XL CILAMCE
IBERO-LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING
NOVEMBER 11-14, 2019
Praiamar Natal Hotel & Convention
Natal, RN-BRAZIL

TRAJECTORY PIECEWISE LINEARIZATION (TPWL) USING THE MATLAB RESERVOIR SIMULATION TOOLBOX (MRST) FOR RESERVOIR SIMULATION

Andresa Dornelas de Castro
Alexandre de Sousa Júnior
Bernardo Horowitz
andresa_dornelas@hotmail.com
souza.alexandrejr@gmail.com
horowitz@ufpe.br
Federal University of Pernambuco
Av. Prof. Moraes Rego,1235-Cidade Universitária, Zip Code: 50670-901, PE, Recife, Brasil

Abstract. The goal of the present work is to show the Trajectory Piecewise Linearization (TPWL) procedure for modeling of two-phase flow in subsurface formations, using the Matlab Reservoir Simulation Toolbox - MRST for reservoir simulation, developed by SINTEF Digital. In this work we will show how MRST can be used to simulate black-oil models using automatic differentiation to compute the Jacobian matrices required for the nonlinear Newton-type solver. In automatic differentiation (AD) the key idea is to keep track of quantities and their derivatives simultaneously: every time an operation is applied to a quantity, the corresponding differential operation is applied to its derivative. Thus, MRST-AD allows the fully-implicit nonlinear pressure equation, using a discrete operators and equations simulator, to compute automatically the correct Jacobian for black-oil system. In general, the simulator code presented in automatic differentiation black-oil module (*ad – blackoil)* is modified to export the necessary data to build the TPWL procedure. This is known as semi-intrusive methodology, since it requires knowledge of the simulator code, but the changes do not affect the equations solutions. The Trajectory Piecewise Linearization (TPWL) procedure reduces the numerical complexity of the problem by performing the linearization of governing equations around converged states stored during a training simulation. Therefore, we will show implementation of TPWL after getting the system states (pressure and saturation) and the derivatives of residual equation from the MRST. The method is shown to be accurate in the neighborhood of the training trajectory.

Keywords: Matlab Reservoir Simulation Toolbox, Trajectory Piecewise Linearization, Reservoir Simulation.

1 Introduction

Each day it is possible to realize the importance of the need to use reliable computational modeling and simulation for a correct prediction of subsurface or underground flow. In the present work the numerical simulation will be applied in oil and gas reservoirs, specifically in porous oil and water saturated media.

Currently, one of the best known and widely used open access simulators is the MATLAB Reservoir Simulation Toolbox – MRST. MRST was primarily developed by SINTEF Digital and has been published online as a free open-source code for reservoir modelling and simulation under the GNU General Public License since 2009. MRST is not essentially a simulator, but is mainly intended as a toolbox for rapid prototyping of models and demonstration of new simulation methods and modeling concepts.

Different modules provide different simulators for incompressible or compressible fluid with various discretization possibilities (Two-Point Flux Approximation (TPFA), Multi-Point Flux Approximation (MPFA), mimetic and others), allowing the use of upscaling techniques or special models such as geomechanics and fractured reservoirs. For these reasons, MRST will be used in this work as a simulation tool.

Thus, in this article, after a brief theoretical and mathematical foundation regarding the reservoir simulation, the MRST framework will be presented, approaching its main basic features for model generation, emphasizing specifically in solvers presented in ad-blackoil module.

Like most commercial simulators based on a black-oil formulation, MRST provides a fully implicit simulator, offering unconditional stability for a wide range of flow regimes and reservoir heterogeneities. In addition, fully implicit formulation is combined with automatic differentiation, ensuring simple extension of basic flow models. Using numerical routines and MATLAB vectorization combined with MRST differential discrete operators, the model equations can be implemented very compactly and close to the mathematical formulation.

MATLAB is efficient and fully comparable with compiled languages. Tests on two- and three-phase models with the order of ten to hundred thousand cells show that MRST simulators based on automatic differentiation are between two and ten times slower than fully optimized commercial simulator (BAO et al., 2017).

After understanding the structure of MRST we will export the necessary information for the implementation of Trajectory Piecewise Linearization (TPWL) procedure for modeling of two-phase flow in subsurface formations. This technique reduces the complexity of the problem by linearizing its governing equations.

In this methodology a nonlinear system is represented as a weighted combination of piecewise linear systems. A key feature of the TPWL method is that during subsequent simulations it linearizes around one or more states selected from a large collection of snapshots. New states are represented in terms of piecewise linear expansions around previously simulated (and saved) states and Jacobian matrices. (CARDOSO, 2009)

This article starts by describing the equations and the standard finite volume discretization for two-phase flow, followed by a discussion about the MRST-AD and TPWL technique. Finally it is presented a initial test for the described and implemented methodology.

2 Reservoir Simulation

This section starts by presenting an outline the flow equations for a general black-oil model. Then, the governing equations and the discretized system for oil-water flows are described, since, in the present work, the intention is to apply TPWL in MRST for two-phase flow.

2.1 The black-oil model

For reservoir simulation purposes, one normally uses either so-called Black-Oil fluid description, or compositional fluid description. For now, it will be considered the Black Oil model. The term "black-oil" refers to the fluid model, in which water is modeled explicitly together with two hydrocarbon components, oil and gas. This is in contrast with a compositional formulation, in which each hydrocarbon component (arbitrary number) is handled separately.

At reservoir conditions, the two components (oil and gas) can be partially or completely dissolved in each other, depending on the pressure, forming a liquid phase and a gaseous phase. In addition, there is an aqueous phase, which herein it is assumed to be consisted by only water. The corresponding continuity equations are described as it follows:

$$\partial_t\left(\phi b_w s_w\right)+\nabla\cdot\left(b_w v_w\right)-b_w q_w=0\ . \tag{1}$$

$$\partial_t\left[\phi\left(b_o s_o+b_g r_v s_g\right)\right]+\nabla\cdot\left(b_o v_o+b_g r_v v_g\right)-\left(b_o q_o+b_g r_v q_g\right)=0\,. \tag{2}$$

$$\partial_t\left[\phi\left(b_g s_g+b_o r_s s_o\right)\right]+\nabla\cdot\left(b_g v_g+b_o r_s v_o\right)-\left(b_g q_g+b_o r_s q_o\right)=0\,. \tag{3}$$

Where ϕ is the porosity of the rock while s_α denotes saturation, p_α is a phase pressure, q_α is the volumetric source, for phase α (w, o, g) respectively, and b_α is the inverse formation-volume factor, which measure the ratio between the bulk volumes of a fluid component occupied at surface and reservoir conditions. Furthermore, it is defined the gas-oil ratio r_s and oil-gas ratio r_v, which measure the volumes of gas dissolved in the oleic phase and oil vaporized in the gaseous phase, respectively.

At last, the phase fluxes v_α are given from Darcy's law, defined by Eq. (4):

$$v_\alpha=-\lambda_\alpha K\left(\nabla p_\alpha-\rho_\alpha g\nabla z\right)\qquad \alpha=o,w,g\ . \tag{4}$$

Here, K is the absolute permeability of the reservoir rock, while $\lambda_\alpha=kr_\alpha/\mu_\alpha$ is the mobility of phase α, where kr_α is the relative permeability and μ_α is the phase viscosity.

We can also specify the q_α by the well equation, presented below.

$$q_\alpha=-\lambda_\alpha WI\left(p_{bh}-p\right). \tag{5}$$

Where p is the reservoir pressure, p_{bh} is the bottom-hole pressure and WI is the well index.

The model is closed by assuming that the fluids fill the pore space completely, $s_o+s_w+s_g=1$, and by supplying saturation-dependent capillary functions that relate the phase pressures. Then, the equation system will have three primary unknowns.

2.2 Oil-Water Flow Equations and Discretization

In this work, it is specifically studied the two-phase flow. In this case, from Eq.(1) to Eq.(4), the model system is described by equations (6) and (7):

$$\partial_t\left(\phi b_w s_w\right)-\nabla\cdot\left[b_w\lambda_w K\left(\nabla p_w-\rho_w g\nabla z\right)\right]-b_w q_w=0\,. \tag{6}$$

$$\partial_t\left(\phi b_o s_o\right)-\nabla\cdot\left[b_o\lambda_o K\left(\nabla p_o-\rho_o g\nabla z\right)\right]-b_o q_o=0\,. \tag{7}$$

The general two-phase flow description is completed through the saturation constraint $\left(s_w+s_o=1\right)$ and by specifying a capillary pressure relationship by $p_c(s_W)=p_o-s_w$.

The two-phase flow description entails four equations and four unknowns (p_o, p_w, S_o, S_w). p_o and S_w are selected as primary unknowns. Once these are computed, p_w and S_o can be readily determined from the capillary pressure relationship and saturation constraint.

We now briefly discuss the finite volume representation for Eq.(6) and Eq.(7). For the discretized models, we consider logically Cartesian systems (meaning blocks follow a logical i, j, k ordering) containing a total of Nc grid blocks. Figure 1 shows a portion of a one-dimensional grid. For

simplicity, here capillary pressure and gravitational effects are neglected, so $p_o = p_w$. It is also considered horizontal flow in the x-direction, assuming that the grid block dimensions (Δx, Δy, Δz) are constant and adopting the incompressible case, which density does not vary with pressure and $b_\alpha = 1$. In addition to that, it is applied fully implicit discretizations, usually used in commercial simulators.

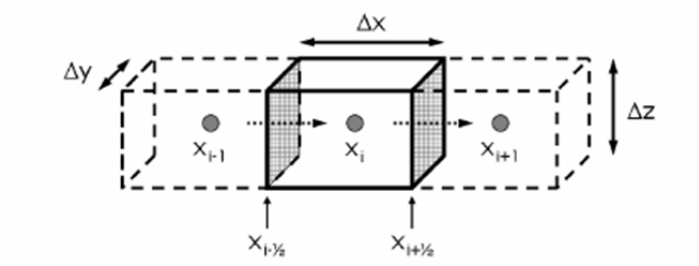


Figure 1. Portion of a one-dimensional grid. Source: (CARDOSO, 2009)

The first term in Eq.(6) and Eq.(7) represents mass accumulation, discretely, it can be represented by:

$$\partial_t\left(\phi b_\alpha s_\alpha\right) \approx \frac{1}{\Delta t}\left[\left(\phi b_\alpha s_\alpha\right)^{n+1} - \left(\phi b_\alpha s_\alpha\right)^{n}\right]. \tag{8}$$

Where subscript α indicates phase, superscript n is the time step and n + 1 specifies the next time step. For the mentioned case, incompressible systems ϕ is constant and $b_\alpha = 1$, Eq.(8) is reduced by:

$$\partial_t\left(\phi b_\alpha s_\alpha\right) \approx \frac{\phi}{\Delta t}\left[\left(s_\alpha\right)^{n+1} - \left(s_\alpha\right)^{n}\right]. \tag{9}$$

The first term in Eq.(6) and Eq.(7) represents flow terms, its discretized form is given by:

$$\nabla\cdot\left[b_\alpha\lambda_\alpha K\left(\nabla p_\alpha\right)\right] = \frac{\partial}{\partial x}\cdot\left[K\lambda_\alpha\left(\frac{\partial p_\alpha}{\partial x}\right)\right] \approx \left\{\left(T_\alpha\right)_{i-1/2}^{n+1}\left(p_{i-1}^{n+1} - p_i^{n+1}\right) + \left(T_\alpha\right)_{i+1/2}^{n+1}\left(p_{i+1}^{n+1} - p_i^{n+1}\right)\right\}\frac{1}{V} \tag{10}$$

Where i designates grid block and $V = \Delta x\Delta y\Delta z$ is the volume of grid block i. The transmissibility $\left(T_\alpha\right)_{i-1/2}$ relates flow in phase α to the difference in pressure between grid blocks i-1and i given by:

$$\left(T_\alpha\right)_{i-1/2}^{n+1} = \left(\frac{KA}{\Delta x}\right)_{i-1/2}\left(b_\alpha\lambda_\alpha\right)_{i-1/2}^{n+1}. \tag{11}$$

Where $A = \Delta y\Delta z$ is the area of the common face between blocks i-1 and i. The transmissibility $\left(T_\alpha\right)_{i+1/2}^{n+1}$ is defined the same way. In Eq. (11) $K_{i-1/2}$ is computed as the harmonic average of K_{i-1} and K_i, and $\left(b_\alpha\, kr_\alpha/\mu_\alpha\right)_{i-1/2}^{n+1}$ is upwinded depending on the flow direction of phase j.

The last term in Eq.(6) and Eq.(7) represents source/sink term. In reservoir simulation, the source/sink correspond to wells which are modeled using well equation [Eq.(5)]:

$$\left(q_\alpha\right)_i^{n+1} = \left(q_\alpha\right)_i^{n+1} V = -\cdot WI\cdot\left(\lambda_\alpha\right)_i^{n+1}\left(p_{bh} - p_i^{n+1}\right). \tag{12}$$

Where $\left(q_\alpha\right)_i^{n+1}$ is the volumetric flow rate of phase α from block *i* into the well (or vice versa) at time *n+1*, p_i^{n+1} is grid block pressure at *n+1*, p_{bh} is the bottomhole pressure for well *w* in grid block i, and *WI* is the well index, described using a standard Peaceman type well model.

Introducing the discretizations presented in Eqs. (8) through (12), the discretized fully implicit version of Eqs. (6) and (7) for incompressible systems can be expressed as (K. Aziz and A. Settari, 1986):

$$g = T^{n+1}x^{n+1} - D^{n+1}\left(x^{n+1} - x^{n}\right) - Q^{n+1} = 0. \tag{13}$$

Where g is the residual vector one seeks to drive to zero, x is the state vector that contains the two primary unknowns (po and Sw), T is a block pentadiagonal matrix for two-dimensional grids and a block heptadiagonal matrix for three-dimensional grids, D is a block diagonal matrix, and Q represents the source/sink terms expressed by Eq. (12). Note that the time level is designated by the superscript n or n + 1. The term $T^{n+1}x^{n+1}$ represents transport (flow) effect obtained from Eq. (10) while the $D^{n+1}\left(x^{n+1}-x^{n}\right)$ term represents accumulation derived by Eq. (8). The matrices T, D and Q depend on x and must be updated at each iteration of every time step.

Eq. (13) represents a nonlinear set of algebraic equations and is solved by applying Newton's method to drive g to zero:

$$J\delta = -g \tag{14}$$

Where J is the Jacobian matrix given by $J_{ij} = \partial g_i / \partial x_j$ and $\delta_i = x_i^{n+1,v+1} - x_i^{n+1,v}$ with v and v +1 indicating interation level.

3 Matlab Reservoir Simulation Toolbox (MRST)

The formulation presented in the previous section is be solved using a simulation toolbox, called Matlab Reservoir Simulation Toolbox (MRST).

The MRST was initially developed by SINTEF Digital and does not properly consist of a simulator, and can be understood more assertively as a toolbox for the demonstration of new simulation methods and modeling concepts. MRST has a variety of data structures and computational methods of its own that can be combined to create different numerical examples and custom simulation techniques. This paper aims to present the MRST and its application in the simulation of the ad-blackoil model. To this end, first, it will presented the structure of the software and some its functionalities.

3.1 Quick Overview of the Software

MRST is basically made up of two parts, as can be seen in the Figure 2. The essence of MRST is a central module, called *mrst-core*, which contains a flexible grid structure and a number of grid factory routines; routines for visualizing grids and data represented by cells and cell faces; basic functionality for representing petrophysical properties, boundary conditions, source terms and wells; computation of transmissibilities and data structures holding the primary unknowns; basic functionality for automatic differentiation (AD); and various low-level utility routines that can be applied to solve incompressible single and two-phase models on structured and unstructured grids. These routines, present in MRST-core have been quite stable over many years and are generally well documented in a format that follows the MATLAB standard.

The second and the largest part of the software consist of a set of *add-on modules* that extend, MRST-core. These set of add-on modules offer more advanced models, solvers, viewers, and workflow tools. All these modules are publicly available from the software's webpage (MRST, 2019).

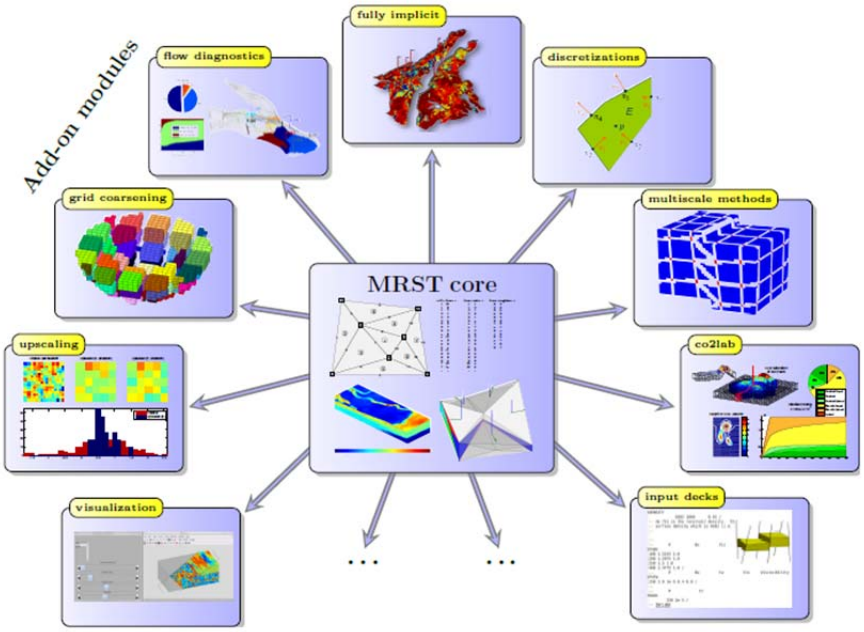


Figure 2. Organization of the Matlab Reservoir Simulation Toolbox - MRST (KROGSTAD, 2015).

In this work, we focus on the AD-OO family of modules rapid prototyping of fully implicit simulators, that includes *ad-core*, which is object-oriented framework for solvers based on automatic differentiation (MRST AD-OO) and *ad-blackoil*, that contains single-phase, two-phase and three-phase, its solvers support source terms, boundary conditions and complex wells with changing controls, production limits and multiple segments. The *ad-blackoil* module contains models for black-oil equations that inherit all basic features from *ad-core*.

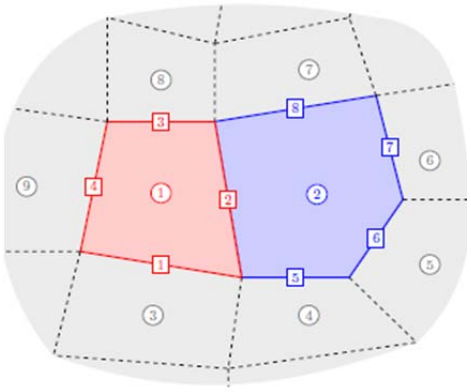
3.2 Grids

First, to understand the fully-implicit simulators developed in *ad-blackoil* module of the MRST, it will be introduced the basic functionality for grids implemented in MRST-core.

To ensure interoperability among a wide variety of different grid types and computational methods, grids are assumed to consist of a set of non-overlapping polyhedral cells, where each cell can have a varying number of planar faces that match the faces of the cell's neighbours. All grids are stored in a general format in which are explicitly represented cells, faces and vertices and the connections between cells and faces. Thus, MRST sacrificed some of the efficiency attainable by exploiting special structures in a particular grid type for the sake of generality and flexibility. (Lie et al, 2012). As a result, grid and also petrophysical properties are passed as input to almost all simulation and workflow tools in MRST.

The grid structure in MRST is called G and contains three fields: cells, faces, and nodes. Each of the n_c cells corresponds to a subset of the n_f faces, and each face to a set of edges, which again are determined by the nodes.

The topology of the grid is defined using two mappings. The first maps a cell to the set of faces that delimit this cell: F: $\{1\ldots n_c\}\rightarrow \{0,1\}^{nf}$. In the grid structure G this is represented as an array called G.cells.faces. The second mapping brings to a given face the two neighboring cells, N_1, N_2: $\{1\ldots n_f\} \rightarrow \{0\ldots n_c\}$, where 0 has been included to denote the exterior of the computational domain. In G, N_1 is given by G.faces.neighbors (: ,1) and N_2 by G.faces.neighbors(:,2). Figure 3 represents these two mappings. The cell and face objects also contain geometrical properties like centroids, volumes, areas, and normal vectors.



MRST c	G.cells.faces F (C)
1	1
1	2
1	3
1	4
2	5
2	6
2	7
2	8
2	2
3	1
⋮	⋮

MRST f	G.faces.neighbors (: ,1) N1	G.faces.neighbors (: ,2) N2
1	3	1
2	1	2
3	1	8
4	9	1
5	4	2
6	2	5
7	2	6
8	2	7
⋮	⋮	⋮

Figure 3. Grid mappings. Source: (LIE,2016).

To implement computational algorithms on the grid, vectorized index operations in combination with the mappings between cells and faces outline above are used. MRST contains several grid-factory routines for creating structured grids, including regular Cartesian, rectilinear and curvilinear grids, as well as unstructured grids, including Delaunay triangulations and Voronoi grids, and 3D grids created by extrusion of 2D shapes, and support for the industry standard corner-point grids given by the ECLIPSE input deck (LIE et al, 2012).A much more information about grid structure can be found in (LIE, 2014), detailed descriptions of how to construct such grids from an input file, using one of the many grid-factory routines that come with the software, or by writing your own grid-generation script.

3.3 Discrete operators

Even though discrete equations and all this associated information do not appear explicitly in MRST and in many simulators, it is important to know its construction to assembly of TPWL. In this section, it is shown how MRST builds abstract operators implementing powerful averaging and discrete differential operators that enable to write the discrete flow equations, present in *ad-blackoil* module, in a very compact form.

To form our discrete differential operators, two mappings presented in the previous section are basically needed. The div operator is a linear mapping from faces to cells. If v[f] denotes a discrete flux over face f with orientation from cell N1(f) to cell N2(f), then the divergence of this flux restricted to cell c is given by Eq. (15).

$$\mathrm{div}(v)[c] = \sum_{f \in F(c)} \mathrm{sgn}(f)\, v[f] \quad \mathrm{sgn}(f) = \begin{cases} 1, \text{se } c = N_1(f) \\ -1, \text{se } c = N_2(f) \end{cases}. \tag{15}$$

The grad operator maps from cell pairs to faces. If, for instance, p denotes the array of discrete cell pressures, the gradient of this cell pressure restricted to face f is defined as follows:

$$\mathrm{grad}(p)[f] = p[N_2(f)] - p[N_1(f)]. \tag{16}$$

If we assume no-flow conditions on the outer faces, the discrete gradient operator is the adjoint of the divergence operator as in the continuous case. Furthermore, we define the transmissibilities that describe the flow across a cell face f given a unit pressure drop between the two neighboring cells i $=N_1(f)$ and k = $N_2(f)$. To this end, let $A_{i,k}$ denote the area of the face, $n_{i,k}$ the normal to this face, and $c_{i,k}$ the vector from the centroid of cell i to the centroid of the face. Then, the transmissibility is defined as follows:

$$T\left[f\right]=\left[T_{i,k}^{-1}+T_{k,i}^{-1}\right]^{-1},\ T_{i,k}=A_{i,k}K_i\frac{c_{i,k}\cdot n_{i,k}}{\left|c_{i,k}\right|^2}. \tag{17}$$

Where $\mathbf{K}_i$ is the permeability tensor in cell *i*. Finally, to provide a complete discretization, it is necessary to define averaging operators that maps rock and fluid properties from cells to faces. For this, it is used arithmetic averaging, which in its simplest form can be written as follows:

$$\mathrm{avg}_{\alpha}\left[\mathrm{f}\right]=\frac{1}{2}\left(\mathrm{q}\left[\mathrm{N}_1\left(\mathrm{f}\right)\right]+\mathrm{q}\left[\mathrm{N}_2\left(\mathrm{f}\right)\right]\right). \tag{18}$$

3.4 Automatic differentiation in MRST

Automatic or algorithmic differentiation (NEIDINGER, 2010) is a technique that exploits the fact that any computer code, regardless of complexity, can be broken down to a limited set of arithmetic operations and evaluation of simple functions.

The key idea is to keep track of variables and their derivatives simultaneously; every time an operation is applied to a variable, the appropriate differential operation is applied to its derivative.

Consider a scalar primary variable x and a function f = f(x). Their AD-representations are the pairs $\langle \mathrm{x},1\rangle$ and $\langle \mathrm{f},\mathrm{f_x}\rangle$ where 1 is the derivate dx/dx and $\mathrm{f_x}$ is the derivative of f with respect to x. Accordingly, the action of the elementary operations and functions must be defined for such pairs, e.g., $\langle \mathrm{f},\mathrm{f_x}\rangle+\langle \mathrm{g},\mathrm{g_x}\rangle=\langle \mathrm{f}+\mathrm{g},\mathrm{f_x}+\mathrm{g_x}\rangle$ or $\langle \mathrm{f},\mathrm{f_x}\rangle*\langle \mathrm{g},\mathrm{g_x}\rangle=\langle \mathrm{f}\cdot\mathrm{g},\mathrm{f}\cdot\mathrm{g_x}+\mathrm{f_x}\cdot\mathrm{g}\rangle$. In addition, one needs to use the chain rule to accumulate derivatives, that is, if $\mathrm{f}\left(\mathrm{x}\right)=\mathrm{g}\left(\mathrm{h}\left(\mathrm{x}\right)\right)$, then $\mathrm{f_x}\left(\mathrm{x}\right)=\mathrm{dg}/\mathrm{dh}\cdot\mathrm{h_x}\left(\mathrm{x}\right)$. This summarizes the key idea behind automatic differentiation: writing the formulas and specifying the independent variables, the software computes the corresponding derivatives or Jacobians.

To implement AD in Matlab, it is used the operator overloading. When Matlab encounters an expression of the form a+b, the software will choose one out of several different addition functions depending on the data types of a and b. Therefore, all one needs to do is to introduce new addition functions for the various classes of data types that a and b may belong to.

Automatic differentiation in MRST use a list of matrices that represent the derivatives with respect to different variables that will constitute sub-blocks in the Jacobian of the full system, instead of working with a single Jacobian of the full discrete system as one matrix. (KROGSTAD, 2015)

Now, we have all the necessary tools to implement a fully-implicit simulator for the nonlinear pressure equation. The key parts of the implementation are: initialization, creation of discrete operators, specification of constitutive functions, specification of discrete equations, and setup and solution of the linearized system inside the time loop. A complete example, including all code lines necessary to generate grid, petrophysical parameters, and well positions, is given in Lie (2014).

3.5 Discrete Flow equations for black-oil model in MRST

The techniques outlined above are basically everything which it is needed to efficiently implement fully-implicit simulators. The most widely used fluid model in reservoir simulation is the black-oil family, which contains three components and the three phases: water, oil, and gas.

The discrete operators defined in section 3.3 can be used to discretize the flow equations in a very compact form. If we use a first-order, implicit temporal discretization and a standard two-point spatial discretization with upstream weighting like present in section 2.2, the discrete conservation for the aqueous phase (Eq. (1)) can be written as

$$\frac{1}{\Delta \mathrm{t}}\left(\phi[\mathrm{c}]\mathrm{b}[\mathrm{c}]\mathrm{s}[\mathrm{c}]\right)^{\mathrm{n}+1}-\frac{1}{\Delta \mathrm{t}}\left(\phi[\mathrm{c}]\mathrm{b}[\mathrm{c}]\mathrm{s}[\mathrm{c}]\right)^{\mathrm{n}}+\mathrm{div}\left(\mathrm{bv}\right)[\mathrm{c}]^{\mathrm{n}+1}-\left(\mathrm{b}[\mathrm{c}]\mathrm{q}[\mathrm{c}]\right)^{\mathrm{n}+1}=0 \tag{19}$$

where we have omitted the subscript 'w' for brevity. To evaluate the product of the inverse formation-volume factor and the phase flux at the cell interfaces, the operator for extracting the upstream value are introduced in MRST:

$$\mathrm{upw}(\mathrm{h})[\mathrm{f}] = \begin{cases} \mathrm{h}\left[\mathrm{N}_1(\mathrm{f})\right], \text{if } \mathrm{grad}(\mathrm{p})[\mathrm{f}] - \mathrm{g}\,\mathrm{avg}_\alpha(\rho)[\mathrm{f}]\mathrm{grad}(\mathrm{z})[\mathrm{f}] > 0, \\ \mathrm{h}\left[\mathrm{N}_2(\mathrm{f})\right], \text{otherwise.} \end{cases} \tag{20}$$

Then, the discrete version of Darcy's law multiplied by b_w is

$$(\mathrm{bv})[\mathrm{f}] = -\mathrm{upw}(\mathrm{b}\lambda)[\mathrm{f}]\mathrm{T}[\mathrm{f}]\left(\mathrm{grad}(\mathrm{p})[\mathrm{f}] - \mathrm{g}\,\mathrm{avg}_\alpha(\rho)[\mathrm{f}]\mathrm{grad}(\mathrm{z})[\mathrm{f}]\right). \tag{21}$$

Similarly it is done to get the discrete conservation equations for oleic, and gaseous phases. These equations along with the well equations - all written on residual form – results in a system of nonlinear equation as

$$\mathrm{R}(\mathrm{x}) = 0 \quad . \tag{22}$$

Where x is the vector of unknown state variables at the next time step. The standard way to solve such a nonlinear system is to use Newton's method. That is, we write $x = x^0 + \Delta x$, and use a standard multidimensional Taylor expansion to derive the iterative scheme,

$$\mathrm{J}(\mathrm{x}^i)\left(\mathrm{x}^{i+1} - \mathrm{x}^i\right) = -\mathrm{R}(\mathrm{x}^i). \tag{23}$$

Where $\mathrm{J} = \mathrm{dR}/\mathrm{dx}$ is the Jacobian matrix of the residual equations.

3.6 Object-oriented implementation in MRST

The discrete equations described above are present in the function called *equationsBlackOil* from the *ad-blackoil* module.

As exposed by Bao et al. (2017), in all their generality, black-oil models can be very computationally challenging. To ensure that saturations stay within their physical bounds, each Newton update needs to be accompanied by a stabilization method that either crops, dampens, or performs a line search along the update directions. Likewise, additional logic is needed to map the updated primary variables back to a consistent reservoir state, switch primary variables as phases appear or disappear, trace changes in fluid components to model hysteretic behavior, etc. To get a robust simulator, it is also need to introduce sophisticated time-step control that monitors the iteration and cuts the time step if this is deemed necessary to improve convergence. And finally, it is also need procedures for updating the well controls in response to changes in the reservoir state and the injection and production of fluids. Introducing all this functionality in a procedural code is possible, but can easily give unwieldy code. A lot of this functionality is also to a large degree generic and can be reused from one model/simulator to another.

This motivated the development of an object-oriented AD framework in *ad-core* module that enables the user to separate physical models and reservoir states, nonlinear solvers and time loops, discrete flow equations and linearizations, and linear solvers like exposed in Figure 4. Notice that various classes, structures, and functions can be organized to formulate an efficient black-oil simulator.

Also note that assembly of the linearized system is due to a special class that stores meta-information about the residual equations (i.e., whether they are reservoir, well, or control equation) and the primary variables. In the case of oil and water flow, this class is called *TwoPhase OilWaterModel.* This information is useful when setting up strategies that utilize structures in the problem.

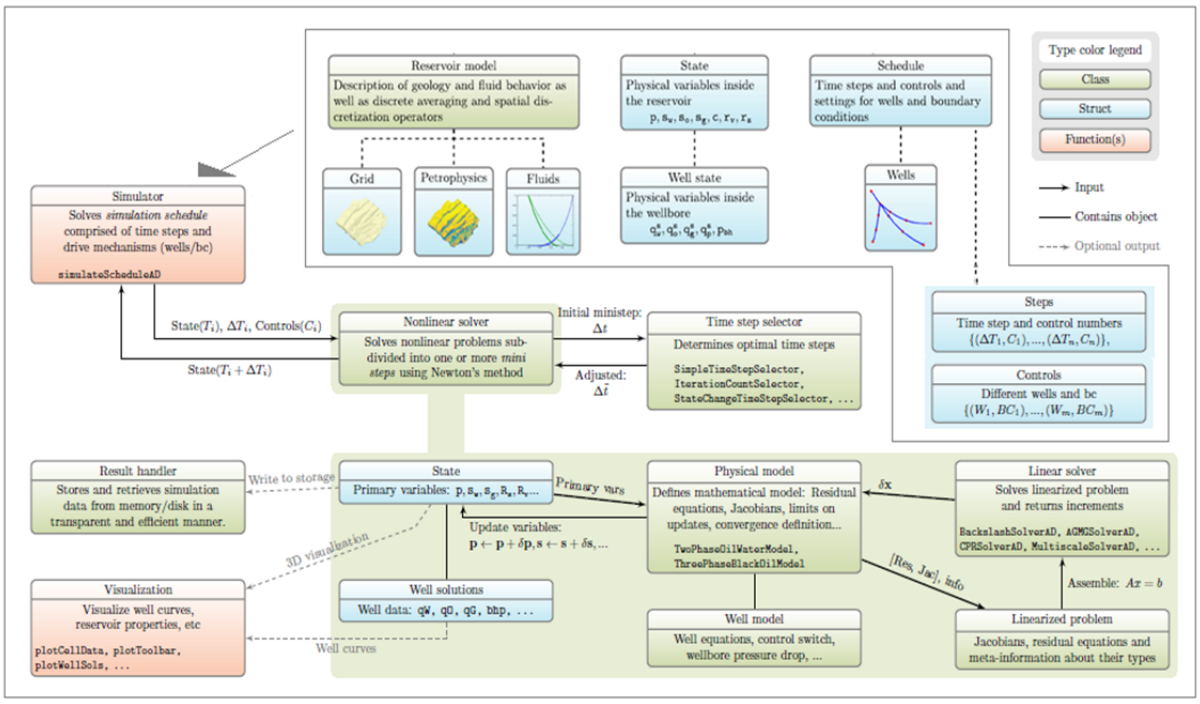


Figure 4. Overview of how components in the object-oriented AD framework are organized to implement a black-oil simulator. The different components are colorized by the type of the corresponding construction. (BAO et al, 2017)

In summary, at first, it is defined the initial reservoir model grid, petrophysical parameters and fluids information which together constitute the model reservoir class. Initial states of the reservoir are attributed, and wells and their schedules are set up, which provides time step and controls. All this information is passed as input to the simulator, called *simulateScheduleAD*. Within the simulator, there are classes and functions that will be used in order to determine the reservoir state and the wells solutions for each time step.

4 TPWL for Subsurface Flow

In this section, it is briefly explained the Trajectory Piecewise Linearization (TPWL) procedure for subsurface flow problems, specifically to two-phase flow presented in section 2.2. Additional information can be found at Cardoso (2009). The governing equations and discretizations are specified by Eqs .(6) e (7). The discrete system is given by Eq. (13) can be written here as:

$$g(x^{n+1},u^{n+1}) = F(x^{n+1}) + A(x^{n+1},x^{n}) + Q(x^{n+1},u^{n+1}) = 0\,. \tag{24}$$

Where g represents the residual vector, which one seeks to drive to zero, x designates the system states (pressure and saturation), u indicates the system controls (well BHPs), $F(x^{n+1}) = T^{n+1}x^{n+1}$ and $A\left(x^{n+1},x^{n}\right) = -D^{n+1}\left(x^{n+1}-x^{n}\right)$, and Q represent the flow, accumulation and source/sink terms, respectively.

As shown in section 2.2, Eq. (24) represents a nonlinear fully implicit system that is solved by an application Newton's method to drive g to zero.

4.1 Linearization of Governing Equations

Linearized models can be constructed through use of a Taylor series expansion around a previously converged and stored state and the respective control vector. These states are saved from preprocessing 'training' simulations. The key idea of TPWL is linearizing the model around

previously converged states (snapshots) saved during previously simulated training runs and, therefore, it replaces the simulation by a simple sequence of linear system solutions based on the physics of the problem.

Analyzing Eq.(24), it is known that g is a function of solution at time step (x^{n+1}), previous states vector (x^n) and the set of controls, which are specified (u^{n+1}). Thus, storing the converged state vectors (x^{i+1}, x^i) during a training simulation and the respective control vector u^{i+1}, Taylor series expansion of residual equation is applied, around these memorized states, as indicated by Eq.(25):

$$g\left(x^{n+1},x^{n},u^{n+1}\right)=g\left(x^{i+1},x^{i},u^{i+1}\right)+\left(\frac{\partial g^{i+1}}{\partial x^{i+1}}\right)\left(x^{n+1}-x^{i+1}\right)+\left(\frac{\partial g^{i+1}}{\partial x^{i}}\right)\left(x^{n}-x^{i}\right)+\left(\frac{\partial g^{i+1}}{\partial u^{i+1}}\right)\left(u^{n+1}-u^{i+1}\right) \tag{25}$$

Note that superscript i represents a time step of a training simulation, where the derivatives, states, and controls used were converged and saved, and superscript n represents the time steps of a new simulation, considering the same initial state, but other controls. (MACHADO, 2014)

From Eq.(25), the term $g\left(x^{i+1},x^{i},u^{i+1}\right)=0$, since the residue was previously converged during the training simulation. The derivatives in Eq. (25) can be simplified as shown in Eq.(26), (27) and(28).

$$\frac{\partial g^{i+1}}{\partial x^{i+1}}=J^{i+1}. \tag{26}$$

$$\frac{\partial g^{i+1}}{\partial x^{i}}=\frac{\partial A^{i+1}}{\partial x^{i}}. \tag{27}$$

$$\frac{\partial g^{i+1}}{\partial u^{i+1}}=\frac{\partial Q^{i+1}}{\partial u^{i+1}}. \tag{28}$$

The Eq.(26) is the Jacobian considered in Newton-Raphson method to solve the non-linear model. By definition, the Jacobian matrix is the derivative of the residual in relation to the state.

To write the other equations one we need analyze the residual as presented in Eq.(24) and consider the fact that the derivative with respect to the previous time $\left(\partial g^{i+1}/\partial x^{i}\right)$ only depends on the accumulation term $A\left(x^{i+1},x^{i}\right)$ and the derivative with respect to the control $\left(\partial g^{i+1}/\partial u^{i+1}\right)$ will appear only in the term source $Q\left(x^{i+1},u^{i+1}\right)$.This results in TPWL equation as expressed in Eq. (29).

$$J^{i+1}\left(x^{n+1}-x^{i+1}\right)\approx-\left[\frac{\partial A^{i+1}}{\partial x^{i}}\left(x^{n}-x^{i}\right)+\frac{\partial Q^{i+1}}{\partial u^{i+1}}\left(u^{n+1}-u^{i+1}\right)\right]. \tag{29}$$

Thus, for TPWL implementation, the following should be exported from the training simulation: x^{i+1}, x^{i}, u^{i+1}, J^{i+1}, $\partial A^{i+1}/\partial x^{i}$, $\partial Q^{i+1}/\partial u^{i}$.

Note that states, control and all derivatives are provided during the Jacobian computation in the full implicit formulation except the derivative $\partial A^{i+1}/\partial x^{i}$. However, it can be calculated in terms of accumulation derivative with some step-size correction by the expression in Eq. (7). This correction is obtained by multiplying $\partial A^{i}/\partial x^{i}$, available in the Jacobian calculation, by the ratio between the step size of the previous stored timestep Δt^{i} and the step size of the current one, Δt^{i+1}. (MACHADO, 2014)

$$\frac{\partial A^{i+1}}{\partial x^{i}}=-\frac{\Delta t^{i}}{\Delta t^{i+1}}\frac{\partial A^{i}}{\partial x^{i}}. \tag{30}$$

5 TPWL at MRST

After understanding the MRST and the TPWL technique, it is possible to join both of them to provide a numerical complexity reduction technique in a free and open access simulation tool. Due to the extension and complexity of the complete black oil simulator code in MRST, in this section, it will only briefly present some relevant points to the implementation of TPWL.

First, as presented in section 3 , one must provide the simulator (called *simulateScheduleAD*) with the initial state, schedule, and model. For the two-phase flow, the generated model, belonging to the *PysicalModel* class is called *TwoPhaseOilWaterModel*. This class incorporates the *equationsOilWater* function where discrete water and oil equations are created as expressed in section 3. In this function, the term referring to the accumulation is first created, then the terms referring to the wells and sink sources are incorporated and finally the term referring to the flow is added. It is important to highlight that as the variables are defined in the ADI classes, all operations performed result in the value determination and its derivation simultaneously. Thus, when writing the discrete equations, the parts of the Jacobian matrix are already being assembled which is composed of sub-blocks. From the assembled water and oil equations, the PysicalModel class calls the LinearizedModel function, where it is explicitly seen the Jacobian matrix and the residual vector. Newton's increment is obtained in the *BackslashSolverAD*. A summary of what is described is given in the code snippets present in Figure 5 and the Jacobian matrix sub-blocks are presented in Figure 6 .

```
%% Simulate base case
[wellSols, states,report] = simulateScheduleAD(state0, model, schedule);
```

```
% EQUATIONS OIL WATER ---------------------------------------------------

function [problem, state] = equationsOilWater(state0, state, model, dt,
drivingForces, varargin)
% Conservation of mass for water
water = (s.pv/dt).*( pvMult.*bW.*sW - pvMult0.*bW0.*sW0 );
% Conservation of mass for oil
oil = (s.pv/dt).*( pvMult.*bO.*sO - pvMult0.*bO0.*sO0 );
eqs = {water, oil};
% Add in and setup well equations
[eqs, names, types, state.wellSol] = model.insertWellEquations(eqs, names,
types, wellSol0, wellSol, wellVars, wellMap, p, mob, rho, {}, {}, dt,
opt);
% Add in fluxes
eqs{1} = eqs{1} + s.Div(bWvW);
eqs{2} = eqs{2} + s.Div(bOvO);
```

```
% LINEARIZED PROBLEM ----------------------------------------------------
classdef LinearizedProblem
function problem = assembleSystem(problem)
% Assemble the linear system from the individual Jacobians and residual
functions.
iseq = cellfun(@(x) ~isempty(x), problem.equations);
eqs = combineEquations(problem.equations{iseq});
% Jacobian Matrix
problem.A = eqs.jac{1};
% Residues
problem.b = -eqs.val;
```

```
% BACKSLASHSOLVERAD -----------------------------------------------------
classdef BackslashSolverAD < LinearSolverAD
    % Linear solver that calls standard MATLAB direct solver mldivide "\"
function [result, report] = solveLinearSystem(solver, A, b)
          result = A\b;
```

Figure 5. MRST code snippets (MRST, 2019)

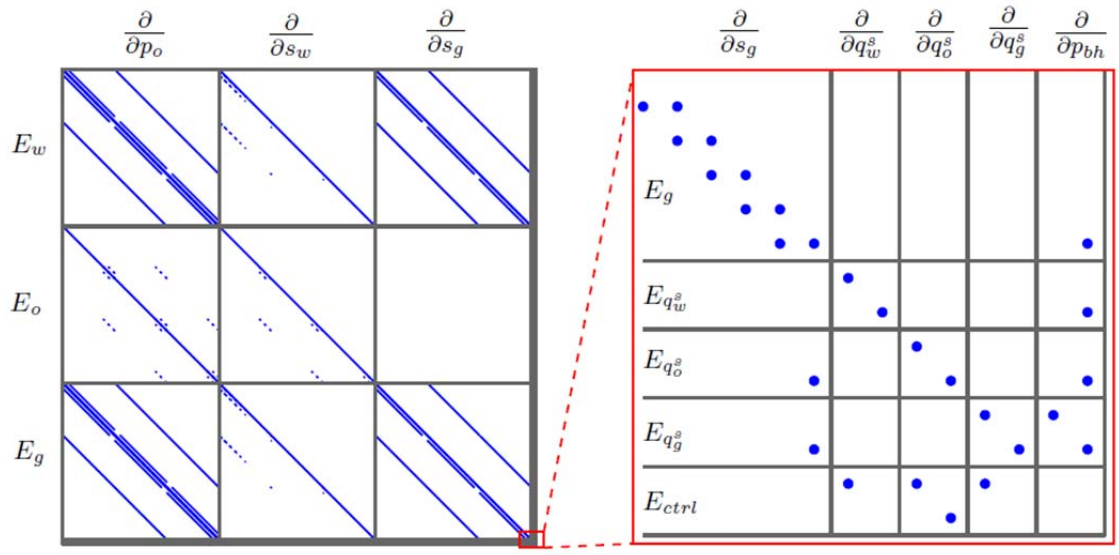


Figure 6. Illustration of the structure of the linearized black-oil equations and all the different sub-Jacobians that make up. (Krogstad, 2015)

After understanding MRST AD-Black-Oil Simulation, the Jacobian matrix was incorporated in the report, as well as the separate terms referring to the derivative of the accumulation term $\partial A^i/\partial x^i$ and derived from the source term in relation to the control $\partial Q^i/\partial u^i$. The matrix of the derivative of the accumulation term with respect to previous time ($\partial A^{i+1}/\partial x^i$) is obtained as described in section 3. In addition to the matrices, the training simulation states and schedule are saved, allowing access to time step information and controls. Provided with this information it is possible to implement TPWL, as presented in section 4.

6 NUMERICAL EXAMPLES

It will be shown, in this section, the application of the TPWL procedure to an illustrative reservoir simulation model in order to test the implementation of this technique and demonstrate its ability to provide accurate predictions for cases that differ from the initial training simulation.

The simple simulation model, shown in Figure 7, is a three-dimensional and contains a total of 4,500 grid blocks (with n_x=30, n_y=30 and n_z=5, where n_x, n_y and n_z designate the number of grid blocks in the corresponding coordinate direction). There is an injector well and a producer well, arranged in the configuration of quarter of five spot.

The mean permeability and porosity are constant, defined as 100 mD and 0.20 respectively. The initial oil and water saturations are 0.8 and 0.2 respectively and the residual oil (Sor) and water (Swr) saturations are 0.2. The relative permeability for the oil and water phases is defined by Corey model with Corey exponent value 2 for both. For oil we set ρ_o= 859 kg/m^3, μ_o = 5.0 cp, for water, ρ_w= 1014 kg/m^3, μ_w = 1.0 cp. The system is incompressible and capillary pressure effects are neglected.

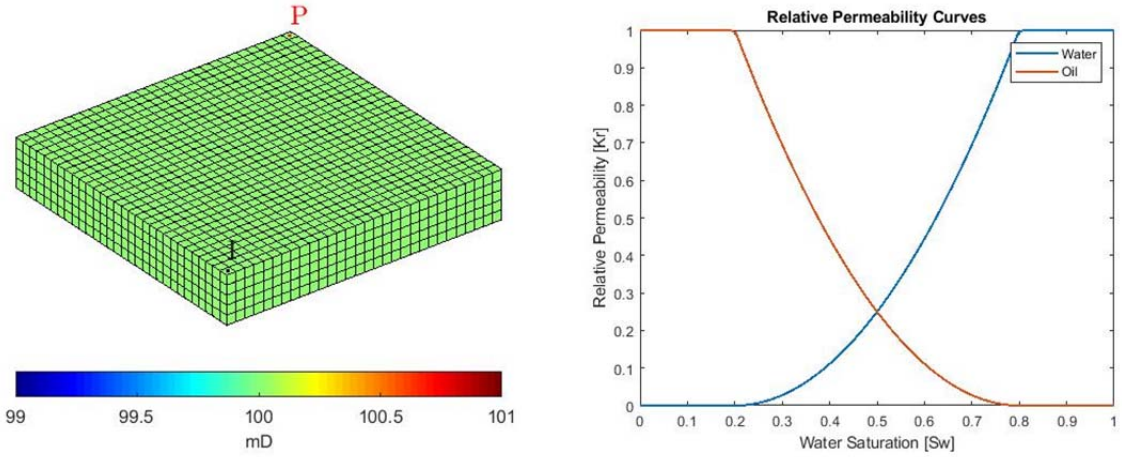


Figure 7. Simple and illustrative reservoir simulation model

The mean permeability and porosity are constant, defined as 100 mD and 0.20 respectively. The initial oil and water saturations are 0.8 and 0.2 respectively and the residual oil (Sor) and water (Swr) saturations are 0.2. The relative permeability for the oil and water phases is defined by Corey model with Corey exponent value 2 for both. For oil we set ρ_o= 859 kg/m^3, μ_o = 5.0 cp, for water, ρ_w= 1014 kg/m^3, μ_w = 1.0 cp. The system is incompressible and capillary pressure effects are neglected.

For the model described, a training run is performed using the high-fidelity model to generate the states and Jacobian matrices. Then, it was simulated the reservoir performance for a total of 5 years with a maximum time step of 30 days.

For training simulation injection well are prescribed to maintain constant bottom hole pressure (BHP) of 2,000 psia. For the production well, the BHPs are prescribed to follow schedule that way it varies every 365 days, randomly and independently, between 500 and 800 psia. Thus pressure and saturation snapshots and Jacobian matrices were recorded, allowing TPWL running.

First, the ability of the TPWL representation (Eq.(29)) to reproduce results from the training simulations is assessed by applying the BHP schedules used for training simulation. Figure 8 shows the result. Note figures that the TPWL results for production well, depicted by circles, are in essentially perfect agreement with the reference high-fidelity (MRST) solutions, depicted by solid curves.

Finally, it is possible to test cases that differ from the initial training simulation. Thus, the specification for producer BHP is changed randomly. Figure 9 shows that TPWL can also predict the results in this case. Note that the closer the new control is to the training simulation, the better the accuracy of the method. Defining the error by Eq. (31), the errors in oil rate and water production rate are $E_o^m = 0.0444$ and $E_w^m = 0.0267$, respectively.

$$E^m = \frac{1}{n_t Q_{hf}} \sum \left| Q_{tpwl} - Q_{hf} \right| . \tag{31}$$

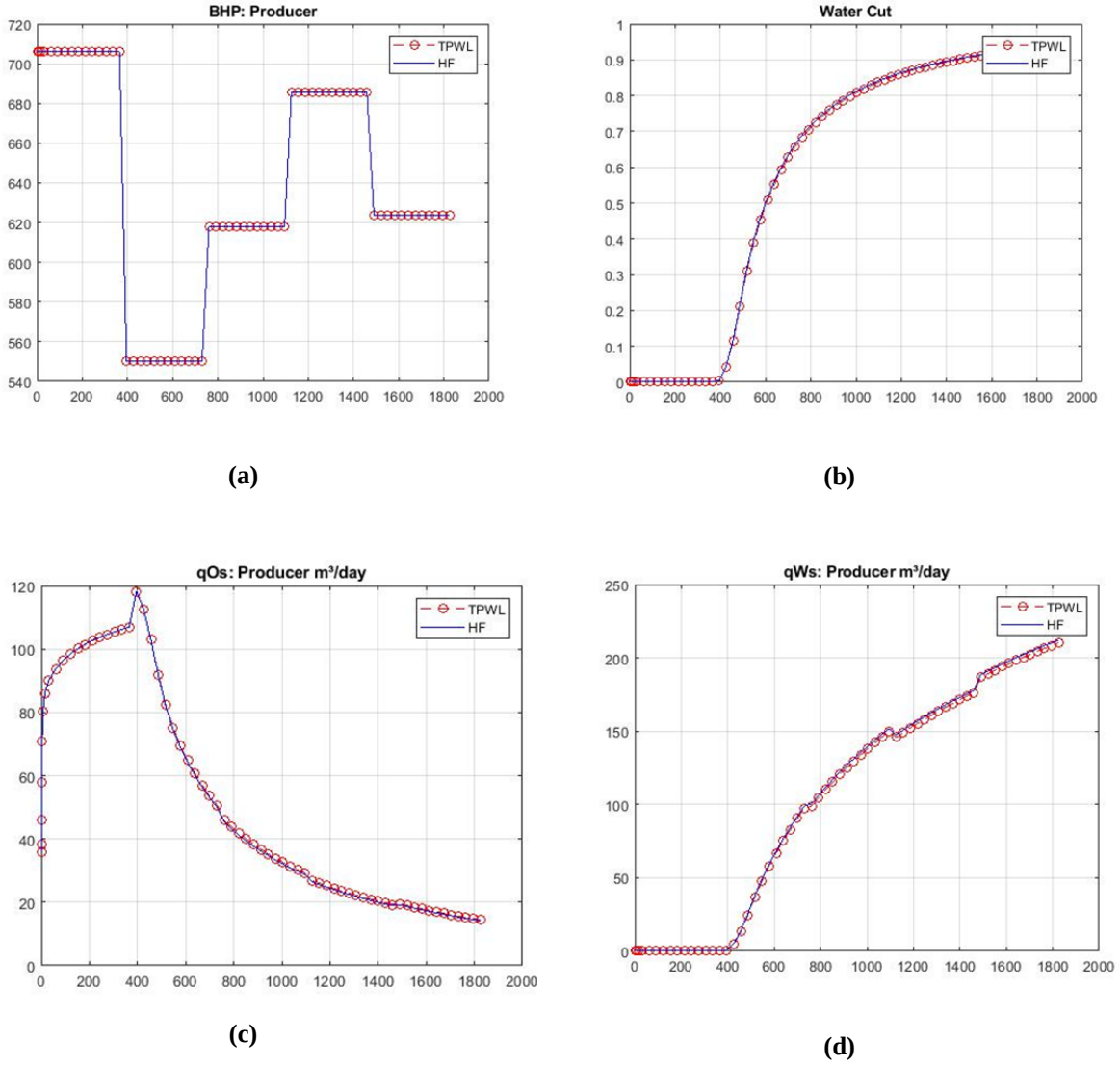


Figure 8. Production well resultes for training run and TPWL by applying the same BHP schedules

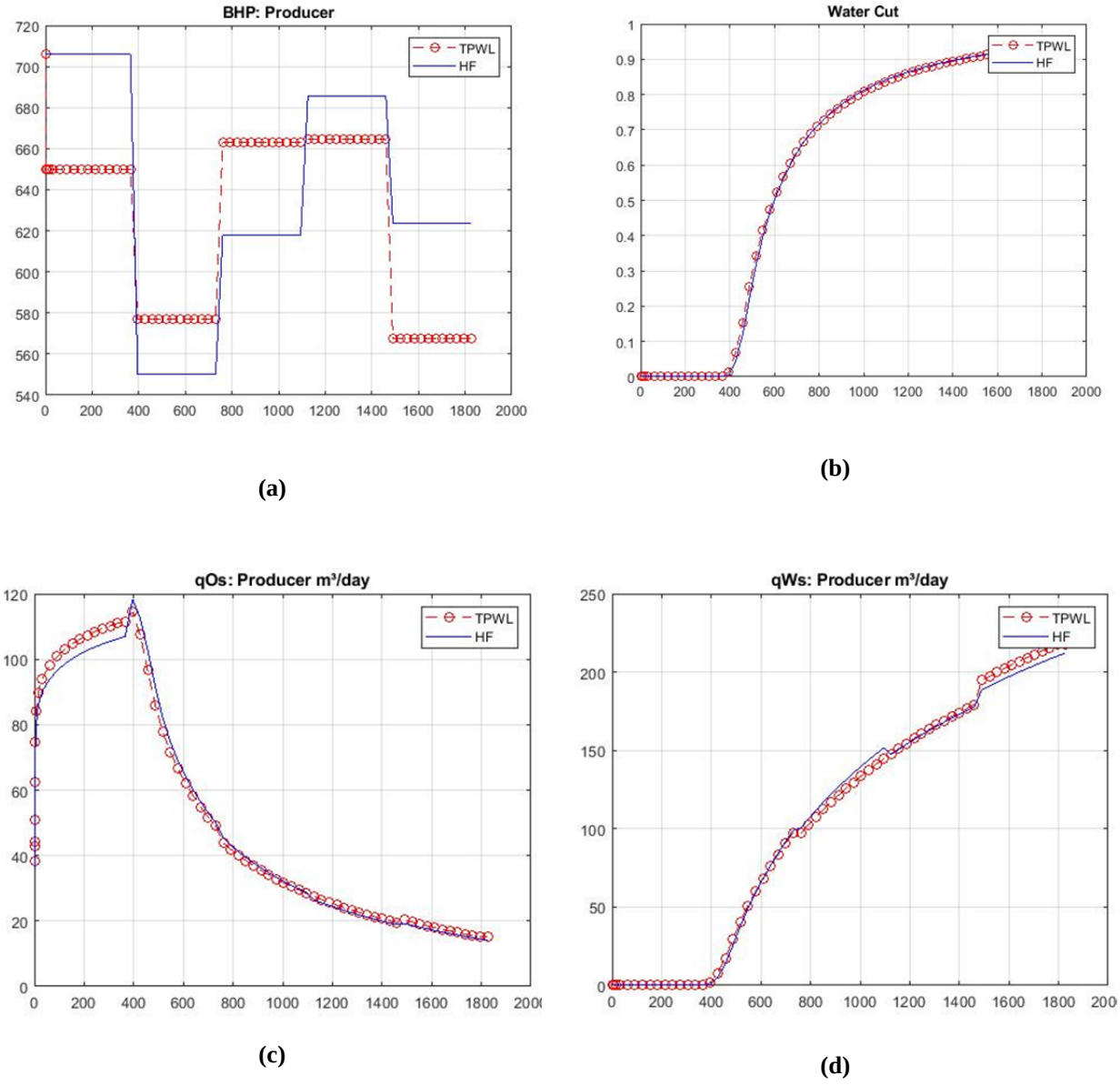


Figure 9. Production well results for training run and TPWL by applying different BHP schedules

7 CONCLUSIONS

In this present paper, it is presented a brief description of the implementation of the Trajectory Piecewise Linearization (TPWL) technique in Matlab Reservoir Simulation Toolbox – MRST.

The ad-black oil module with Automatic Differentiation (AD) allows the obtaining of Jacobian matrices in a simpler way for TPWL implementation.

The TPWL is shown to be accurate in the neighborhood of the training trajectory. As a continuation of this work, the method should be applied to more realistic models, in order to allow further conclusions about its accuracy. When combined with Proper Orthogonal Decomposition (POD), good speedups can be achieved by its application.

Acknowledgements

The authors acknowledge the financial support for this research given by National Council for Scientific and Technological Development (CNPq) and for supporting the development of this work.

References

[1] Bao, K.; Lie, K.-A; Møyner, O; Liu,M. Fully implicit simulation of polymer flooding with MRST. *Computational Geosciences*, [s.l.], v. 21, n. 5-6, p.1219-1244, 2017.
[2] Cardoso, M. A. Development and Application of Reduced-Order Modeling Procedures for Reservoir Simulation, 2009.
[3] K. Aziz and A. Settari. Fundamentals of Reservoir Simulation. Elsevier Applied Science Publishers, 1986.
[4] Krogstad; Lie, K–A; Møyner, O.; Nilsen. H. M.= Raynaud, X.; Skaflestad, B., MRST-AD – an Open-Source Framework for Rapid Prototyping and Evaluation of Reservoir Simulation Problems Stein SINTEF ICT, *Society of Petroleum Engineers*, 2015.
[5] Lie, K.-A., Krogstad, S., Ligaarden, I. S., Natvig, J. R., Nilsen, H., and Skaflestad, B. 2012. Open-source MATLAB implementation of consistent discretisations on complex grids. *Computational Geosciences*, 16:297–322. doi: 10.1007/s10596-011-9244-4.
[6] Lie, K.-A. 2014. An Introduction to Reservoir Simulation Using MATLAB. http://www.sintef.no/Projectweb/MRST/Publications/.
[7] LIE, K.-A. An Introduction to Reservoir Simulation Using MATLAB/ User Guide for the MATLAB Reservoir Simulation Toolbox (MRST). Oslo (Norway): SINTEF ICT, Departement of Applied Mathematics, 392 p., 2016.
[8] Machado, M. F. J. Aplicações de um modelo substituto de ordem reduzida a estudos de gerenciamento de reservatórios de petróleo, 2014.
[9] MRST, 2019. The MATLAB Reservoir Simulation Toolbox, version 2019a. In : https://www.sintef.no/projectweb/mrst/.
[10] Neidinger, R. 2010. Introduction to automatic differentiation and MATLAB object-oriented programming. SIAM Review, 52(3):545–563. doi:10.1137/080743627.