at the accumulated MBE error for each time step, and for oil, water and gas separately. The MBE error is calculated by (for each time step) subtracting from the initial mass in place the current mass in place, and adjusting this by the total amount of mass injected and extracted up to the given point in time. The current mass in place is found by summing up the mass in place for each region in the domain. Thus, what we end up working with is the MBE error for the whole domain — not for each individual cell. This is on a par with the approach taken in previous works.

Fig. 12 shows a comparison between the default simulation and an optimised simulation result, where the accumulated MBE errors for oil, water and gas are plotted for each time step. Although the MBE errors

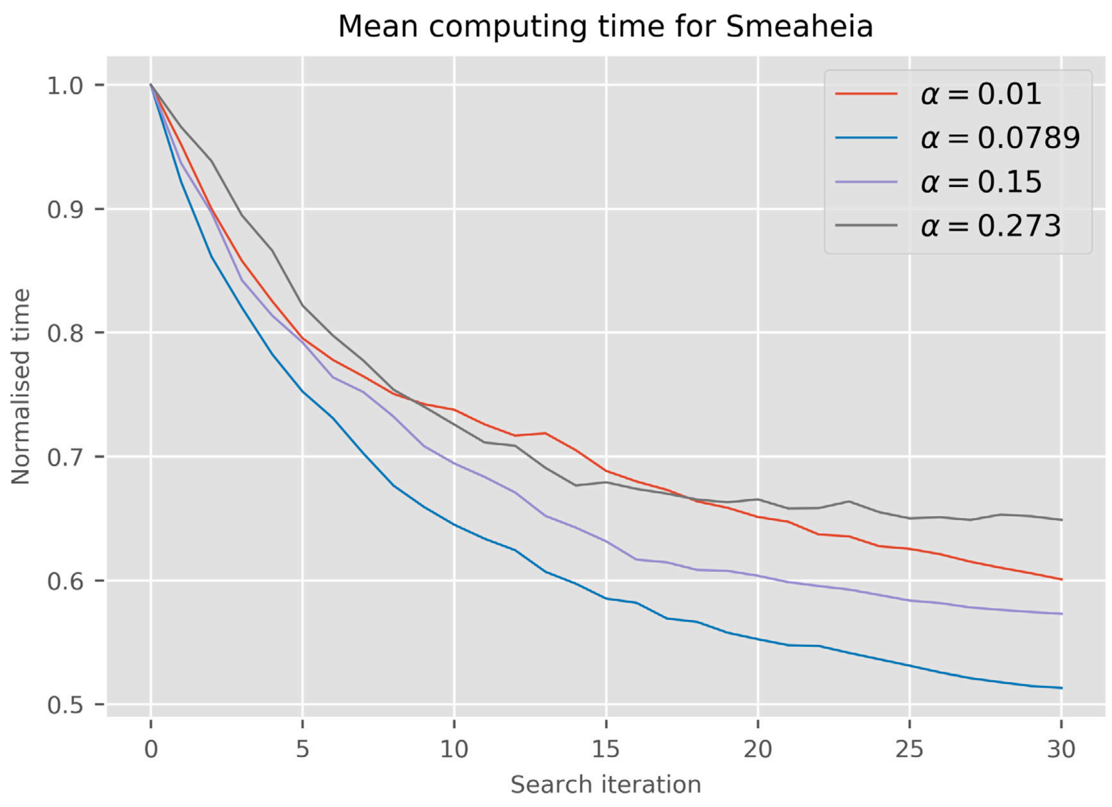


Fig. 11. The mean computing time when running the optimisation algorithm on the Smeaheia model (see Section 4.2.3). The data is based on 40 independent optimisation runs for each α value, and then 10 independent time measurements for each new parameter setting, the minimum of these giving the time measurement for that set of parameters.

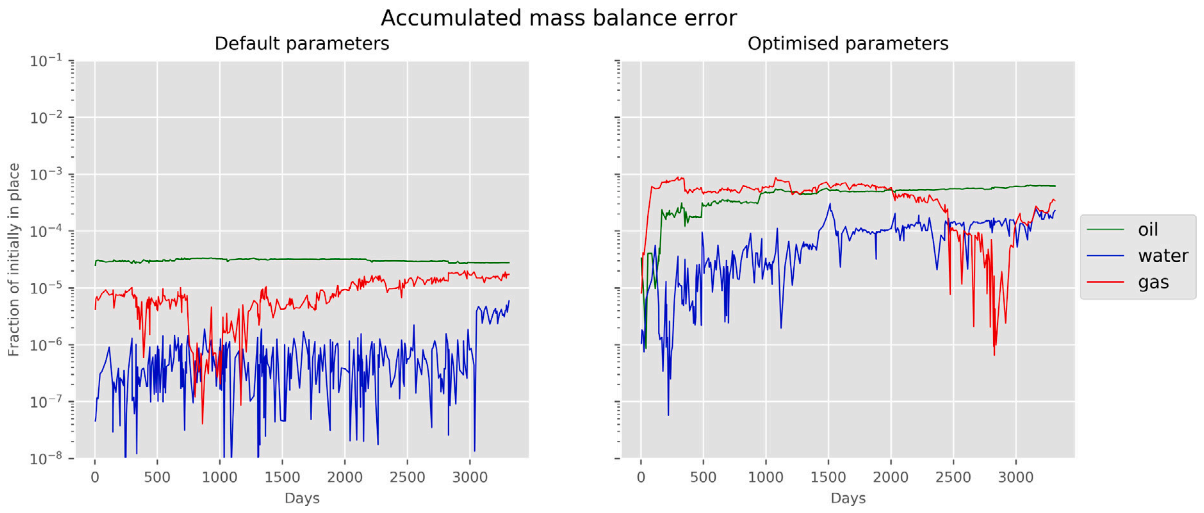


Fig. 12. The accumulated mass balance equation (MBE) error for oil, water and gas for the simulation of Norne [9] when using the default parameter settings (to the left) and when using an optimised set of parameters (to the right).

are clearly quite a lot larger for the optimised simulation, they are still less than 0.1% for each fluid and all time steps. A test of four other randomly chosen optimised parameter settings indicates that the results in Fig. 12 are indeed quite general, with the maximum MBE error for any time step being just over 0.2%. These results are similar to or better than the results from previous works.

5. Related work

Due to the nature of the work described in the earlier sections, the following section is divided into three subsections covering first the general subject of parameter optimisation, then the use of similarity measures (especially in the context of time series data), and finally a more concrete discussion concerning the optimisation of reservoir simulations (including both parameter optimisation and accuracy control).

5.1. Parameter optimisation

Parameter optimisation is a general problem found in a multitude of different research areas. In recent years, optimising hyperparameters for machine learning algorithms has received much attention [16, 17], but applications also include flight scheduling simulations software [18], quantification of gene expression in bioinformatics [19], solving combinatorial optimisation problems [20], and optimisation of chemical processes [21]. While these applications are indeed very different from our reservoir simulator application, the general problem of tuning parameters in order to improve performance with respect to computing time remains the same.

In several applications, the number of parameters and the mathematical properties of the parameter spaces make classical optimisation procedures useless, as they require continuously differentiable spaces. Furthermore, the objective function in the optimisation might not be easily computed, as it could entail running a simulation or training a machine learning model — both of which are generally time consuming tasks. Hence, many approaches have been developed for optimisation in higher dimensions which do not include mathematically well-behaved parameter spaces, and which do not necessitate a huge amount of evaluations of the objective function. Indeed, in the work presented in this article both of these points had to be taken into account when designing the optimisation procedure. As a side note, such problems can be categorised as high-dimensional, expensive (computationally), black-box (HEB) problems, and there are several approaches to solving them [22].

Without the availability of a gradient to provide information about directions that might be useful for further exploration, many optimisation methods still try to explore the parameter space in an intelligent manner. A whole range of bio-inspired artificial intelligence algorithms exist to solve this problem [23], ranging from evolutionary algorithms [24] to particle swarm optimisation [25] and ant colony optimisation [26]. Indeed, observing the behaviour of certain animals and designing a *metaheuristic algorithm* based on this has produced numerous optimisation approaches for high-dimensional optimisation problems. More recent examples are whale optimisation [27], grey wolf [28] and horse herd optimisation [29] algorithms.

Generally, metaheuristic algorithms start out with an initial set of points in the parameter space, and then use some method to decide which new points to investigate in the next iteration step. The new points are chosen based on information gained from the previous points during previous iterations. Many of these algorithms have proven useful in practical applications, even though the mathematical rigour of the methods tend to be limited.

Statistical approaches to parameter optimisation are also much used, the most common of which might be Bayesian optimisation [30, 31]. Here, a statistical distribution is placed on the parameter space to guide the search for better parameter values by locating regions for which there is high probability of finding better parameters. Exploration of the parameter space then leads to updates to the statistical distribution, through application of Bayes' formula.

What makes our work challenging is the presence of both a mathematically ill-behaved parameter space and an objective function whose evaluations require a lot of time. This has motivated us to develop an optimisation algorithm which can deliver useful improvements without too many steps (and hence objective function evaluations), or to make use of a gradient to determine which direction to take in the parameter space.

5.2. Similarity measures

Measuring the similarity between time series is a problem which finds applications in a vast range of research areas, and hence there exists lots of research on the topic [32]. The time series similarity measures used in this article can be categorised as lock-step measures, in that they use the time series data directly in a mathematical equation, and the time series are at the exact same temporal locations. By extracting some information from the time series (e.g. by calculating the Fourier coefficients) and using this in the lock-step formulas, feature-based measures can be obtained. A further approach is to use time series models like the auto-regressive model. Here, the idea is to fit a model to the time series data, and then use the resulting parameters to calculate a distance between the data. Such approaches are called model-based measures [33]. Lastly, there are elastic measures, such as dynamic time warping (DTW), which work by finding the cost of aligning the time series in the temporal domain [34].

Much, if not most, research into similarity measures for time series focus on classification and clustering, and is not necessarily much concerned the exact distance between two time series. In this work, although the scale of the similarity measures differs greatly, we use the distances to determine suitable threshold values. That is, we are not interested in deciding whether a new time series falls into one of several categories, but rather whether it is sufficiently close to a given (reference) time series.

Of the time series used in this work, both the L^2 norm and the correlation coefficient have been used in the literature on measuring the similarity between time series [35–37]. Further, some previous work has used the cosine similarity as the basis of a comparison between time series data [38]. An important distinction between their use of this measure and the one used in this article, is that their version is calculated by creating a new (adjusted) time series of the differences between consecutive time steps, instead of using the original time series and calculating the cosine similarity directly, using Eq. (4).

The use of Jensen–Shannon divergence to evaluate the distance between two time series has also been investigated to some degree [39, 40]. Originally, Jensen–Shannon divergence is used to quantify the distance between two probability distributions. When using it to measure the distance between two time series, one approach would be to first create probability distributions from the time series data. By mapping each entry in the time series to one of several categories (or symbols), one can create a probability distribution by finding the frequency of appearance of each symbol [39]. In this work we have opted for the simpler approach of using the time series directly in the formula for the Jensen–Shannon divergence.

As stated in Section 2.1, our work only deals with similarity measures where the time series are composed of the exact same time steps. However, there do exist some similarity measures for which this is not required [41].

5.3. Optimising reservoir simulations

When optimising the numerical algorithm for simulations, there are two main approaches available. One could change the algorithm and then run the simulation (i.e. *offline* optimisation), or one could run the simulation and change the algorithm between certain time steps or iterations (i.e. *online* optimisation). In this article, we have focused on the former option. However, some work has also been done for the latter option in the context of reservoir simulations [42,43].

There seems to be little previous work done on offline optimisation of reservoir simulations. However, some promising work has been published quite recently. Indeed, initial work suggested that performing some kind of optimisation of the numerical parameters will in some cases result in improved performance and be worth the extra time needed to perform the optimisation, depending on "the amount of applications, number of runs and complexity of reservoir numerical model" [14].

Two approaches to the search for improved numerical parameters have been investigated. Firstly, CMG Designed Exploration and Controlled Evolution, a commercial algorithm, was used to tune the parameters [14,44]. Later work trained several machine learning algorithms to find a suitable oracle which could provide parameter settings based on output from previous simulations [15].

Optimising without controlling the accuracy of the simulation results might lead to improved performance, but in the end useless results. Hence, an important point in previous works has been to find a way to control the accuracy of the simulation results — that is, to make sure that no huge numerical errors have been introduced by the changes in the numerical algorithm. This has traditionally been done by looking at the mass balance equation (MBE) error. At the heart of porous media flow lies the assumption of mass conservation. Hence, for each time step of the simulation, the total mass (adjusted for injections and extractions) should stay constant. Any inaccuracies caused by the approximate nature of the numerical algorithm will likely result in the MBE error being non-zero. However, by making sure that the MBE error relative to the total mass initially in place stays sufficiently close to zero, we can assume that the simulation results are accurate enough. Ideally, one would like to look at the MBE error for each cell, but the practice adopted has been to only check the global MBE error [14,15].

The work described in this article distinguishes itself from previous work (as outlined above) in the way it solves the two problems (1) how we optimise the numerical parameters in order to reduce the simulation time, and (2) how we make sure that the simulation results are accurate enough.

Our approach to the optimisation procedure does not rely on any proprietary software, and can be easily implemented by users in a programming language of their choice. It also lends itself easily to substitution of the particular parameters which should be tuned, in contrast to the oracle-approach [15] which will require retraining the oracle after each alteration of the parameter list.

The accuracy control used in this work includes much more information from each simulation than the approach of using the global MBE error adopted in similar work [14,15,44]. While this is no guarantee that the accuracy control works better, it should increase the likelihood of picking up small divergences between simulation results typically due to the new simulation being slightly inaccurate. It should also make it easier to tune the thresholds in such a way that a desired degree of accuracy is more or less guaranteed.

6. Discussion

In this work we have presented an automated optimisation procedure. More concretely, we have detailed an algorithm to move through a parameter space in search for an optimal set of parameter values, which boosts the rate of improvement compared to a simple random search. We have then used this method to iteratively find improved sets of parameter values (i.e. values which result in less and less computing time spent on the simulation) in the case of reservoir simulations.

In order to ensure the quality of the simulation results, we have incorporated into the search algorithm five measures for comparing time series data, and used these to check the accuracy of the well bottom-hole pressure (WBHP) for the different wells in the reservoir model. A too large deviation, when comparing a new simulation result with that from a simulation using default parameters, indicates an inadequate accuracy.

For example, compared with simulations run with the default parameters, the mean improvement in run-time when running the optimisation algorithm on the Norne model for 30 iterations is around 20%. Cutting the search earlier (e.g. after one third or half the number of iterations) does not reduce the improvement too much, as the steepest improvement in run-time takes place early on in the optimisation.

The method presented in this paper has the advantage of being highly flexible. One could keep the similarity measures, while replacing the optimisation procedure, or one could do the opposite. It is also possible to introduce new similarity measures to complement or replace the existing ones. Furthermore, the choice of the target time series data as input to the similarity measures can be changed based on the choice of model and application. As an example in the reservoir simulation context, one could use pressure and/or saturation for each individual computational cell instead of the WBHP for each individual well, or one could of course use both in combination.

One limitation of the work presented here is the dependence on a sufficiently accurate reference simulation result, and the assumption that the default parameters produce such a baseline result. While investigations into the Norne black-oil model, running multiple simulations with very strict parameter values, do indicate that the default parameters provide sufficiently accurate results in this case, the present work offers no general method of making sure that this is indeed the case.

Due to a combination of the desire to make use of open-source models and the limited availability of proprietary models, the present work makes use of models which are arguably quite small in scale. Hence, a remaining question is whether or not the findings are applicable to larger (more realistic) industrial models. The only use of proprietary models can be found in Section 4.5, whereas the model itself was never used in the optimisation procedure described in Section 3.

As future work, there are several points worthy of investigation. Throughout this work, 32 MPI processes have been used for all simulations of the three open-source reservoir models. Although some tests were carried out to ascertain the stability of the optimisation procedure with regards to the number of MPI processes, one should take a closer look at how different numbers of MPI processes might affect the results of the optimisation procedure. Another topic is how to best tune optimisation settings such as the choice of the constant α in the optimisation target, the threshold values for the similarity measures, the probability of perturbing parameters and the number of perturbations per search iteration.

Furthermore, there are several possible similarity measures which might be used in place of the five chosen for this work. One version of the cosine similarity using adjusted arrays of the consecutive differences [38] seems an interesting alternative. Also, measures which do not require coinciding time steps [41] (and hence can work without the use of interpolation methods) should be tested. Also, the threshold value for each individual similarity measure might be tuned in order to get a better balance between retaining the simulation results which are accurate enough and discarding the ones which are not.

In this work, for all the historical reservoir models, we have made use of the well bottom-hole pressure (WBHP) to gauge the degree of (dis)similarity between different simulation results. While this seems to work well, it is worth investigation whether other approaches might work similarly or better. One such approach is to look at the pressure and saturation for each cell in the grid and compare these across simulation results.

A central part of the automated search algorithm introduced in this work is the optimisation target from Eq. (15). This is a rather crude model of the computing time required to complete a simulation, and hence it may be necessary to include cross terms and/or higher order terms of the linear and Newton iterations in the objective function. One might also include other variables which also contribute to the computing time. Another topic that is worth a detailed investigation is Eq. (12) which has been used to find a new sample by the gradient-inspired step at the end of each search iteration. Other formulations can be tested.

Lastly, it remains to be investigated whether numerical parameters that are automatically identified by the optimisation procedure (when applied to one specific reservoir model) can lead to similar performance improvements for other simulations in an ensemble. A related question is the suitability of the automated optimisation procedure for larger reservoir models. Assuming that the number of numerical parameters involved in large reservoir models is the same as in small models (such as Norne), we expect the optimisation procedure to have the same effectiveness, i.e., in terms of iteration and sample numbers. However, the optimisation procedure will inevitably be very time consuming. A possible approach is to apply the optimisation procedure to a similar but much smaller reservoir model. The hope is that the improved parameters can also benefit a much larger reservoir model. The experiments reported in Section 4.5 are promising, but extensive experimentation is needed.

CRediT authorship contribution statement

Erik Hide Sæternes: Conceptualization, Data curation, Formal Analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Andreas Thune:** Data curation, Investigation, Software, Writing – original draft, Writing – review & editing. **Alf Birger Rustad:** Conceptualization, Funding acquisition, Resources, Writing – review & editing. **Tor Skeie:** Conceptualization, Funding acquisition, Project administration, Supervision, Writing – original draft, Writing – review & editing. **Xing Cai:** Conceptualization, Funding acquisition, Methodology, Project administration, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The work of Erik Hide Sæternes and Andreas Thune was partially supported by the Research Council of Norway under Grant 237898. The work of Xing Cai was supported in part by the European High-Performance Computing Joint Undertaking under Grant 956213, and in part by the Research Council of Norway under Grant 329017. The research presented in this paper benefited from the Experimental Infrastructure for the Exploration of Exascale Computing (eX3), which was financially supported in part by the Research Council of Norway under Grant 270053.

References

[1] OPM, The open porous media initiative, https://opm-project.org/.

[2] A.F. Rasmussen, T.H. Sandve, K. Bao, A. Lauser, J. Hove, B. Skaflestad, R. Klöfkorn, M. Blatt, A.B. Rustad, O. Sævareid, K.-A. Lie, A. Thune, The open porous media flow reservoir simulator, Comput. Math. Appl. 81 (2021) 159–185, http://dx.doi.org/10.1016/j.camwa.2020.05.014.

[3] Encyclopedia of Mathematics, Linear interpolation, https://encyclopediaofmath.org/index.php?Title=Linear_interpolation&oldid=27068.

[4] Encyclopedia of Mathematics, Correlation (in statistics), https://encyclopediaofmath.org/index.php?Title=Correlation_(in_statistics)&oldid=52436.

[5] B. Li, L. Han, Distance weighted cosine similarity measure for text classification, in: Intelligent Data Engineering and Automated Learning – IDEAL 2013, Springer Berlin Heidelberg, 2013, pp. 611–618, http://dx.doi.org/10.1007/978-3-642-41278-3_74.

[6] J. Lin, S.K.M. Wong, A new direct divergence measure and its characterization, Int. J. Gen. Syst. 17 (1) (1990) 73–81, http://dx.doi.org/10.1080/03081079008935097.

[7] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Stat. 22 (1) (1951) 79–86, http://dx.doi.org/10.1214/aoms/1177729694.

[8] J. Lin, Divergence measures based on the Shannon entropy, IEEE Trans. Inform. Theory 37 (1) (1991) 145–151, http://dx.doi.org/10.1109/18.61115.

[9] OPM, Open datasets, https://opm-project.org/?page_id=559.

[10] D.E. Knuth, Two notes on notation, Amer. Math. Monthly 99 (5) (1992) 403–422, http://dx.doi.org/10.2307/2325085.

[11] O.K. Foundation, Open data commons, https://opendatacommons.org/licenses/odbl/.

[12] OPM, Norne data models, https://github.com/OPM/opm-tests/tree/master/norne.

[13] Equinor ASA, Gassnova SF, Smeaheia dataset, https://co2datashare.org/dataset/smeaheia-dataset. http://dx.doi.org/10.11582/2021.00012.

[14] G. Avansi, V. Rios, D. Schiozer, Numerical tuning in reservoir simulation: It is worth the effort in practical petroleum applications, J. Braz. Soc. Mech. Sci. Eng. 41 (59) (2019) http://dx.doi.org/10.1007/s40430-018-1559-9.

[15] F. Portella, D. Buchaca, J.R. Rodrigues, J.L. Berral, TunaOil: A tuning algorithm strategy for reservoir simulation workloads, J. Comput. Sci. 63 (2022) http://dx.doi.org/10.1016/j.jocs.2022.101811.

[16] L. Yang, A. Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, Neurocomputing 415 (2020) 295–316, http://dx.doi.org/10.1016/j.neucom.2020.07.061.

[17] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng, M. Lindauer, Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges, WIREs Data Min. Knowl. Discov. (2023) http://dx.doi.org/10.1002/widm.1484.

[18] A.E.I. Brownlee, M.G. Epitropakis, J. Mulder, M. Paelinck, E.K. Burke, A systematic approach to parameter optimization and its application to flight schedule simulation software, J. Heuristics 28 (2022) 509–538, http://dx.doi.org/10.1007/s10732-022-09501-8.

[19] G. AU Baruzzo, K.E. Hayer, E.J. Kim, B. Di Camillo, G.A. FitzGerald, G.R. Grant, Simulation-based comprehensive benchmarking of RNA-seq aligners, Nature Methods 14 (2) (2017) 135–139, http://dx.doi.org/10.1038/nmeth.4106.

[20] A.A. Juan, J. Faulin, S.E. Grasman, M. Rabe, G. Figueira, A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems, Oper. Res. Perspect. 2 (2015) 62–72, http://dx.doi.org/10.1016/j.orp.2015.03.001.

[21] B.J. Shields, J. Stevens, J. Li, M. Parasram, F. Damani, J.I.M. Alvarado, J.M. Janey, R.P. Adams, A.G. Doyle, Bayesian reaction optimization as a tool for chemical synthesis, Nature 590 (2021) 89–96, http://dx.doi.org/10.1038/s41586-021-03213-y.

[22] S. Shan, G.G. Wang, Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions, Struct. Multidiscip. Optim. 41 (2010) 219–241, http://dx.doi.org/10.1007/s00158-009-0420-2.

[23] D. Floreano, C. Mattiussi, Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies, The MIT Press, 2008.

[24] A. Slowik, H. Kwasnicka, Evolutionary algorithms and their applications to engineering problems, Neural Comput. Appl. 32 (2020) 12363–12379, http://dx.doi.org/10.1007/s00521-020-04832-8.

[25] T.M. Shami, A.A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M.A. Summakieh, S. Mirjalili, Particle swarm optimization: A comprehensive survey, IEEE Access 10 (2022) 10031–10061, http://dx.doi.org/10.1109/ACCESS.2022.3142859.

[26] M. Dorigo, T. Stützle, Ant colony optimization: Overview and recent advances, in: Handbook of Metaheuristics, Springer International Publishing, 2019, pp. 311–351, http://dx.doi.org/10.1007/978-3-319-91086-4_10.

[27] Y. Sun, Y. Chen, Multi-population improved whale optimization algorithm for high dimensional optimization, Appl. Soft Comput. 112 (2021) 107854, http://dx.doi.org/10.1016/j.asoc.2021.107854.

[28] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey wolf optimizer, Adv. Eng. Softw. 69 (2014) 46–61, http://dx.doi.org/10.1016/j.advengsoft.2013.12.007.

[29] F. MiarNaeimi, G. Azizyan, M. Rashki, Horse herd optimization algorithm: A nature-inspired algorithm for high-dimensional optimization problems, Knowl.-Based Syst. 213 (2021) 106711, http://dx.doi.org/10.1016/j.knosys.2020.106711.

[30] A.H. Victoria, G. Maragatham, Automatic tuning of hyperparameters using Bayesian optimization, Evol. Syst. 12 (2021) 217–223, http://dx.doi.org/10.1007/s12530-020-09345-2.

[31] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, S.-H. Deng, Hyperparameter optimization for machine learning models based on Bayesian optimization, J. Electron. Sci. Technol. 17 (1) (2019) 26–40, http://dx.doi.org/10.11989/JEST.1674-862X.80904120.

[32] J. Serrà, J.L. Arcos, An empirical evaluation of similarity measures for time series classification, Knowl.-Based Syst. 67 (2014) 305–314, http://dx.doi.org/10.1016/j.knosys.2014.04.035.

[33] T.W. Liao, Clustering of time series data — A survey, Pattern Recognit. 38 (11) (2005) 1857–1874, http://dx.doi.org/10.1016/j.patcog.2005.01.025.

[34] C. Holder, M. Middlehurst, A. Bagnall, A review and evaluation of elastic distance functions for time series clustering, Knowl. Inf. Syst. (2023) http://dx.doi.org/10.1007/s10115-023-01952-0.

[35] A. Kianimajd, M.G. Ruano, P. Carvalho, J. Henriques, T. Rocha, S. Paredes, A.E. Ruano, Comparison of different methods of measuring similarity in physiologic time series, IFAC-PapersOnLine 50 (1) (2017) 11005–11010, http://dx.doi.org/10.1016/j.ifacol.2017.08.2479.

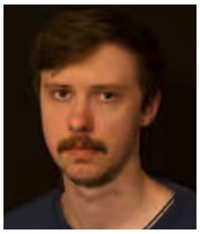
[36] S. Lhermitte, J. Verbesselt, W.W. Verstraeten, P. Coppin, A comparison of time series similarity measures for classification and change detection of ecosystem dynamics, Remote Sens. Environ. 115 (12) (2011) 3129–3152, http://dx.doi.org/10.1016/j.rse.2011.06.020.

[37] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. Keogh, Experimental comparison of representation methods and distance measures for time series data, Data Min. Knowl. Discov. 26 (2013) 275–309, http://dx.doi.org/10.1007/s10618-012-0250-5.

[38] T. Nakamura, K. Taki, H. Nomiya, K. Seki, K. Uehara, A shape-based similarity measure for time series data with ensemble learning, Pattern Anal. Appl. 16 (2013) 535–548, http://dx.doi.org/10.1007/s10044-011-0262-6.

[39] D.M. Mateos, L.E. Riveaud, P.W. Lamberti, Detecting dynamical changes in time series by using the Jensen Shannon divergence, Chaos 27 (8) (2017) 083118, http://dx.doi.org/10.1063/1.4999613.
[40] L. Zunino, F. Olivares, H.V. Ribeiro, O.A. Rosso, Permutation Jensen-Shannon distance: A versatile and fast symbolic tool for complex time-series analysis, Phys. Rev. E 105 (4) (2022) 045310, http://dx.doi.org/10.1103/PhysRevE.105.045310.
[41] M. Zhang, D. Pi, A new time series representation model and corresponding similarity measure for fast and accurate similarity detection, IEEE Access 5 (2017) 24503–24519, http://dx.doi.org/10.1109/ACCESS.2017.2764633.
[42] I.D. Mishev, N. Fedorova, S. Terekhov, B.L. Beckner, A.K. Usadi, M.B. Ray, O. Diyankov, Adaptive control for solver performance optimization in reservoir simulation, in: Conference Proceedings, ECMOR XI - 11th European Conference on the Mathematics of Oil Recovery, 2008, http://dx.doi.org/10.3997/2214-4609.20146368.
[43] D. Bagaev, I. Konshin, K. Nikitin, Dynamic optimization of linear solver parameters in mathematical modelling of unsteady processes, in: Supercomputing, Springer International Publishing, 2017, pp. 54–66, http://dx.doi.org/10.1007/978-3-319-71255-0_5.
[44] V.S. Rios, G.D. Avansi, D.J. Schiozer, Practical workflow to improve numerical performance in time-consuming reservoir simulation models using submodels and shorter period of time, J. Pet. Sci. Eng. 195 (2020) http://dx.doi.org/10.1016/j.petrol.2020.107547.

Erik Hide Sæternes received the M.Sc. degree in Industrial mathematics from the Department of Mathematical Sciences at the Norwegian University of Science and Technology (NTNU) in 2020. He is currently working toward a Ph.D. degree with the University of Oslo and Simula Research Laboratory. His research topic is high performance computing in the domain of reservoir simulation.



Andreas Thune received the M.Sc. degree from the Department of Mathematics at the University of Oslo in 2017. He is currently working toward the Ph.D. degree with the University of Oslo and Simula Research Laboratory. His research topic is high performance computing in the domain of reservoir simulation.

Alf Birger Rustad received his Ph.D. degree in Mathematics at the Norwegian University of Science and Technology (NTNU) in 2001. He held a position of Associate Professor in mathematics at NTNU between 2001 and 2003. Dr. Rustad has worked at Equinor (previously called Statoil) since 2003. He is currently project manager for OPM reservoir simulator development at Equinor's research Centre in Trondheim, Norway.

Tor Skeie is a professor with the University of Oslo and Simula Research Laboratory; his research has mainly contributed to the High-Performance Computing field (HPC). Herein he has focused on effective routing, fault tolerance, congestion control, quality of service, reservoir simulations, edge and cloud computing. He is also a researcher with the Industrial Ethernet and wireless networking areas. Several of his research results have been published in the most respected IEEE transactions and magazines.

Xing Cai received the Ph.D. degree in scientific computing from the University of Oslo, in 1998, and was appointed to the position of associate professor with the University of Oslo, in 1999, later promoted to full professorship in 2008. He joined Simula with its very beginning in 2001, taking an 80 % leave from his university position. His research interests include parallel programming and high-performance scientific computing on multi-core CPUs and GPUs, numerical methods for solving PDEs, and generic PDE software.