Compressible problem

To describe single-phase flow through a porous medium, we use the continuity equations:

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = q, \qquad \mathbf{v} = -\frac{\mathbf{K}}{\mu}(\nabla \mathbf{p} - \rho g \nabla z), \tag{1}$$

or

$$\frac{\partial(\rho\phi)}{\partial t} - \nabla \cdot \left(\frac{\rho \mathbf{K}}{\mu}(\nabla \mathbf{p} - \rho g \nabla z) \right) = q. \tag{2}$$

Where the primary unknown is the pressure $\mathbf{p}$ and the fluid $\rho = \rho(\mathbf{p})$ and rock $\phi = \phi(\mathbf{p})$ compressibilities can be pressure dependent. Rock compressibility is defined by:

$$c_r = \frac{1}{\phi}\frac{d\phi}{dp} = \frac{ln(\phi)}{dp},$$

If the rock compressibility is constant, the previous equation can be integrated as:

$$\phi(p) = \phi_0 e^{c_r(p-p_0)}.$$

Fluid compressibility is defined as:

$$c_f = \frac{1}{\rho}\frac{d\rho}{dp} = \frac{ln(\rho)}{dp}. \tag{3}$$

If the fluid compressibility is constant, the previous equation can be integrated as:

$$\rho(\mathbf{p}) = \rho_0 e^{c_f(\mathbf{p}-\mathbf{p}_0)}. \tag{4}$$

Well model
The relation between the bottom-hole pressure and the surface flow rate in a well is given by the the linear law:

$$q_0 = J(p_R - p_{bhp}),$$

where J is the productivity or injectivity index, in MRST

$$q_j = W_j(p_{r,j} - p_{bhp,j}).$$

Using implicit discretization, Equation (1) becomes:

$$\frac{(\phi\rho)^{n+1} - (\phi\rho)^n}{\Delta t^n} + \nabla \cdot (\rho \mathbf{v})^{n+1} = \mathbf{q}^n, \qquad \mathbf{v}^{n+1} = -\frac{\mathbf{K}}{\mu^{n+1}}[\nabla(\mathbf{p}^{n+1}) - g\rho^{n+1}\nabla(\mathbf{z})]. \tag{5}$$

If ϕ and ρ depend nonlinearly on $\mathbf{p}$ we have a nonlinear system of equations to be solved for each time step.
Assuming no gravity terms, Equation (1) can be rewritten as:

$$\frac{\phi(\mathbf{p}^{n+1})\rho(\mathbf{p}^{n+1}) - \phi(\mathbf{p}^n)\rho(\mathbf{p}^n)}{\Delta t^n} + \nabla \cdot (\rho \mathbf{v})^{n+1} = \mathbf{q}^n, \qquad \mathbf{v}^{n+1} = -\frac{\mathbf{K}}{\mu^{n+1}}\nabla(\mathbf{p}^{n+1}). \tag{6}$$

Or:

$$\frac{\phi(\mathbf{p}^{n+1})\rho(\mathbf{p}^{n+1}) - \phi(\mathbf{p}^{n})\rho(\mathbf{p}^{n})}{\Delta t^{n}} - \nabla \cdot (\rho(\mathbf{p}^{n+1})\frac{\mathbf{K}}{\mu^{n+1}}\nabla(\mathbf{p}^{n+1})) - \mathbf{q}^{n} = 0. \tag{7}$$

The latter system can be written in short vector form as:

$$\mathbf{F}(\mathbf{p}^{n+1}; \mathbf{p}^{n}) = 0, \tag{8}$$

with $\mathbf{p}^n$ the vector of unknown state variables at the time step n.
This non-linear system can be solved by Newton's method, the $(i+1)$-th iteration approximation is obtained from:

$$\frac{\partial \mathbf{F}(\mathbf{x}^i)}{\partial \mathbf{x}^i}\delta\mathbf{x}^i = -\mathbf{F}(\mathbf{x}^i), \qquad \mathbf{x}^{i+1} = \mathbf{x}^i + \delta\mathbf{x}^{i+1},$$

where $\mathbf{J}(\mathbf{x}^i) = \frac{\partial \mathbf{F}(\mathbf{x}^i)}{\partial \mathbf{x}^i}$ is the Jacobian matrix, and $\delta\mathbf{x}^{i+1}$ is the Newton update at iteration step $i+1$.

MRST

The MRST (MATLAB Reservoir Simulation Toolbox) allows the use of various grid types, which are stored using a general unstructured format, in which cells, faces, vertices's and connections between cells and faces are explicitly represented.
There is a wide range of structured and unstructured grids that can be constructed in MRST. For the structured grids, a pattern is chosen and repeated. The most typical structured grids are based on quadrilaterals in 2D and hexahedral 3D.

Cartesian grids.

The simplest structured grid is based in a square in 2D and a cube in 3D. to construct these grids in the MRST we need to specify the domain $[0\,Lx,\ 0\,Ly]$ represented as $[Lx\,Ly]$, and number of cells nx, ny, for cartesian grids the construction is in the next form:

$$\begin{aligned} G &= cartGrid([nx, ny], [Lx, Ly]) \qquad 2D, \\ G &= cartGrid([nx, ny], [Lx, Ly, Lz]) \qquad 3D. \end{aligned}$$

Rectilinear grids.

These are also called tensor grids, are rectilinear shapes (rectangles or parallelepipeds) not necessarily congruent to each other.
The grid structure contains three fields: cells, faces and nodes that specify the individual properties of each cell/face/vertex in the grid. The cell structure, G.cells, contains the following fields:

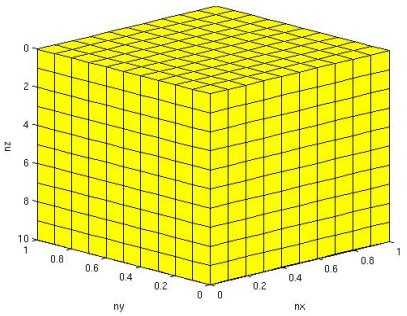

Figure 1: Cartesian grid in 3D.

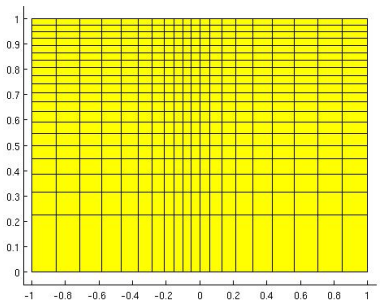

Figure 2: Rectangular grid in 2D.

- num: the number n_c of cells in the global grid.
- facePos: an indirection map of size [num+1,1] into the faces array. Specifi- cally, the face information of cell i is found in the submatrix faces(facePos(i) : facePos(i+1)−1, :) The number of faces of each cell may be computed using the state- ment diff(facePos) and the total number of faces is given as nf = facePos(end)−1.
- faces: an $n_f \times 3$ array that gives the global faces connected to a given cell. If faces(i,1)==j, the face with global number faces(i,2) is connected to cell number j. The last component, faces(i,3), is optional and can for certain types of grids contain a tag used to distinguish face directions: East, West, South, North, Bottom, Top.
- indexMap: an optional $n_c \times 1$ array that maps internal cell indices to external cell indices. For models with no inactive cells, indexMap equals 1 : nc . For cases with inactive cells, indexMap contains the indices of the active cells sorted in ascending order.

The face structure, G.faces, consists of the following mandatory fields:

- num: the number n_f of global faces in the grid.
- nodePos: an indirection map of size [num+1,1] into the nodes array. The node information of face i is found in the submatrix nodes(nodePos(i) : nodePos(i+1)−1, :) The number of nodes of each face may be computed using the state- ment diff(nodePos). Likewise, the total number of nodes is given as $nn = nodePos(end) - 1$.
- nodes: an $N_n \times 2$ array of vertices in the grid. If nodes(i,1)==j, the local vertex i is part of global face number j and corresponds to global vertex nodes(i,2). For each face the nodes are assumed to be oriented such that a right-hand rule determines the direction of the face normal. As for cells.faces, the first column of nodes is redundant and can be easily reconstructed.
- neighbors: an $n_f \times 2$ array of neighboring information. Global face i is shared by global cells neighbors(i,1) and neighbors(i,2). One of the entries in neighbors(i,:), but

not both, can be zero, to indicate that face i is an external face that belongs to only one cell (the nonzero entry).

To solve this problem with MRST, it is necessary to define the equations and some operators.
Assuming no-flow boundary conditions, the equations are defined only for the interior faces of the grid. The MRST function *computeTrans* computes the half transmissibilities associated with the two-point flux approximation (TPFA). Then, it is necessary to compute the harmonic average to obtain the face-transmissibilities. For neighboring cells, this is $T_{ij} = (T_{i,j}^{-1} + T_{j,i}^{-1})^{-1}$.
The divergence and gradients are defined as discrete functions. Taking N as the interior faces, the gradient of a vector $\mathbf{x}$ is defined as:

$$grad(\mathbf{x}) = \mathbf{x}(N(:,2)) - \mathbf{x}(N(:,1)) = \mathbf{C}\mathbf{x},$$

where $N(:,1)$ and $N(:,1)$ are the Neighboring faces. The divergence function is the negative transpose of the *grad* function.

$$div(\mathbf{x}) = -\mathbf{C}'\mathbf{x}.$$

To model flow through a porous medium with MRST, we need to define the equation for the pressures, the equations for the mass flow rates at the wells and the control equations for the wells. To define this equations, first we have to define the equations that relate the pressure with the density and the porosity. In this first cases, we will set the rock compressibility as zero, therefore, only the equation for the density is defined.

$$\boxed{\begin{array}{l} c = 1e-3/bars \\ rho_r = 850 * kilogram/meter^3 \\ rho = @(p) rho_r .* exp(c * p - p_r) \end{array}}$$

Darcy's law (6) for each face f in MRST is written as:

$$\mathbf{v}[f] = -\frac{\mathbf{T}[f]}{\mu} grad(p),$$

and the continuity equation for each cell c:

$$\frac{1}{\Delta t}\left((\phi(\mathbf{p})[c]\rho(\mathbf{p})[c])^{n+1} - (\phi(\mathbf{p})[c]\rho(\mathbf{p})[c])^{n}\right) + div(\rho_a \mathbf{v})[c] = \mathbf{0},$$

where $\rho_a$is the arithmetic average of the density. In MRST, the latter equations are implemented as anonymous functions of pressure:

$$v = @(p) \quad (T/mu) .* grad(p),$$

$$pressEq = @(p, p_0, dt) \quad (1/dt)(pv(p) .* rho(p) - pv(p0) .* rho(p0)) + div(avg(rho(p)) .* v(p)), \tag{9}$$

where p_0 is the pressure at the previous time step, and p is the pressure at the current time step.

The sources in this problem are wells, each cell where the well is connected to the reservoir is called connection. In MRST wc are the connection grid cells, WI are the well indices and dz is the depth relative to bottom-hole.
In presence of gravity, we need to consider the hydrostatic pressure drop (in this case we assume no gravity, so is not necessary). Assuming the fluid density in the well as constant, the pressure $\mathbf{p}[N_c(w)]$ in connection w of well $N_w(w)$ is given by:

$$\mathbf{p}_c[w] = \mathbf{p}_{bh}[N_w(w)] + g\Delta z[w]\rho(\mathbf{p}_{bh}[N_w(w)]),$$

where $\Delta z[w]$ is the vertical distance from bottom-hole to the connection,

$$p_{conn} = @(bhp) \quad bhp + g * dz. * rho(bhp).$$

The pressure at the well connection is related to the average pressure inside the grid cell by the Peaceman model. With this model, the mass flow rate at connection c is given by:

$$q_c[w] = \frac{\rho(\mathbf{p}[N_c(w)])}{\mu} WI[w](\mathbf{p}_c[w] - \mathbf{p}[N_c(w)]),$$

where $\mathbf{p}[N_c(w)]$ is the pressure in the cell $N_c(w)$ surrounding connection w,

$$q_{conn} = @(p, bhp) \quad WI. * (rho(p(wc))/mu). * (p_{conn}(bhp) - p(wc)).$$

The volumetric well-rate at surface conditions is obtained summing up all the mass well rates and dividing by the surface density:

$$rateEq = @(p, bhp, qS) \quad qS - sum(q_{conn}(p, bhp))/rhoS, \tag{10}$$

where the free variables are p, bhp and qS. To control the well we need an equation to specify the bottom-hole pressure (p_{bhp}):

$$ctrlEq = @(bhp) \quad bhp - p_{bhp}. \tag{11}$$

Once that the equations are defined for p (Equation (9)), bhp (Equation (11)) and qS (Equation (10)), the variables are initialized for automatic differentiation (see Appendix) with the initial values, which gives as result:

$\mathbf{p}_{ad}$= ADI Properties:	bhp_{ad}= ADI Properties:
val: $\mathbf{p}_{init}$	val: $p(wc)_{init}$
jac: $\{[\frac{\partial \mathbf{p}}{\partial \mathbf{p}} = \mathbf{I}]\ [\frac{\partial \mathbf{p}}{\partial p_{bh}} = 0]\ [\frac{\partial \mathbf{p}}{\partial qS} = 0]\}$	jac: $\{[\frac{\partial p_{bh}}{\partial \mathbf{p}} = \mathbf{0}]\ [\frac{\partial p_{bh}}{\partial p_{bh}} = 1]\ [\frac{\partial p_{bh}}{\partial qS} = 0]\}$

qS_{ad}= ADI Properties:
val: 0
jac: $\{[\frac{\partial qS}{\partial \mathbf{p}} = \mathbf{0}]\ [\frac{\partial qS}{\partial p_{bh}} = 0]\ [\frac{\partial qS}{\partial qS} = 1]\}$

These results are stored in a single matrix that contains sub-matrices with the jacobians of the variables. The jacobian matrix of the full system (eq) is the next one:

$$\begin{array}{|ccc|}\hline eq\{1\}.jac\{1\} = \frac{\partial \mathbf{p}}{\partial \mathbf{p}} & eq\{1\}.jac\{2\} = \frac{\partial \mathbf{p}}{\partial p_{bh}} & eq\{1\}.jac\{3\} = \frac{\partial \mathbf{p}}{\partial qS} \\ eq\{2\}.jac\{1\} = \frac{\partial p_{bh}}{\partial \mathbf{p}} & eq\{2\}.jac\{2\} = \frac{\partial p_{bh}}{\partial p_{bh}} & eq\{2\}.jac\{3\} = \frac{\partial p_{bh}}{\partial qS} \\ eq\{3\}.jac\{1\} = \frac{\partial qS}{\partial \mathbf{p}} & eq\{3\}.jac\{2\} = \frac{\partial qS}{\partial p_{bh}} & eq\{3\}.jac\{3\} = \frac{\partial qS}{\partial qS} \\ \hline \end{array}$$

From this matrix, we can obtain the jacobian and the residual for the Newton-Raphson iteration. Once the jacobian is obtained, the solution can be computed.

$$\begin{array}{|c|}\hline \mathbf{J} = eq.jac\{1\} \ \%\text{Jacobian} \\ res = eq.val \ \%\text{residual} \\ upd = -(\mathbf{J}/res) \ \%\text{Newton update} \\ \hline \end{array}$$

The algorithm is presented below:

Algorithm 1

$numSteps = 52$ % number of time−steps
$totTime = 365 * day$ % total simulation time
$dt = totTime$ / numSteps % constant time step
$tol = 1e-5$ % Newton tolerance
$maxits = 10$ % max number of Newton its

$t = 0$
$step = 0$ %initialize time and steps
$[p_{ad}, bhp_{ad}, qS_{ad}] = initVariableADI(p_{init}, p(wc)_{init}, 0)$ %initialize variables

while t < totTime
$t = t + dt$
$step = step + 1$
% Newton loop
$resNorm = 1e99$
$p0 = double(p_{ad})$ % Previous step pressure
$nit = 0$

while (resNorm > tol) && (nit <= maxits)
% Newton update
$\mathbf{J} = eq.jac\{1\}$ %Jacobian
$res = eq.val$ %residual
$upd = -(\mathbf{J}/res)$ [1] %Newton update, the solution of this system is obtained with ICCG or DICCG
% Update variables
$p_{ad}.val + upd(pIx)$
$bhp_{ad}.val + upd(bhpIx)$
$qS_{ad}.val + upd(qsIx)$
resNorm = norm(res) [2]
$nit = nit + 1$ [3]
end
end

[1] Note that the update is computed without taking into account the norm of the residual ($resNorm$) for each time step . If the solution is reached in the previous time

step, it won't be taken into account and the first update will be computed. [2] Only when the variables are already updated, the residual is computed. At this point, the number of iterations for the NR method is 0, it is only updated [3] after the first computation of the residual, when a first update has already be done.

For the solver the two algorithms are presented below:

Algorithm 2. ICCG

Select x_0 % It is random for the first time step
% and the solution of the previous time step for the rest
Compute $r_0 = b - Ax_0$ % the residual
Solve $Mr_0 = r_0$ % in this case $M = l^T l$ then we need to solve $r_0 = l \backslash r_0$
Solve $M^{-T} r_0 = p_0$ % in this case $M = l^T l$ then we need to solve $p_0 = l^T \backslash r_0$
$nb = norm(M^{-1}b)$ % for the stopping criteria, we need to compute the
2-norm of the preconditioned right hand side
$tol = 1e-7$ % Tolerance

for $j = 0, ...,$ until convergence
$w_j = Ap_j$;
$\alpha_j = \frac{(r_j, r_j)}{(p_j, w_j)}$
$x_{j+1} = x_j + \alpha_j p_j$
$r_{j+1} = r_j - \alpha_j * (l \backslash w_j)$
$\beta_j = \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)}$
$p_{j+1} = l^T \backslash r_{j+1} + \beta_j p_j$
$rr = \frac{(r_{j+1}, r_{j+1})}{nb}$ %Compute the relative residual
and if it is smaller than the tolerance stop
$\frac{||l^{-1} r||_2}{||l^{-1} b||_2}$
end

Algorithm 3. DICCG

Select x_0 % It is random for the first time step
% and the solution of the previous time step for the rest
Compute $r_0 = b - Ax_0$
Compute $r_0 = (I - AQ)r_0 = r0 - AZE^{-1}Z^T r_0$ % preconditioned residual
Solve $Mr_0 = r_0$ % in this case $M = l^T l$ then we need to solve $r_0 = l \backslash r_0$
Solve $M^{-T} r_0 = p_0$ % in this case $M = l^T l$ then we need to solve $p_0 = l^T \backslash r_0$
Compute $b = (I - AQ)b = b - AZE^{-1}Z^T b$ % preconditioned right hand side
$nb = norm(M^{-1}b)$ % for the stopping criteria, we need to compute the
2-norm of the preconditioned deflated right hand side
$tol = 1e - 7$ % Tolerance

for $j = 0, ...,$ until convergence
$w_j = Ap_j - AZE^{-1}Z^T Ap_j$ % deflated projection vector $P(Ap_j)$
$\alpha_j = \frac{(r_j, r_j)}{(p_j, w_j)}$
$x_{j+1} = x_j + \alpha_j p_j$
$r_{j+1} = r_j - \alpha_j * (l \backslash w_j)$
$\beta_j = \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)}$
$p_{j+1} = l^T \backslash r_{j+1} + \beta_j p_j$
$rr = \frac{(r_{j+1}, r_{j+1})}{nb}$ %Compute the relative residual
and if it is smaller than the tolerance stop
$\frac{||l^{-1}r||_2}{||l^{-1}b||_2}$
end

A first question is if it is OK to compute the update every time, or if the residual is small enough we can just use the previous time step solution. To investigate this, we perform some experiments, in the first one the code is as above, in the second one, the norm of the residual is computed and if it is greater than the defined tolerance, the update will be computed, otherwise, it will stop and we will use the previous time step as solution.

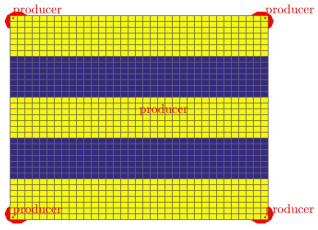

Figure 3: Heterogeneous permeability.

No deflation vectors, incompressible

The first set of experiments is an incompressible model, the fluid's compressibility is set as zero. We study a layered heterogeneous permeability problem with the first set of layers of permeability $\sigma_1 = 3mD$, and the second set of layers with permeability $\sigma_2 = 30mD$ (see Figure 3). We solve the linear system with the ICCG method

Results

In Figure 5, we observe that the number of ICCG iterations for the first NR- iteration decreases from a value of 55 to around 15 for the rest of the time steps. However, in Figure 4, we observe that the final solution does not change with time, as expected because it is an incompressible model. In Table 1, we also observe that the residual of the NR iterations is decreasing for the original code, but if we don't perform the ICCG iterations it remains the same (an enough accurate value). In Figure 7, we observe that we only need to compute the solution for the first time step if the previous residual is taken into account, as expected if the solution has already been achieved.

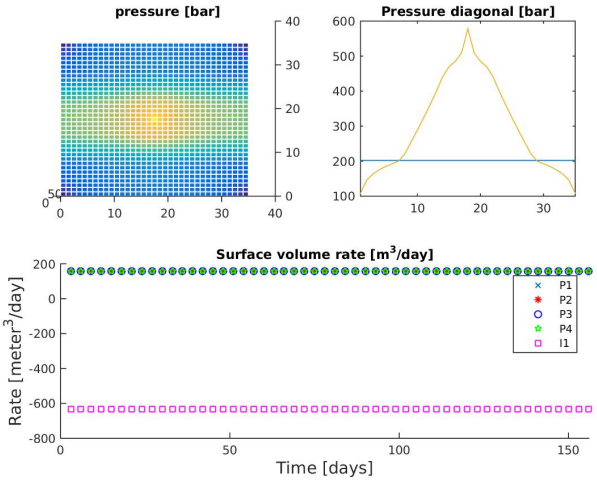

Figure 4: Solution, well fluxes, original code

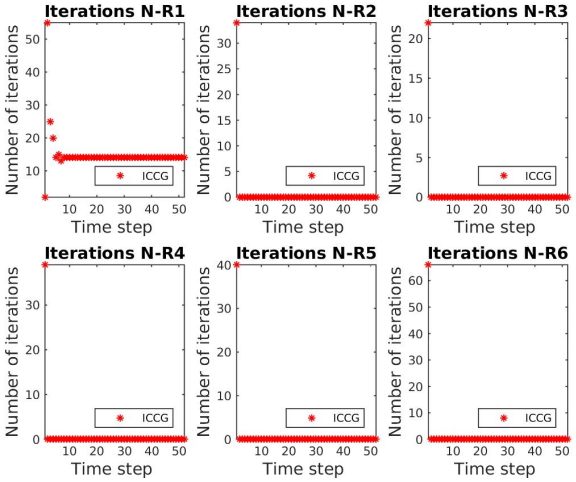

Figure 5: Number of iterations ICCG only, original code

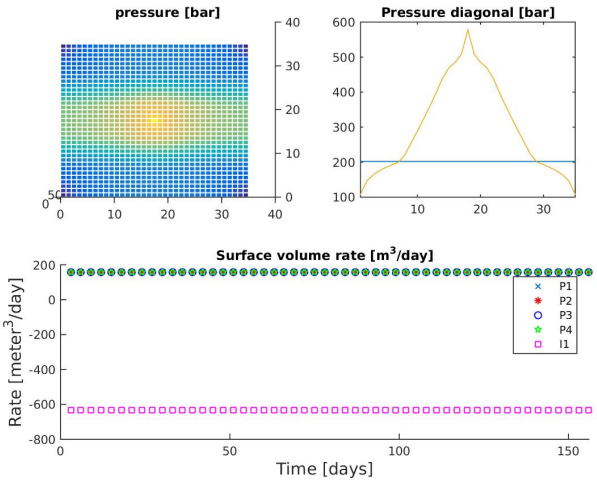

Figure 6: Solution, well fluxes, modified code

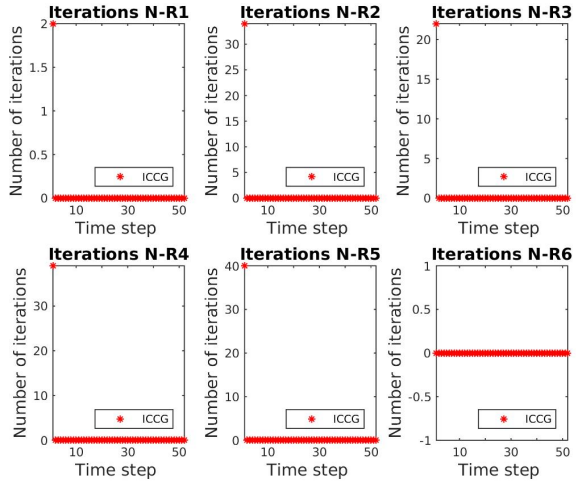

Figure 7: Number of iterations ICCG only, modified code

Time step 1: Time 0.00− > 3.00 days	Time step 1: Time 0.00− > 3.00 days
Iteration 1: Res = 4.4724e+07	Iteration 1: Res = 4.4724e+07
Iteration 2: Res = 8.4855e+01	Iteration 2: Res = 8.4855e+01
Iteration 3: Res = 7.5588e+01	Iteration 3: Res = 7.5588e+01
Iteration 4: Res = 2.0917e-04	Iteration 4: Res = 2.0917e-04
Iteration 5: Res = 1.1755e-04	Iteration 5: Res = 1.1755e-04
Iteration 6: Res = 9.2067e-10	Iteration 6: Res = 9.2067e-10
Time step 2: Time 3.00− > 6.00 days	Time step 2: Time 3.00− > 6.00 days
Iteration 1: Res = 6.3218e-12	Iteration 1: Res = 9.2067e-10
Time step 3: Time 6.00− > 9.00 days	Time step 3: Time 6.00− > 9.00 days
Iteration 1: Res = 4.2028e-14	Iteration 1: Res = 9.2067e-10
Time step 4: Time 9.00− > 12.00 days	Time step 4: Time 9.00− > 12.00 days
Iteration 1: Res = 4.5151e-14	Iteration 1: Res = 9.2067e-10

Table 1: NR-residual, left: Original code, right: if the residual is small enough, no ICCG iteration is performed

No deflation vectors, compressible

As a second set of experiments, we use compressible model, with a fluid's compressibility of $c = 1e - 3$.
In Figure 21 we observe that if we use the original code, the number of iterations for the first NR iteration is around 30 for the first forty time steps, after this, it decreases but it is always greater than zero. When we check the norm of the residual before using the linear solver (see Figure 23) we can see that after approximately the 20th time step no linear solver-iterations are needed. For the rest of the NR iterations a similar behavior is observed. In Table 2, we observe that the residual of the NR procedure is decreasing each time step for the original code, contrary to the case of the modified code where it remains the same. We also observe that the solution is the same in both cases (see Figures 20 and 22).

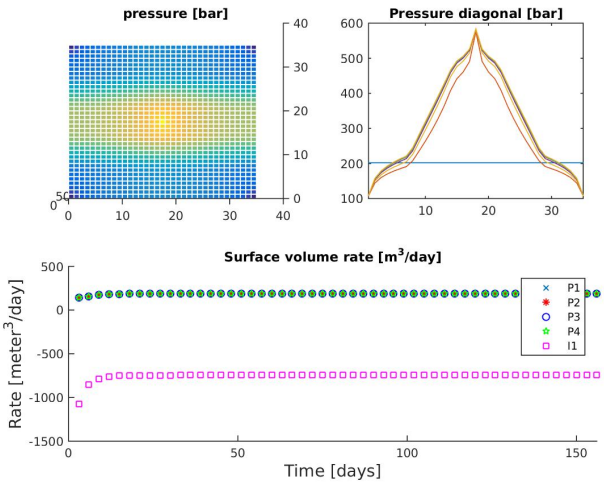

Figure 8: Solution, well fluxes

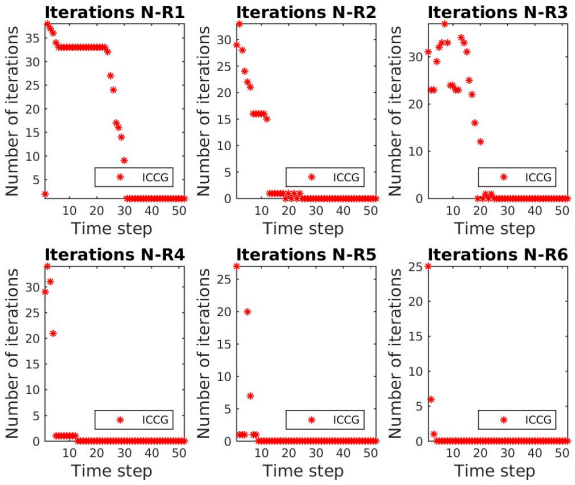

Figure 9: Number of iterations ICCG only

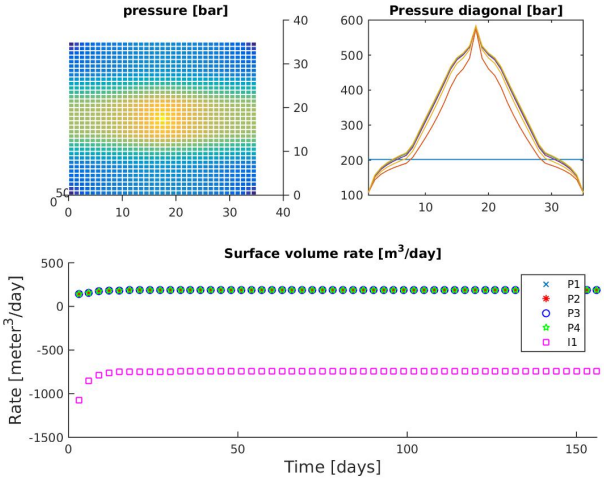

Figure 10: Solution, well fluxes

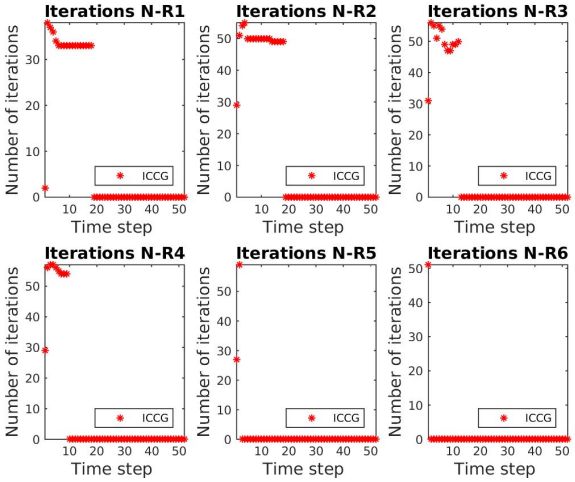

Figure 11: Number of iterations ICCG only

Time step 18: Time 51.00− > 54.00 days	Time step 18: Time 51.00− > 54.00 days
Iteration 1: Res = 1.1592e-07	Iteration 1: Res = 1.1585e-07
Iteration 2: Res = 8.5458e-11	Iteration 2: Res = 8.3023e-11
Time step 19: Time 54.00− > 57.00 days	Time step 19: Time 54.00− > 57.00 days
Iteration 1: Res = 4.8412e-08	Iteration 1: Res = 4.8381e-08
Time step 20: Time 57.00− > 60.00 days	Time step 20: Time 57.00− > 60.00 days
Iteration 1: Res = 2.0221e-08	Iteration 1: Res = 4.8381e-08
Time step 21: Time 60.00− > 63.00 days	Time step 21: Time 60.00− > 63.00 days
Iteration 1: Res = 8.4444e-09	Iteration 1: Res = 4.8381e-08
Time step 22: Time 63.00− > 66.00 days	Time step 22: Time 63.00− > 66.00 days
Iteration 1: Res = 3.5266e-09	Iteration 1: Res = 4.8381e-08
Time step 23: Time 66.00− > 69.00 days	Time step 23: Time 66.00− > 69.00 days
Iteration 1: Res = 1.4727e-09	Iteration 1: Res = 4.8381e-08
Time step 24: Time 69.00− > 72.00 days	Time step 24: Time 69.00− > 72.00 days
Iteration 1: Res = 6.1503e-10	Iteration 1: Res = 4.8381e-08

Table 2: NR-residual, left: Original code, right: if the residual is small enough, no ICCG iteration is performed

Deflation: 5 deflation vectors, compressible

The previous experiments are performed with ICCG. For this experiment, 5 snapshots are obtained with ICCG. SVD is computed for this set of snapshots and the eigenvectors corresponding to the 3 larger eigenvalues are used as deflation vectors (if we used 5 we have problems with the matrix $\mathbf{E}$).
As in the previous case, we computed the solutions with the original code and then we computed the residual of the NR iteration before the linear solver, if the required value of the residual is already reached, no further solution is computed.
Results
In Figure 25 we observe that the number of iterations is considerably reduced when we change the code for the first NR iteration and slightly reduced for the other NR iterations. With respect to the ICCG solver, we observe a reduction in the number of iterations of the linear solver when we use DICCG for the third and fourth NR-iteration. However, we observe that for the first and second NR iterations there is no change with respect to the ICCG solution. For the sixth NR iteration, case 1 (original code) no further ICCG solution is required, but we still need DICCG iterations. The same happens for the 5th NR iteration for the second case.
The previous results leads to the question if it is OK the stopping criteria of the solver (See Algorithm 2 and 3). Next section is to investigate this question.

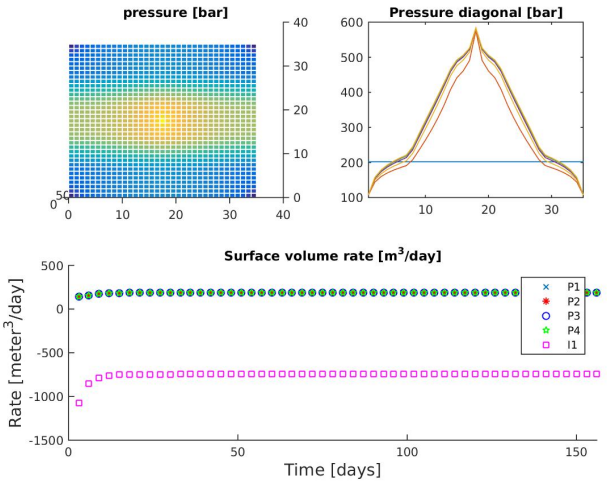

Figure 12: Solution, well fluxes

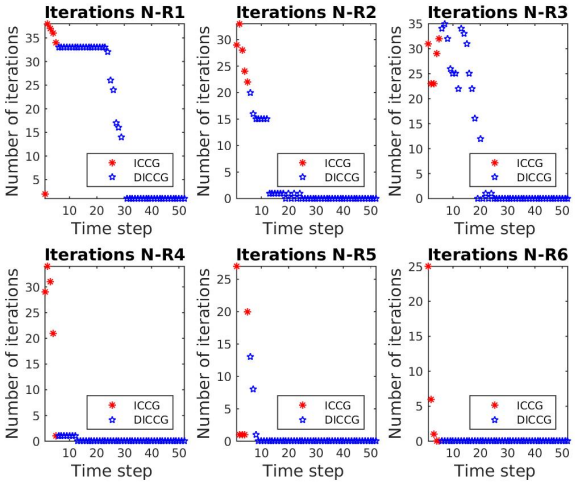

Figure 13: Number of iterations ICCG and DICCG

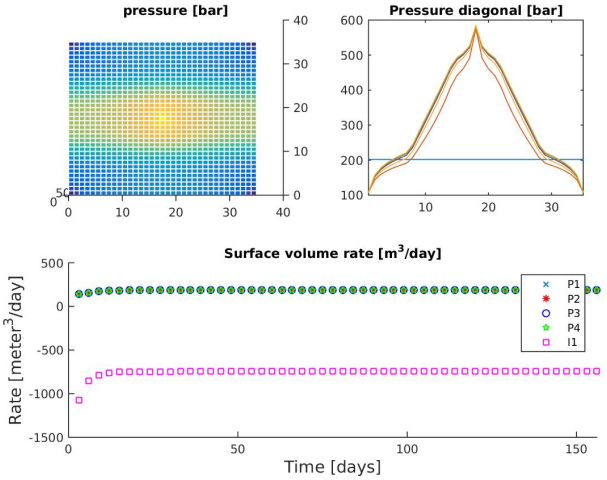

Figure 14: Solution, well fluxes

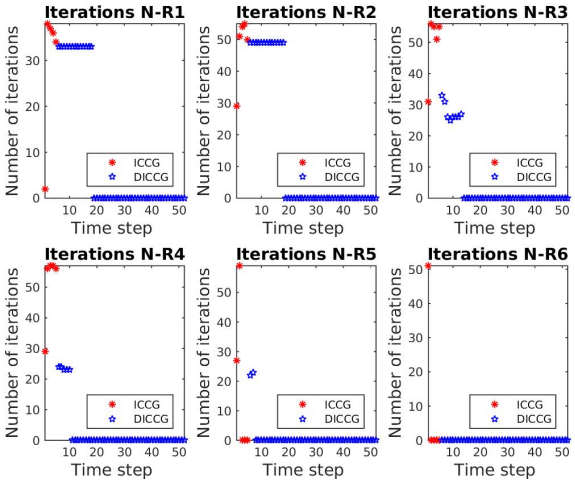

Figure 15: Number of iterations ICCG and DICCG

Time step 5: Time 12.00− > 15.00 days
Iteration 1: Res = 9.8962e-03
Iteration 2: Res = 4.2711e-05
Iteration 3: Res = 1.5146e-08
Time step 6: Time 15.00− > 18.00 days
Iteration 1: Res = 4.1194e-03
Iteration 2: Res = 7.7991e-06
Iteration 3: Res = 2.7457e-09
Time step 7: Time 18.00− > 21.00 days
Iteration 1: Res = 1.7192e-03
Iteration 2: Res = 1.7729e-06
Iteration 3: Res = 4.5272e-10
Time step 8: Time 21.00− > 24.00 days
Iteration 1: Res = 7.1789e-04
Iteration 2: Res = 5.7413e-07
Iteration 3: Res = 1.5161e-10

Time step 5: Time 12.00− > 15.00 days
Iteration 1: Res = 9.8962e-03
Iteration 2: Res = 4.2711e-05
Iteration 3: Res = 1.5146e-08
Time step 6: Time 15.00− > 18.00 days
Iteration 1: Res = 4.1194e-03
Iteration 2: Res = 5.9423e-04
Iteration 3: Res = 4.0076e-05
Iteration 4: Res = 5.4688e-06
Iteration 5: Res = 4.0496e-07
Iteration 6: Res = 3.3640e-08
Time step 7: Time 18.00− > 21.00 days
Iteration 1: Res = 1.7192e-03
Iteration 2: Res = 2.5263e-04
Iteration 3: Res = 1.7014e-05
Iteration 4: Res = 2.3323e-06
Iteration 5: Res = 1.7251e-07
Iteration 6: Res = 1.4387e-08
Time step 8: Time 21.00− > 24.00 days
Iteration 1: Res = 7.1789e-04
Iteration 2: Res = 1.0636e-04
Iteration 3: Res = 7.1587e-06
Iteration 4: Res = 9.8312e-07
Iteration 5: Res = 7.2696e-08

Table 3: NR-residual, left: ICCG iterations only, right: ICCG for the time step 5, DICCG for the others. Original code

Stopping criteria for the solvers

The Algorithms for the linear solvers are presented above. The stopping criterium used for this solvers is the residual divided by the right hand side, for the ICCG method is:

$$\frac{||l^{-1}r_{j+1}||_2}{||l^{-1}b||_2},$$

and for the DICCG is:

$$\frac{||l^{-1}\hat{r}_{j+1}||_2}{||l^{-1}b||_2},$$

where $\hat{r}$ is the preconditioned residual.
For the next series of experiments we use as stopping criterium the difference between the solution obtained for the iteration j and the solution obtained for iteration $j+1$ divided by the solution obtained for iteration j, for the ICCCG method:

$$\frac{||x_{j+1}-x_j||_2}{||x_j||_2},$$

and for the DICCG method:

$$\frac{||\hat{x}_{j+1}-\hat{x}_j||_2}{||\hat{x}_j||_2},$$

where $\hat{x}$ is the preconditioned solution. We solve the same problems as before, we just change the stopping criterium. The results are presented below.

No deflation vectors, incompressible 2

In Figure 17 we observe that the number of ICCG iterations for the first NR- iteration is around 30. However, in Figure 16, we observe that the final solution does not change with time, as expected because is an incompressible model. In Table 4, we also observe that the residual of the NR iterations is decreasing for the original code, but if we don't perform the ICCG iteration it remains the same (an enough accurate value). In Figure 19, we observe that we only need 1 NR iteration if the previous residual is taken into account, as expected if the solution has already been achieved.

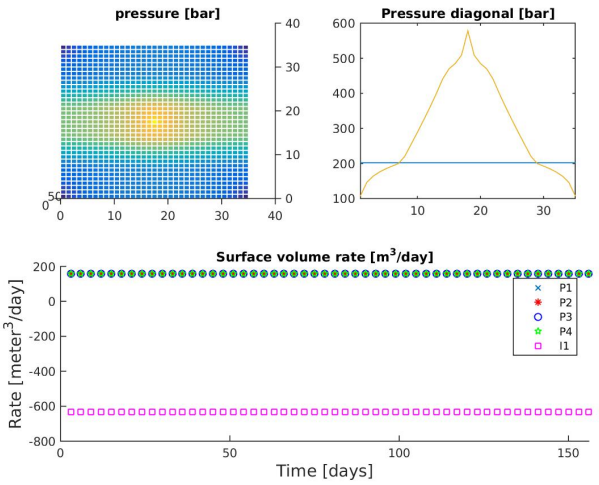

Figure 16: Solution, well fluxes

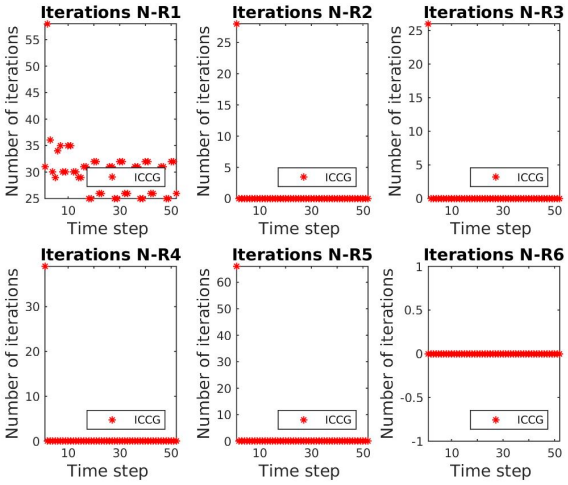

Figure 17: Number of iterations ICCG only

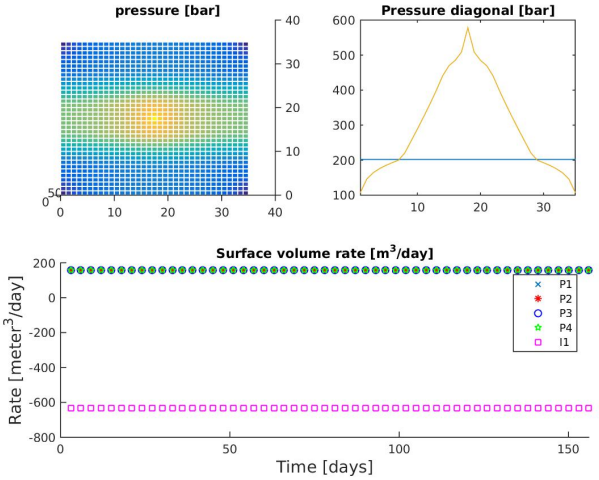

Figure 18: Solution, well fluxes

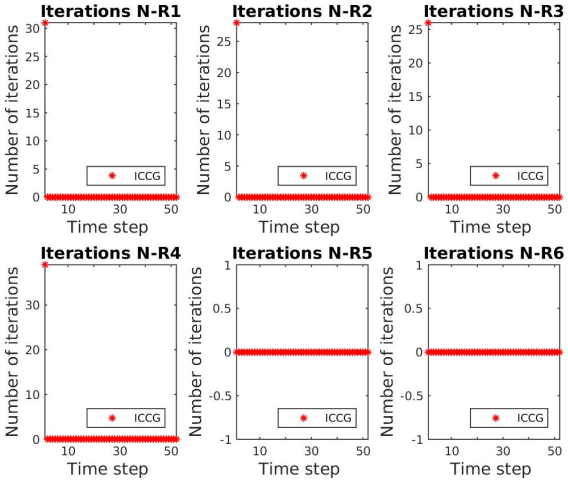

Figure 19: Number of iterations ICCG only

No deflation vectors, compressible 2

In Figure 21 we observe that if we use the original code the number of iterations for the first NR iteration is between 25 and 30. For the case when we check the norm of the residual before the linear solver (see Figure 23) after approximately the 20th time step, there is no further computation of the solution. For the other NR iterations a similar behavior is observed. We also observe that the solution is the same in both cases (see Figures 20 and 22).

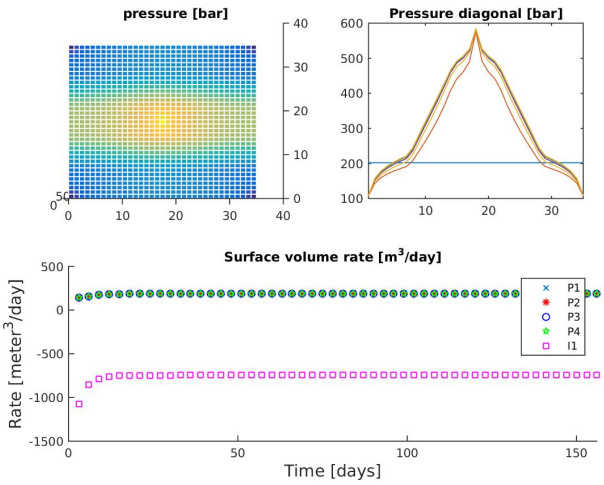

Figure 20: Solution, well fluxes

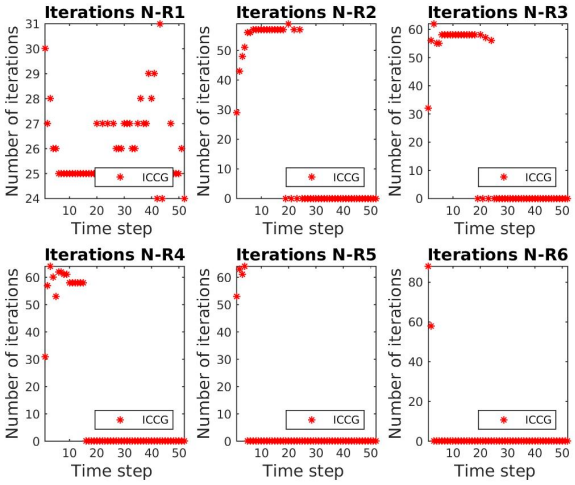

Figure 21: Number of iterations ICCG only

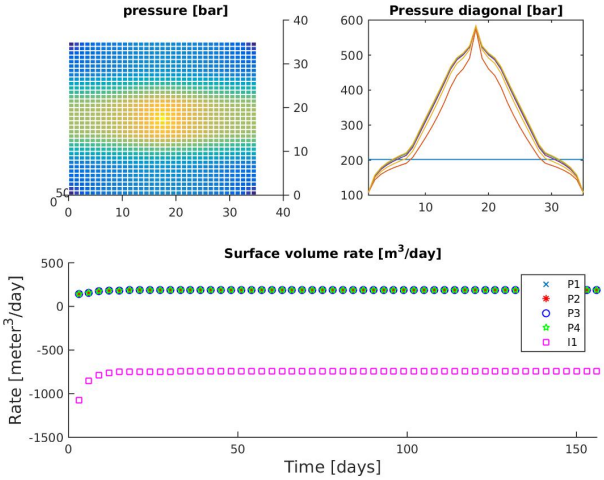

Figure 22: Solution, well fluxes

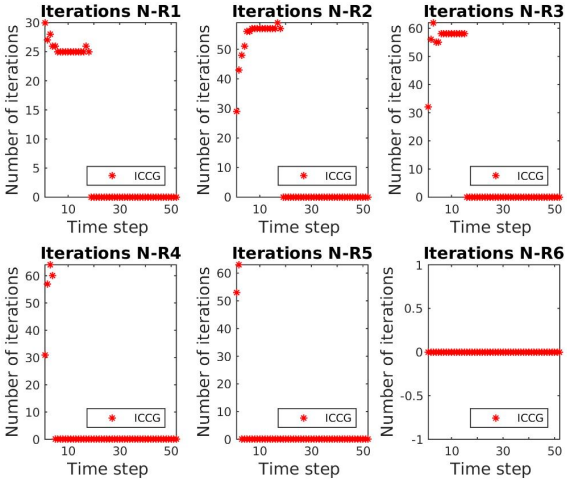

Figure 23: Number of iterations ICCG only

Deflation: 5 deflation vectors, compressible 2

In Figure 25 we observe that the number of iterations is considerably reduced when we change the code for the first NR iteration and slightly reduced for the other NR iterations. With respect to the ICCG solver, we observe a reduction in the number of iterations of the linear solver when we use DICCG for the second and third NR-iteration. However, we observe that for the first and second NR iterations there is no change with respect to the ICCG solver. For the sixth NR iteration, case 1 (original code) any further ICCG solution is required, but we still need DICCG iterations. The same happens for the 6th NR iteration for the second case.

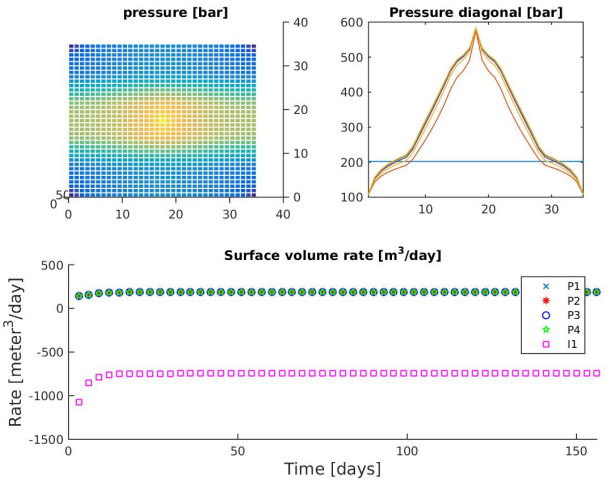

Figure 24: Solution, well fluxes

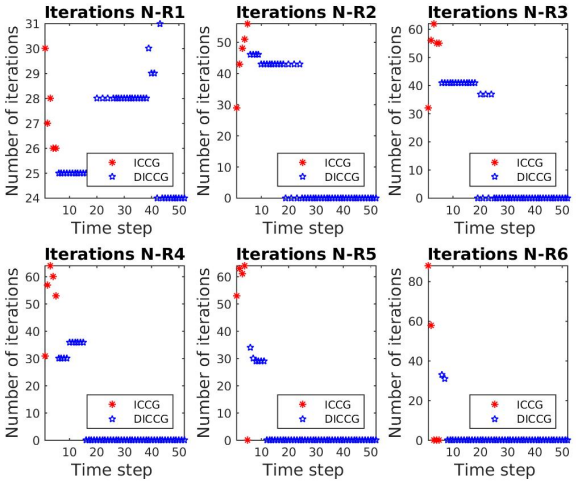

Figure 25: Number of iterations ICCG and DICCG

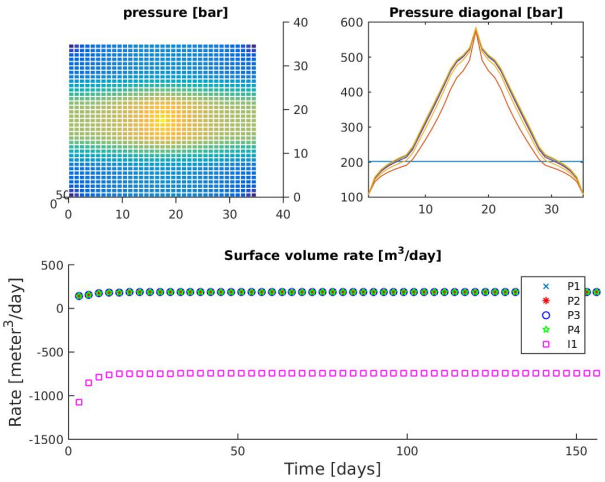

Figure 26: Solution, well fluxes

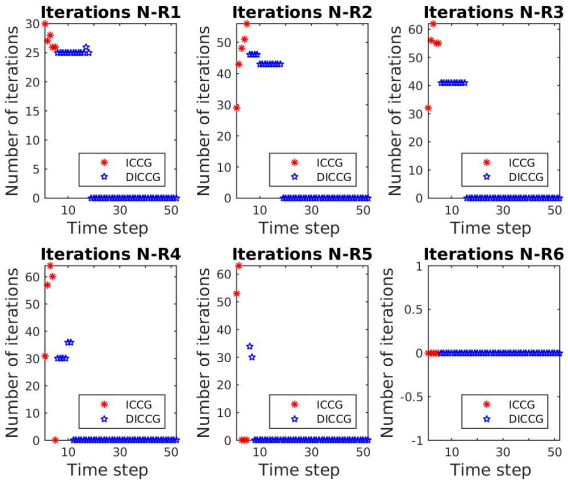

Figure 27: Number of iterations ICCG and DICCG

Appendix 1. Automatic differentiation

The automatic differentiation library of MRST uses a list of matrices that represent the derivatives with respect to different variables (pressure, saturation, bottom-hole-pressures, etc.), which constitute sub-blocks in the Jacobian matrix of the full system. If we want to compute the derivative of a function, eg. $\mathbf{f}$, with respect to the variables x and y, first we need to initialize the variables with the initial values: $x = x_0$, $y = y_0$.

$$[x, y] = initVariableADI(x_0, y_0);$$

then we define the function in terms of the variables

$$\mathbf{f} = \mathbf{f}(x, y).$$

Once the function is defined in terms of ADI variables, it becomes also an ADI variable and therefore, we have now three ADI variables. These variables contain two values, *val* with the value of the variables or the functions evaluated at the given values, and *jac* with the derivative with respect to the other variables. For the function, this derivative is evaluated at the given values of the variables. The ADI variables are presented below:

x= ADI Properties:	y= ADI Properties:	z= ADI Properties:
val: x_0	val: y_0	val: $f(x_0, y_0)$
jac: $\{[\frac{\partial x}{\partial x} = 1]\ [\frac{\partial x}{\partial y} = 0]\}$	jac: $\{[\frac{\partial y}{\partial x} = 0]\ [\frac{\partial y}{\partial y} = 1]\}$	jac: $\{[\frac{\partial f}{\partial x}\vert_{x_0,y_0}]\ [\frac{\partial f}{\partial y}\vert_{x_0,y_0}]\}$