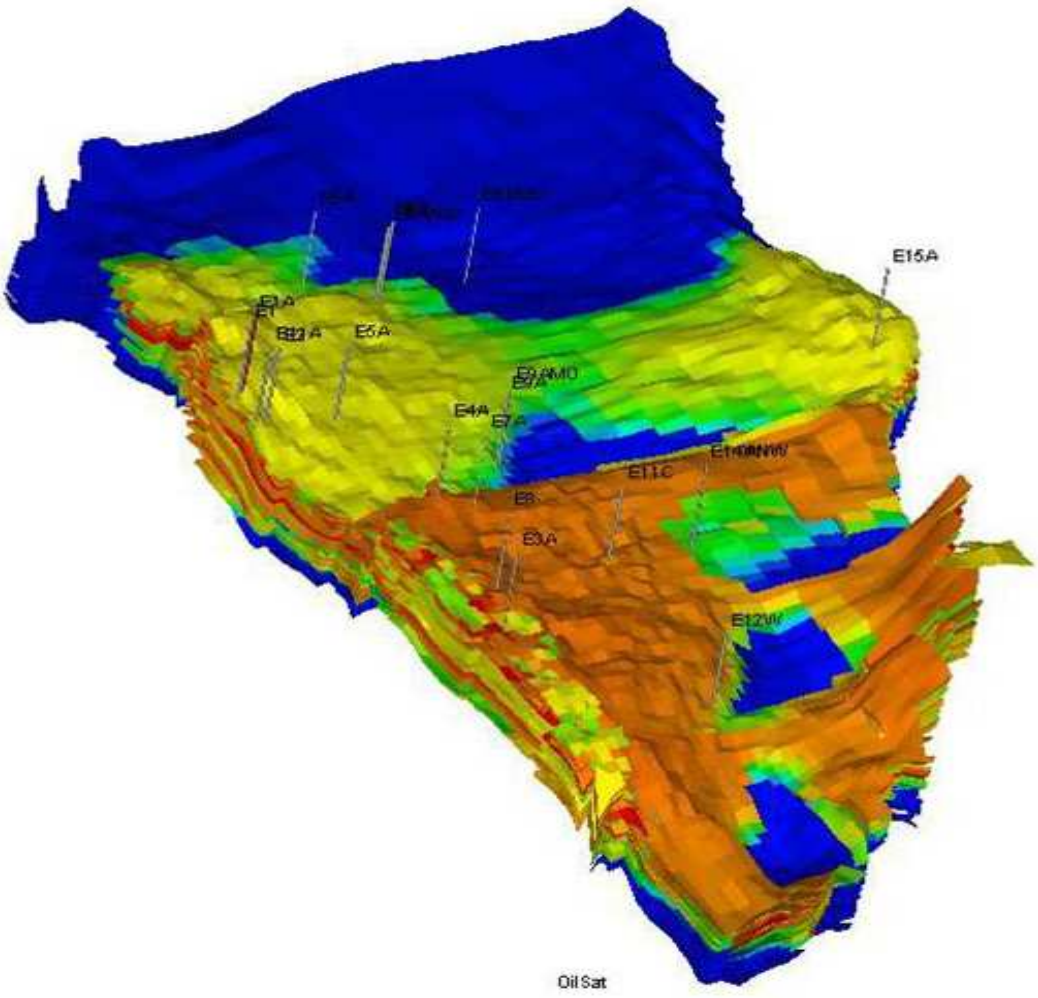
Basics of Reservoir Simulation
With the
Eclipse Reservoir Simulator



Lecture Notes

Øystein Pettersen

Contents

List of Figures

Introduction

Simulators are used in several different contexts, e.g.

- flight simulators
- control room simulators
- ship bridge simulators

Such simulators are characterised by a real-life environment, i.e. the user has a "look-and-feel" of the real world that is entirely authentic, with a simulated real-world feedback provided by a computer program. E.g. in a flight simulator the controls are identical to the real aircraft controls and the view through the "air plane windows", which are actually computer screens, is just like in the real world, a view which is updated based on user interactions. The simulated air plane's reactions to user actions are perfectly described by laws of motion, so that the computer (simulator) gives the user an authentic feedback to the way the aircraft is handled. The advantage is of course that the user can get training in handling of dangerous situations without the real-life consequences of errors.

In *reservoir simulators* the objective is much the same – to simulate the exploitation of a real reservoir without the costs of real life trial and error, e.g. to test different productions scenarios to find an optimal one before the reservoir is actually put on production. Unfortunately, in contrast to e.g. a control room simulator, we cannot expect the reservoir simulator to provide an exact replica of flow in, and production from the reservoir. This is due to a number of factors, but the most important difference is that while e.g. an aircraft is accurately and detailed described in the computer program, and the equations of air flow are well known and can be solved with great accuracy, the description of the reservoir and the boundary conditions for the equations for flow in a porous rock are known only with a great deal of uncertainty. Primarily, the pore system itself and the flow in this system happens on a detail level that is impossible to describe or model – and if it were possible the resulting problem would be too large to solve on present-day computers anyway. In addition to that, the detail structure of the pore system will obviously be unknown. So the problems we look at are a result of

- generalisation of real micro-scale data from macro scale observations (seismics and well data)
- upscaling of detail level behaviour to a manageable scale and problem size.
- some uncertainty in the descriptive model itself (equations for multiphase flow in porous media).

With such uncertainties both in the input data and the solution procedure, it is not surprising that "exact" answers cannot be provided – typically manifested by comparing the simulated data to actual field data after some time of field production.

Still reservoir simulation is regularly used with great success, and remains one of the support pillars for decision making when field plans for development and operations are made. This is, because even if there may be uncertainties tied to simulated results, the simulations reveal invaluable information about the reservoir flow, and not the least, can point to areas which need closer investigation or can represent elements of risk to the production scenarios.

An important aspect in this discussion is that offshore operations require much more robust plans and economics than small onshore reservoirs do – plainly put, one cannot afford to do (grave) errors when determining an operations plan, since the investments up front are so large.

In such a context, some of the tasks of the reservoir engineer in charge of doing reservoir simulations are,

- evaluate the quality of the input data
- transform the input data to a form suitable for simulation
- identify which parts of the data are most sensitive to uncertainty
- identify necessary additional data acquisition
- identify key data which may directly influence choice of operations plans, and uncertainty tied to these
- perform a suite of reservoir simulations
- evaluate quality of results from the simulations, uncertainties, areas of robustness
- utilize available field production data to tune simulations
- point to potential future problems / solutions, suggest production plans

On this background we clearly see that the simulator itself is merely a tool used in the process – in many cases an indispensable tool, but the human factor – the engineer, is undoubtedly the main factor in this process.

Knowledge of how to use the tool is a necessary ingredient in the work flow, but really not a goal *per se*. Still, it remains a fact that it would be impossible to carry out the tasks above without this knowledge.

The objective of these lecture notes is to provide the necessary background for using the reservoir simulator as a tool. Many subjects are common to most or all simulators, and have been tried covered in a general sense. But in the end all input must be defined in a detail language defined by each simulator. Although the general set-up for input files for different reservoir simulators has much in common, the detail syntax is obviously unique to each simulator. In these notes, we focus on the syntax used by the market leading reservoir simulator *Eclipse*, by Geoquest / Schlumberger, which is used by many oil companies with offices in Norway.

Eclipse includes options for simulating "almost anything", through several thousand keywords. In these notes we aim at covering those subjects and keywords which are sufficient and necessary to run the majority of "standard" simulations.

Many, perhaps all the most common, subjects and keywords are described in these notes, which hence can be a sufficient basis for setting up input data files for a great deal of problems. Still, the user must be aware that *only a small subset of the keywords are described*, and for many of the keywords *only a subset of the available options* have been described. The objective of these lecture notes is thus to enable the user to understand the required input data to a reservoir simulator, convert the input issues to Eclipse format and run simulations with Eclipse, for not-too-complex problems. With this background it should be relatively easy to advance to other and more general problem settings, by consulting the Eclipse reference documentation.

Hence, the notes cannot in any way replace the *Eclipse reference manual*, where the user should always look to find the official and detailed description of any keyword. Further, the *Eclipse technical reference* contains detail descriptions of methodology and physical background for many subjects that are relevant when attempting to understand the workings of the simulator.

It cannot be avoided that the contents of the notes becomes technical at times, and one may easily get the impression that the reservoir simulation is an exercise in technicalities. Therefore keep in mind the list of tasks above, and that the simulator is merely a tool – the real challenge is not to get keyword syntaxes correct (although they must be), but to interpret input and output from the simulations in a process where the decisions based on the simulations are the goal, not the simulations.

From the author's point of view it is the use of the simulator that is important, not the technicalities. Some general rules can be given, and are given in many excellent books, but expert use of simulators is primarily a question of experience. Unfortunately most of the subjects tied to real life use of the simulator are beyond the scope of these notes, so interested readers should consult other books in the field.

One of the important tasks for a user of a reservoir simulator is to evaluate the quality of simulated results. Are the results "as expected"? – Do the results appear trustworthy? ...
Again, experience is an important factor in such evaluations, but another factor should not be neglected, namely knowledge of the inner workings of the simulator. (Too) many engineers treat and use the simulator as a "black box". It is the author's opinion and experience that input data will generally be of higher quality, potential simulation errors will be quicker identified, and result evaluation will be more robust if the user has some knowledge of how the input data is handled by the simulator.

A complete treatment of the inner workings of Eclipse (or any other simulator) is obviously far beyond the scope of these notes (Ref. Eclipse technical appendices or many books on the subject), but many basic features will be covered.

Prerequisites

Knowledge in basic reservoir technology is required – as the very least the reader must know the terminology, although more is an advantage.
Some background in calculus and differential equations is recommended.
Especially for the last part of the text, a basic background in linear algebra and some numerical methods is advantageous.

1. Overview – minimum required data input

1.1 The grid

Real reservoirs are of course continuous, and all flow-dependent parameters change continuously with time. A reservoir simulator, which is a computer program, cannot, however, relate to continuous variables. It is therefore necessary to subdivide the continuous reservoir into a finite number of discrete elements, and also to define time development in a discrete sense. Then at any time, all properties will be understood as being constant within one element, and dynamic (time dependent) updates will occur only at defined time steps, while all data are seen as constant between such updates.

The subdivision of the reservoir into finite volume elements or cells is denoted a discretisation of the reservoir, and the set of elements is called the reservoir grid. Intuitively, we would expect the simulator results to be more reliable if the grid closely approximates the reservoir itself. Since most real reservoirs have a complex geometry and internal structure, a good approximation will normally require a large number of cells. Computational considerations will on the other hand restrict the number of cells by available computer memory and/or acceptable run times, so the grid we choose to use will almost always be a compromise between different desires.

Grid design, or how to build a suitable ("best") grid for a given reservoir is a science (or art perhaps) in itself, which we will not study here.

The simplest grids are those which are comprised of a number of equal cube-shaped cells. The grid is then uniquely defined by the size of each cube, and the number of cubes in each of the major directions, *X, Y, Z*.

Eclipse convention: *I, J, K* are used for indices in *X, Y, Z* – directions, and the ordering is defined as in "normal reading order", i.e. *I* runs from left to right, *J* runs from top of a page to bottom, and *K* runs from page to page. In a grid oriented along standard geographical axes, *I* would increase from west to east, *J* would increase from north to south, and *K* downwards.

We denote the length of the cell (cube) in the *X, Y, Z*-directions by *DX, DY, DZ* respectively. Then the simplest grid is comprised of identical cells, *NX* cells in the *X*-direction, *NY* cells in the *Y*-direction, and *NZ* cells in the *Z*-direction, totally *NX*NY*NZ* cells.

A simple extension of this grid, is to allow for *DX* to be dependent on the *I*-index, and *DY* to be dependent on the *J*-index. (Note that we don't allow for *DX* to be dependent on *J*. Figure out why this restriction exists.). An example of such a grid is shown in Figure 1.

A real reservoir is never completely horizontal, and also has a complex internal structure defined by the different geologic units that build the reservoir. These units are also commonly natural flow units, and hence when building grids, we generally try to honour the depth variation of the units in the grid. So a natural generalisation of the simple grid is to keep *DX* and *DY* constant, but allow *DZ* (the cell thickness) and cell depth to vary from cell to cell. Figure 2 shows an example. Such grids are much used in the industry.

1.2 Petrophysics

In the reservoir porosity and permeability may have large variation. In each grid cell we define a value of these variables that is a representative cell average value of the property. Although permeability is a

tensor variable, Eclipse only allows for the diagonal elements K_{xx}, K_{yy}, and K_{zz}, called *PERMX*, *PERMY*, and *PERMZ*. Also, the mixture of porous and non-porous material is handled by the net-to-gross parameter, *NTG*.

1.3 Fluid properties

In this introduction we look at the simple case two-phase flow with water and undersaturated oil (i.e. the oil pressure is always above the bubble point pressure). Then the required data for defining the fluids are,

- density at surface conditions
- PVT relations (volume factors, viscosity)
- constant gas resolution factor
- relative permeabilities k_{rw} and k_{ro} as functions of water saturation
- water – oil capillary pressure

1.4 Soil properties

These are simplified in most reservoir simulators, and typically are comprised of some relationship which define how porosity (or pore volumes) and permeability change with fluid pressure. The simplest and most used assumption is constant compressibility.

1.5 Equilibration

The initial reservoir state is defined by the pressure and saturations in each grid cell at the start of the simulation. It is convenient to let the simulator calculate the initial state based on the reasonable assumption that the reservoir fluids are in equilibrium at no-flow conditions. We then only need supply the depths of the oil-water and gas-oil contacts, and the fluid pressures at a reference depth. The simulator can then calculate the necessary state from fluid weight versus depth gradients.

1.6 Well specification

Wells are the most significant means to add or remove fluid to/from the reservoir. The definition of how wells should be operated in a production scheme can be very complex, but at least we will need to describe the well positions, at which depths they are open to the reservoir, and the production or injection rates.

1.7 Dynamics

The reservoir production scheme is rarely or never static. During the field's life time, wells will be added or closed, well rates will be changed, and operating constraints may be added or removed. All such events are handled by defining a series of milestones, dates at which operating conditions may change.

1.8 Output

The simulation would be rather useless if we don't require some sort of results from the run. Such results can be either text files, spreadsheet files, or files intended for graphic visualisation. Since the amount of possible output is enormous, *only data items required by the user are written*, and at user defined times only. Often the milestones mentioned in 1.7 coincide with these output-times.

1.9 Simple Eclipse data file contents

A. Syntax

All data needed by Eclipse is collected in an input data file, which is an ordinary text file. The different data items are identified by keywords, and most often followed by the associated data. The keyword data is always terminated by a slash ('/'). In some cases data occur in groups, where each group is terminated by a slash. Exactly how this works in practice will become clear when we look at the keywords in detail.

Data organisation
An Eclipse data file is comprised of eight sections headed by a section header. (Some of the sections are optional). These sections must come in the prescribed order, but the order of the keywords within each section is arbitrary (except the SCHEDULE section where time-dependency is handled in the order it is defined).

The data sections, with headers, are (note: *This order is required*),

RUNSPEC (required)
Run specifications. Includes a description of the run, such as grid size, table sizes, number of wells, which phases to include an so forth.

GRID (required)
Defines the grid dimensions and shape, including petrophysics (porosity, permeability, net-to-gross).

EDIT (optional)
User-defined changes to the grid data which are applied *after* Eclipse has processed them, can be defined in this section. The EDIT section will not be covered in these notes.

PROPS (required)
Fluid and rock properties (relative permeabilities, PVT tables, ...)

REGIONS (optional)
User defined report regions, or e.g. regions where different rel.-perm. curves apply can be defined in this section.

SOLUTION (required)
Equilibration data (description of how the model is to be initialised).

SUMMARY (optional)
Results output is primarily of two types:
1) Scalar data as a function of time (e.g. average field pressure)
2) Data with one value pr. grid cell (e.g. oil saturation). These are only output at chosen times.

This section is used to define output of the first kind, by specifying which data items to write to report files.

SCHEDULE (required)
Well definitions, description of operating schedule, convergence control, control of output of the second kind described above.

Comments
Any line in an Eclipse input file that starts with two dashes (--) is treated as a comment.
It is strongly recommended to use many comments, the advantage is easily seen when returning to the file for later updates. (Note that the two dashes must be in column 1 and 2, no leading blanks.)

The Eclipse data format, default values and data repeats
A property like porosity must be defined with one value for each grid cell. The *Eclipse data format* expects exactly this: With a total of $N = NX*NY*NZ$ cells, N porosity values must be given. The order is according to the "book order", line after line until a page has been read, then the next page until all data has been read. I.e. First the top layer ($K=1$) is read, line by line (begin with $J=1$, read $I=1,...,NX$, then $J=2$, etc.) Then repeat for $K=2,...,NZ$.

An example for porosity is then,

```
PORO
0.32 0.24 0.24 0.3 0.31 0.31 0.312 0.32
0.26 0.26 0.26 0.27 0.3 ...
...
... 0.26 0.25 0.25 0.25 0.28 /   (Totally N values)
```

Note that it is allowed to put a comment after the terminating slash.
For this particular example, if the entire porosity array must be supplied, it is typically generated by an external program, like e.g. IRAP RMS or FloGrid, not input manually.

When providing a data vector associated with a keyword, often the same value will be repeated. One example is that we often look at models where we assume that porosity is constant in each layer. Then the porosity keyword would be comprised of *NX*NY* equal values for layer 1, followed by another *NX*NY* equal values for layer 2 etc. This situation is so common that Eclipse fortunately allows for a shorthand notation: `n*val` means that the value `val` shall be repeated n times. So,

```
PORO
   0.23 0.23 0.23 0.23 0.23 0.23 0.23 0.23 0.24 0.25 0.25 0.25 /
```
is equivalent to

```
PORO
 8*0.23 0.24 3*0.25 /
```
Note that there must not be any blanks on any side of the asterisk (star).

In many cases required data items may be defaulted. Eclipse has defined default values for many values, and if we on input set these items to default, Eclipse will use the defined default values instead. A default value is specified by "`1*`", and can be repeated as above, with n* (e.g. to specify that the next three items shall be defaulted use 3*).
If "the rest" of the data items are defaulted, this is not necessary to state explicitly. As an example, the COMPDAT keyword requires 14 data items, but we will default the last five.
Either of the syntaxes below are valid (don't care about the meaning just now).

```
COMPDAT
  WP1  10   2    1   2    OPEN  2*  0.5  5* /
  WP2  10   3    1   2    OPEN  2*  0.5 /
/
```

INCLUDE statements
Some of the data items can be really huge, especially the GRID section data. Since we seldom want to look at such data once they have been defined anyway, it is convenient to keep them outside the DATA file.

Eclipse allows for including other files in the data file, also on several levels (a file that is included may itself contain INCLUDE statements). Eclipse also knows about absolute and relative addressing of folders, so we can (and the user is encouraged to) keep included files in separate folders. If we e.g. use one folder for each run, keep all the data that don't change between runs in one or several folders, and use relative addressing in the INCLUDE statements, we can obtain a well organized file structure.
(The main reason for using relative addressing is that we then can move an entire project to a new parent folder without having to change all the INCLUDEs in the data files.)

Example (recommended type of file structure)
Assume a folder hierarchy where the "`home`" folder for this suite of runs is `SimField`. This folder contains the two subfolders `include` and `runs`. The `include` folder contains the subfolders `GRID, FLUIDS, DIV, SCHEDULE`, while the `runs` folder contains one folder for each run, e.g. `BASE, SENS1, SENS2, ...`
The grid definition (grid cell coordinate data) is in the file SIMF.GRDECL, which resides in the GRID subfolder. Our data file is in one of the subfolders to `runs` (note how it doesn't matter which), and the appropriate include statement in the GRID section would be,

```
INCLUDE
 '../../include/GRID/SIMF.GRDECL'  /
```

If the file to include resides in the same folder as the data file, this will obviously be shorter, as e.g.

```
INCLUDE
 GF_PVT_TABLES.PROPS  /
```

Note: Each INCLUDE statement contains exactly one file to include. It is not possible to list several files in the same INCLUDE keyword.

B. Data file (“BASIC data input example”)

Following is an example of minimum required data input to Eclipse. The keywords are commented using lines starting with two dashes. Full understanding of the file is not to be expected at this stage. Note: Section headers are in bold for clarity only. An eclipse data file must be in text-only format, not e.g. saved as a Word-document.

```
RUNSPEC  ============================================================
-- Title is used as header on output, to identify run
TITLE
   Example simple Eclipse file, lecture notes

-- Specify dimension of model, NX, NY, NZ
DIMENS
-- NX  NY   NZ
   10   3    3  /

-- Phases included (oil and water, i.e. 2-phase run)
OIL
WATER

-- Units to use, alternatives are METRIC, FIELD or LAB
FIELD

-- Specify maximum values for well data (# means “max number of”)
WELLDIMS
-- #wells  #cell-connections  Next three values are defaulted
     4            5               3* /

-- Start date, i.e. “Day 0” in simulation
START
  1 JAN 2004 /

GRID      ============================================================
-- Turn off writing of data to report file
NOECHO

-- Definition of grid cells. From DIMENS keyword above, we have:
-- N = NX*NY*NZ = 10*3*3 = 90.
-- In each cell DX = DY = 1000 ft and DZ = 50 ft
-- (Dimension measured in feet since units were defined as FIELD above)
-- The grid cells are then defined by:
DX
   90*1000 /
DY
   90*1000 /
DZ
   90*50 /

-- So far the grid shape has been defined, but we also need to define at
-- which depth (measured from sea level) the cell tops are. We assume a
-- horizontal reservoir, and input depths which are consistent with the
-- DZ-values we defined above.
TOPS
 30*5000 30*5050 30*5100  /

-- Permeabilities are constant, so define 90 equal values.
PERMX
  90*200 /
PERMY
  90*200 /
PERMZ
  90*200 /

-- Porosity is constant in each layer, so define 30 values in layer 1,
-- 30 in layer 2, and 30 in layer 3. Each layer contains 30 cells.
PORO
 30*0.3
 30*0.23
 30*0.18 /
```

```
-- Request to write an INIT (initial) file. Contains all data used to
-- initialize the model
INIT

-- Turn report writing back on
ECHO

PROPS    ================================================================

-- Relative permeability for water and oil,
-- and water-oil capillary pressure, as a function of water saturation
SWOF
-- Sw  Krw     Krow  Pcow

0.220  0.0000  1.0000  0
0.300  0.0700  0.4000  0
0.400  0.1500  0.1250  0
0.500  0.2400  0.0649  0
0.600  0.3300  0.0048  0
0.800  0.6500  0.0     0
0.900  0.8300  0.0     0
1.000  1.0000  0.0     0  /

-- PVT properties for water.
-- (Pref: Reference pressure for rest of data (psi)
--  Bw: Volume formation factor for water
--  Cw: Water compressibiblity
--  ViscW: Water viscosity )
PVTW
-- Pref     Bw       Cw     ViscW
  4014.7  1.029   3.13E-6   0.31      0 /

-- PVT properties for oil
PVDO
-- P      Bo     viscO
 3337   1.2600   1.042
 3725   1.2555   1.072
 4139.5 1.2507   1.096
 4573.2 1.2463   1.118
 5053.9 1.24173  1.151
 5487.5 1.2377   1.174
 5813.9 1.2356   1.2    /

-- Dead oil: Rs (Gas resolution factor) is constant
RSCONST
-- Rs  Bubble-point-pressure
  0.4       3337.0  /

-- Specify constant rock compressibility.
ROCK
-- Pref    Cr
  14.7    3.0D-6          /

-- Fluid densities at surface conditions
DENSITY
-- Oil    Water   Gas
  49.1   64.79  0.06054  /

SOLUTION   ================================================================

EQUIL
-- DD = Datum depth, the depth to which all reports will be referenced.
-- DD     Pressure@DD  OWC   Pcow(OWC)    Default rest of data items
  5000       4800      6000    0.0           6*     /
```

```
SUMMARY   ============================================================

-- List data vectors which we want stored for graphics post-processing

-- Field Oil Production Rate
FOPR
-- Field Oil Production Total
FOPT
-- Field Water Cut
FWCT
-- Field Pressure (averaged reservoir pressure)
FPR
-- Field Oil In Place
FOIP
-- Field Water Production Rate
FWPR
-- Field Water Injection Rate
FWIR
-- Well Water Cut for all wells
WWCT
 /

SCHEDULE  ============================================================

-- Specify output of graphics result files for cell data, and times which
-- to write these. (Details to come later)
RPTRST
 BASIC=5 NORST=1 FREQ=6 /

-- Well specification: Give names, positions (i, j) and main phase of wells
WELSPECS
--wname  group  i  j   Z(bhp)  prefPhase
   WP1     G   10  2    1*      OIL   /
  /
-- (Note two slashes, one terminates each well, one terminates the keyword)

-- Completion data, the well is open to the reservoir in cells in layers
-- from k_hi to k_lo.
COMPDAT
--wname  ic  jc  k_hi k_lo open/shut 2*Don'tCare well_diam  Default...
   WP1   10   2    1   2     OPEN       2*           0.5  4*      /
  /

-- Production data, specify producing rate and constraints
-- Well WP1 will produce with an oil rate of 2000 STB/day constrained by
-- that well's flowing bottomhole pressure is minimum 3350 psi.
-- (Details to come later)
WCONPROD
--wname open/shut ctrlmode orat 4*Default bhpmin Rest default...
   WP1     OPEN     ORAT   2000    4*      3350  /
  /

-- Milestone dates, e.g. to write results
DATES
 1 JUL 2004  /
 1 JAN 2005  /
/

-- Change production rate to 4000 STB/day
WCONPROD
--wname open/shut ctrlmode orat 4*Default bhpmin Rest default...
   WP1     OPEN     ORAT   4000    4*      3350  /
  /

DATES
  1 JUL 2005  /
  /

END
```

A note on units

Eclipse accepts data in metric units, as well as field or lab units. Specification of which unit convention to use is done in the RUNSPEC section. Since metric units are widely used by Norwegian oil companies and international companies based in Norway, unless otherwise specified all the examples in these notes will assume metric units are used.

Note on character data

In many keywords some of the required input is text, or character data. (Example: In the DATES keyword, each line contains "DAY MONTH YEAR". Here DAY and YEAR are numerical values, while MONTH is a character string, e.g. JUL as in the example above.
When character data is defined, this can be done with or without quotes (single quote, the vertical quote on the keyboard). Hence the DATES keyword above can be defined in two alternative ways:

```
DATES
  1 JUL 2005  /
  /
```

or

```
DATES
  1 'JUL' 2005  /
  /
```

Both these syntaxes are accepted by ECLIPSE. It is *always* permitted to use quotes when defining character data, but in most cases ECLIPSE accepts the input without quotes, which can be preferable from the user viewpoint. In some exceptional cases, however, Eclipse *requires* the quotes. As a rule of thumb the quotes are required if necessary to resolve ambiguities. (And nothing worse than an error message results if a text string was defined without quote where one was expected.)

Examples where quotes are required

1. When the text string contains *blanks* (or other separator characters)
   A well name as B-1HA is accepted without quotes, but well name 'B1 REV' requires quotes.
2. File paths which include subfolders, as the slash (/) would else be misinterpreted as a terminator. Example: The two INCLUDE statements on the bottom of page 10. The first one requires quotes, the second one, which is a single file name, does not.
3. *Wildcards* (as defined in the Well Completions section, page 42) always require quotes.

In the examples in these lecture notes, quotes have been used if and only if they are required.

Technical comment: In early versions of ECLIPSE, quotes where required for *all* character input. As new versions have been released the set of data requiring quotes has steadily been reduced. Syntax which is accepted in current versions of ECLIPSE may therefore not be valid in older versions.

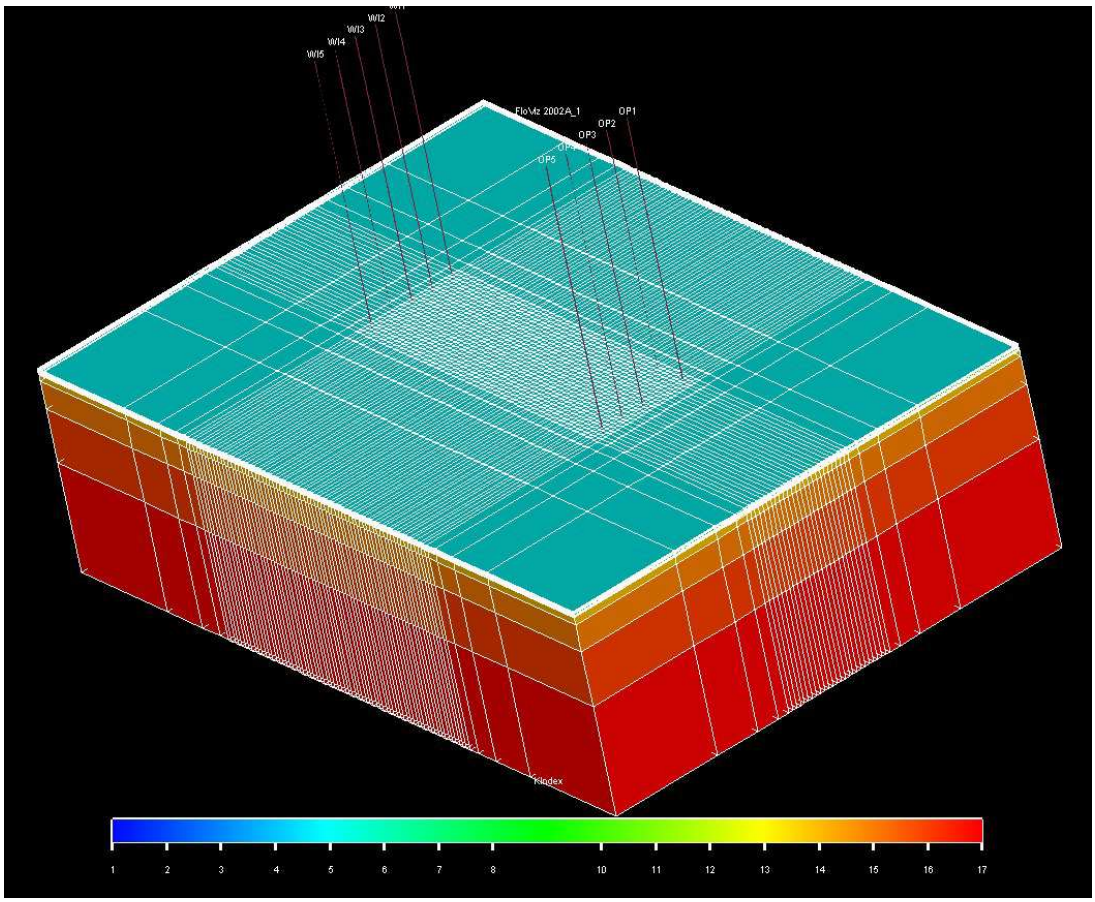


Figure 1. Regular Cartesian Grid

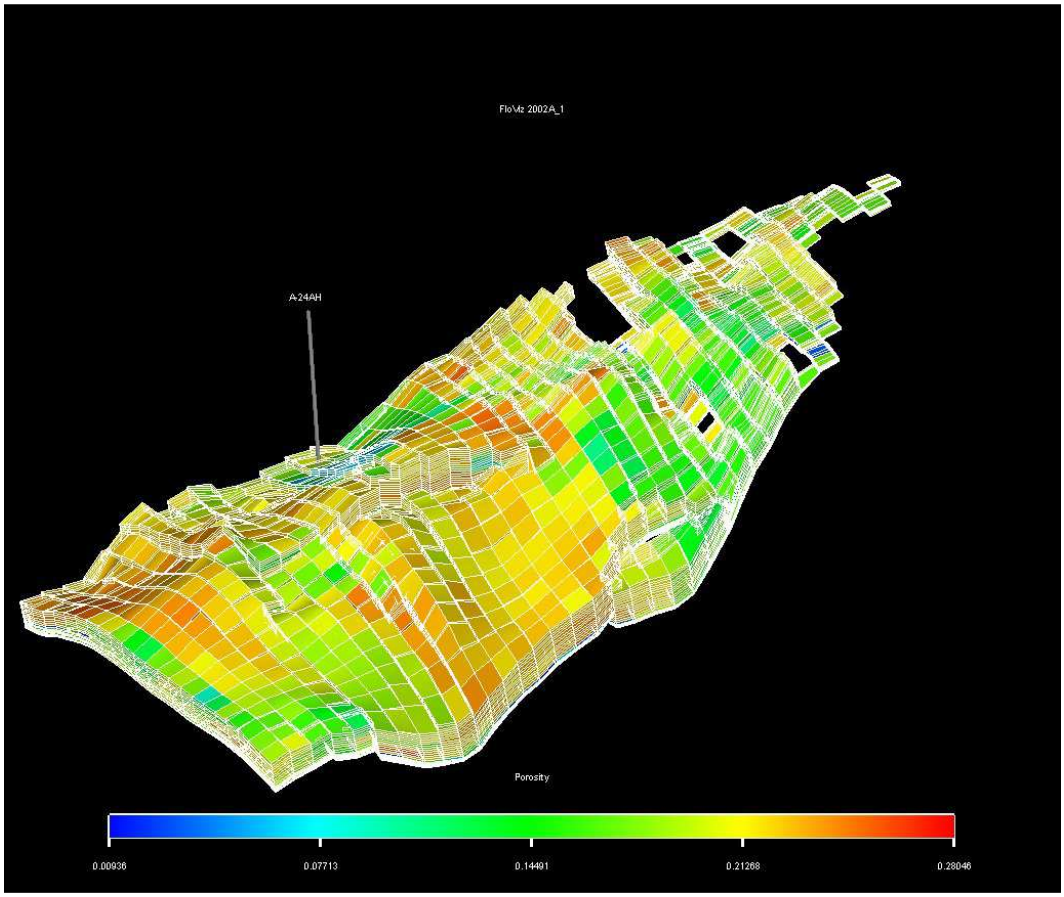


Figure 2. Regular XY-Cartesian Grid

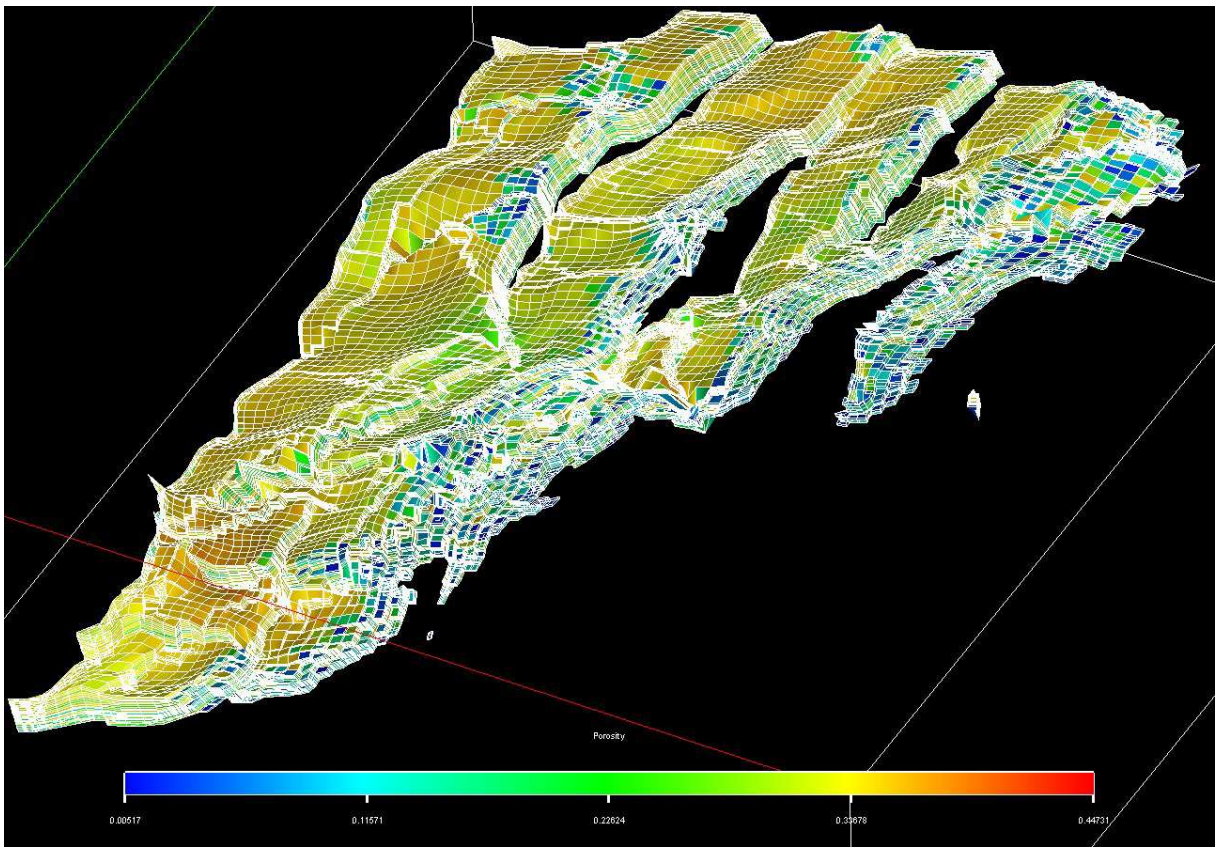


Figure 3. Irregular structured grid

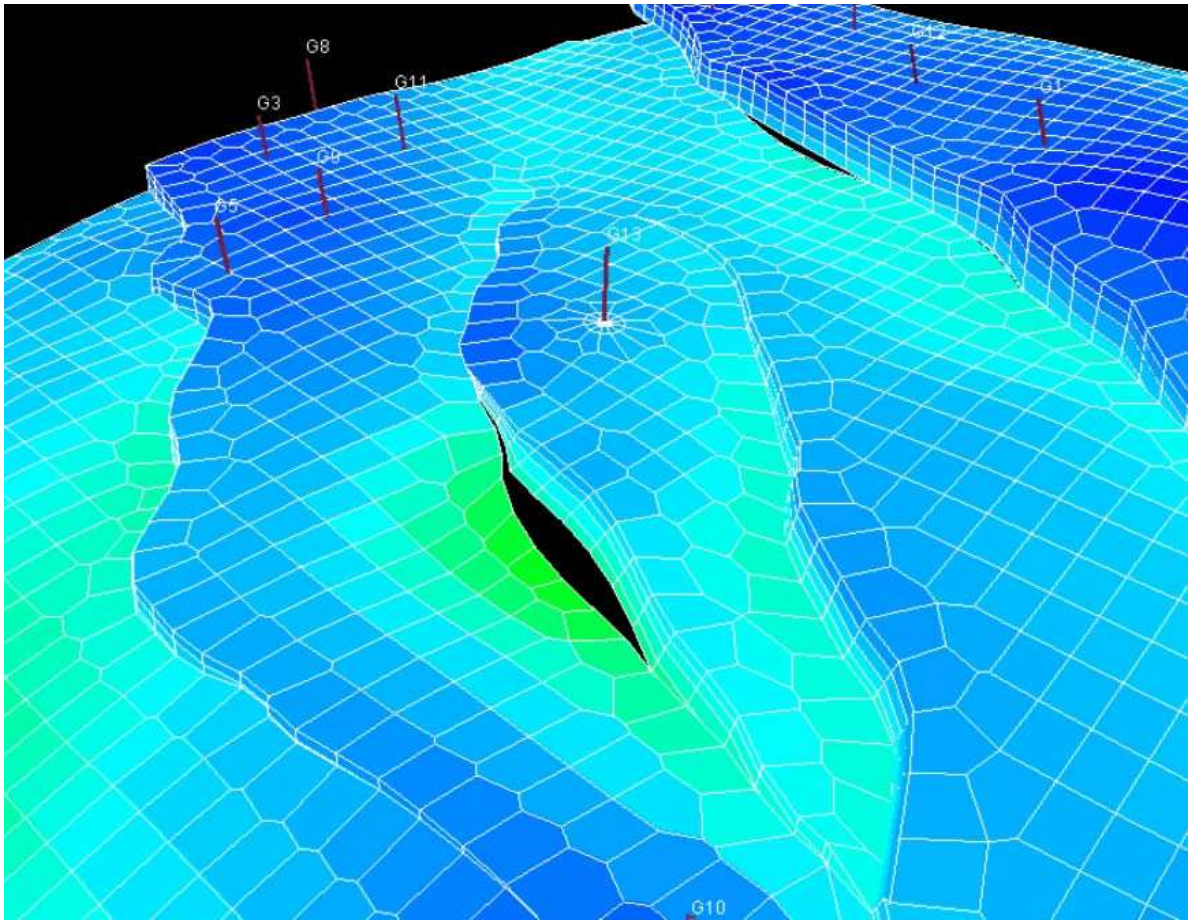


Figure 4. Unstructured grid

2. The RUNSPEC section

As the RUNSPEC section comes first in the Eclipse data file it is described here. Many of the terms used in this description will not be defined before later, but are included here for completeness.
The section contains *run specification* data, i.e. information Eclipse needs at the very beginning of the job, before it commences reading of any other data, typically for determining how much computer memory to reserve for the different kinds of input.

Grid dimension – keyword DIMENS

Specification of the number of cells in *x*, *y*, and *z*-directions:
DIMENS
NX NY NZ /

Example:
The grid is 30 cells in the *x*-direction, 15 cells in *y*-direction and 5 cells in the *z*-direction (vertical) – will also be referred to as 5 *layers*.

```
DIMENS
-- NX NY NZ
   30 15  5  /
```

Phases

The phase definition specifies how many and which phases are present in the run. Each included phase is specified by one keyword. The alternatives are,
OIL
WATER
GAS
VAPOIL (Vaporized oil in gas)
DISGAS (Dissolved gas in oil)

(No slashes)

Example
For a standard three-phase problem with no gas dissolved in the oil we would specify,

```
OIL
WATER
GAS
```

Unit system

Specify the unit system the data are defined in (all data must be in the same unit system)
Choose one of

`FIELD` (Standard oil-field units)
`METRIC` (well known metric system)
`LAB` (similar to metric, but everything is scaled down to lab size)

(No slashes)

Start date

Specify the date when the simulation starts, also called initial date or “day 0”.
Date format is identical to the format in the DATE specification (see section 8.2)

Example
The simulation starts at April 14., 1997:

```
START
 14 APR 1997  /
```

Unified / Non-unified files (chapter 11)

Specify if unified files are to be used. Non-unified is the default and will be assumed if specification of unified is omitted.
Keywords:

`UNIFIN`	Specifies unified input files are used (relevant for restarts)
`UNIFOUT`	Request for unified output files (summary and restart)

Data checking only

Eclipse can be requested to check the input data syntax without running the simulation. This is done by the keyword

`NOSIM` (no slash)

Input errors in the SCHEDULE section will not be discovered before Eclipse encounters them. (Eclipse does not read the entire file before the run starts, only as much as it needs). By routinely doing a data check run after major changes, unexpected terminations of simulations due to data errors can be avoided.

Table dimensions

Eclipse needs to know how many tables we are going to define, and how many entries the tables will have when it sets up the run dimensioning, i.e. when the RUNSPEC section is read.

> Technical note: This is because Eclipse is written in FORTRAN, and does not use dynamic memory allocation.

The appropriate keyword is `TABDIMS`, which contains the following relevant items, (for others, see Eclipse documentation)

1. `NTSFUN:` Number of saturation table families (rel.-perm-families / regions)
2. `NTPVT:` Number of PVT families / regions
3. `NSSFUN:` Max number of lines in any rel-perm table
4. `NPPVT:` Max number of lines in any PVT table or Rock compaction table
5. `NTFIP:` Max number of FIP regions (defined by FIPNUM)
6. `NRPVT:` Max number of R_s-nodes in a live oil PVT table or R_v-nodes in wet gas PVT table

Example

To set all values to their default:

```
TABDIMS
-- NTSFUN NTPVT NSSFUN NPPVT NTFIP NRPVT
      1     1     20     20    1     20   /
```

EQLDIMS

Used to set some parameters for equilibration. Can be defaulted (omitted) in most cases.

Well data dimensions

Eclipse must know how many wells to allocate memory for, and how much memory is needed for each well. This is specified in the WELLDIMS keyword, with entries,

MAXWELLS	Maximum number of wells in the model
MAXCONNW	Maximum number of connections (completions) in each well
MAXGROUP	Maximum number of groups in model
MAXWELLG	Maximum number of wells in a single group

The default value for all the entries is zero, so this keyword must be specified if the model contains any wells.

Example

```
WELLDIMS
-- MAXWELLS MAXCONNW MAXGROUP MAXWELLG
     18        6        1        18      /
```

NSTACK (chapters 17-18)

To redefine the size of the NSTACK, i.e. the number of search directions to store for use by the linear solver, include the keyword NSTACK. The default value of NSTACK is 10.

Example

To increase NSTACK to 25,

```
NSTACK
  25   /
```

Aquifer specifications – AQUDIMS (chapter 14)

Specify maximum number of lines in the AQUNUM and AQUCON keywords. (I.e. not the number of connections defined, but the number of lines used to define the connections in the keyword)
Default values are zero, so this keyword must be specified if aquifers are used.

Example

```
AQUDIMS
-- Max#NumLines  Max#ConnLines  (rest not used for numerical aquifers)
       5             7        /
```

Grid options (chapter 13)

The GRIDOPTS keyword is used to flag that negative index transmissibility multipliers are used. (And some other flags which are not used in these notes)
To allow for MULTX– and related keywords, set

```
GRIDOPTS
  YES  /
```

The default is “NO”, so the keyword is not needed if negative index transmissibility multipliers are not used.

Rock compressibility options (chapter 6)

When rock compressibility is defined by the ROCKTAB (or ROCKTABH) keyword, the keyword ROCKCOMP must be included in the RUNSPEC section. The keyword has two associated items,

1. Specify if rock compaction is reversible (elastic) or irreversible (plastic), or if tables with hysteresis are used. Choices are REVERS, IRREVERS, and HYSTER
2. Number of rock compaction tables (normally equal to number of ROCKNUM regions)

Example

To specify that ROCKTAB tables are irreversible, and that the model includes three different rocknum regions,

```
ROCKCOMP
  IRREVERS  3  /
```

Local Grid Refinement (chapter 15)

If local grid refinements are included in the model, they must be specified in the RUNSPEC section, with keyword LGR, containing the following items,

1. MAXLGR Maximum number of LGRs in the model
2. MAXCELLS Maximum number of cells in any LGR
3. MAXAMLGC Maximum amalgamated coarse cells (not covered in these notes)
4. MAXAMLGF Maximum number of LGR amalgamations
5. MAXLGRAM Maximum number of LGRs in any amalgamation
6. LSTACK Equivalent to NSTACK for Local Grid solving (default: Equal to NSTACK)

Example

The model contains 10 local grids, five of these have been amalgamated to one, and three others comprise another amalgamation. The largest of the LGRs contains 160 cells

```
LGR
-- MAXLGR  MAXCELLS  MAXAMALGC  MAXAMALGF  MAXLGRAM  LSTACK
     10       160        1*         2          5       1*    /
```

3. Structured grids (Corner point grids) (GRID section)

The simple grids we defined in Chapter 1 can not in general be used to approximate complex reservoir geometries satisfactory, and this is regarded as being of great importance in the oil industry. The grid serves two purposes, which unfortunately often are conflicting. Firstly the grid is a set of finite volume cells which approximates the reservoir volume including internal characteristics. Secondly the grid is a device for solving the reservoir flow equations in a numerical fashion. Even though a great deal of research has been put into the challenge of constructing numerical schemes which work satisfactorily for "any" cell shapes, it is still true that the "best" numeric results are obtained on a regular cartesian grid, while the grids that are "good enough" approximations to the reservoir shape seldom are very regular. As noted above, constructing a grid that is an optimal compromise between actual reservoir shape, numeric accuracy and affordable computing time is a science in itself, which cannot be covered in this context.

The ideal grid is regular and cartesian. Cartesian means that all cell faces meet at right angles, while regular is used to specify that the lengths *DX*, *DY*, and *DZ* are compatible. If the flow rate components are equal in all three directions (no preferred flow direction) then compatible would mean $DX = DY = DZ$. However, in a reservoir the vertical flow is typically much smaller than horizontal flow[1], and in this case the ideal aspect ratio (ratio of the different cell lengths) is such that the typical time a flow particle uses to traverse the cell is roughly the same in all cell lengths. By this rule we should expect that in most grid cells, *DX* and *DY* are of comparable magnitude, and *DZ* much smaller (which is indeed the case in industrial grids). Many grids are comprised of cells which appear regular cartesian in the *XY*-plane, as seen from above, but not in cross-sections, i.e. in the *XZ* or *YZ*-planes, since the grid layers attempt to approximate the geological layers (Figure 2). Be aware that if the angles between the top or bottom and the vertical sides deviate too much from 90° the quality of the numerical results can be expected to get worse as the angle deviation increases. Apart from this warning we will not discuss this further.

The major reason that *XY-cartesian grids* are not more or less exclusively used is the existence of faults in the reservoir. Faults are surfaces of discontinuity in the reservoir with often very complex geometry. Often the goal is a grid that honours the geometry of the faults as closely as possible, and this is simply not in general doable with a regular grid.
Figure 3 shows an *irregular structured grid* where it has been attempted to let cell boundaries align to the faults as closely as possible.

[1] The term horizontal is used loosely, to denote flow along geological layers, or in the bedding plane direction

One alternative solution which is growing in popularity, is to use *unstructured grids*. By allowing for cells with any number of faces, any geometry can be modelled, and at the same time keeping the total number of cells at a manageable level. (Figure 4.)

The organisation of the cell structure and computational challenges have to now not been completely solved, which is the main reason the "traditional" grids are still dominating the scene. We will henceforth concentrate on structured grids.

A **structured grid** is characterized by

1. All cells are six-sided with eight nodes, i.e. topologically equivalent to a cube.
2. The cells are logically organized in a regular scheme, such that each cell's position in the grid is uniquely determined by it's *(I, J, K)*-index.

The Corner Point Grid

A structured grid is uniquely determined when all eight corners of all cells have been defined. In Eclipse, the corners cannot be completely arbitrary, but are constrained in the following manner:

Coordinate Lines

A coordinate line is a straight non-horizontal line defined by two coordinates

$(x, y, z)_{TOP}$ and $(x, y, z)_{BTM}$.

In the simplest and most common case the coordinate lines are vertical. For the regular cartesian grid the coordinate lines would be the vertical lines which in the *XY*-plane can be seen as the nodes, or the "meeting point" between four and four cells. The same principle is valid in general, although more complex. The coordinate lines form a skeleton of lines, each line identified by its index *(I, J)*. Except for edge cells, any coordinate line is associated with four columns of cells. Refer to Figure 5, and confirm that coordinate line *(I, J)* is associated with the four cell columns *(I-1, J-1, K)*, *(I, J-1, K)*, *(I-1, J, K)*, and *(I, J, K)*. (*K* takes all values 1,...,*NZ*). Recall that each cell was defined by its eight corner nodes. In each of the associated cells, exactly two corners (one on the top face and one on the bottom face) are on the coord line. Denote the four associated cell columns C_1, C_2, C_3, and C_4, and let the corners lying on the coordinate line be N_{1A} and N_{1B} in cell column C_1, N_{2A} and N_{2B} in C_2 etc. (Ref. Figure 6) The constraint defined by Eclipse is then that cell corners $N_{1A,}$ N_{1B},..., $N_{4A,}$ N_{4B} lie on the coordinate line for all layers *K*. A way of looking at this is to envisage the coordinate line as a straw, and the corner nodes as beads on this straw. The beads can move up and down on the straw, but are restricted to stay on the straw, which by definition cannot be bended – coordinate lines must be straight lines.

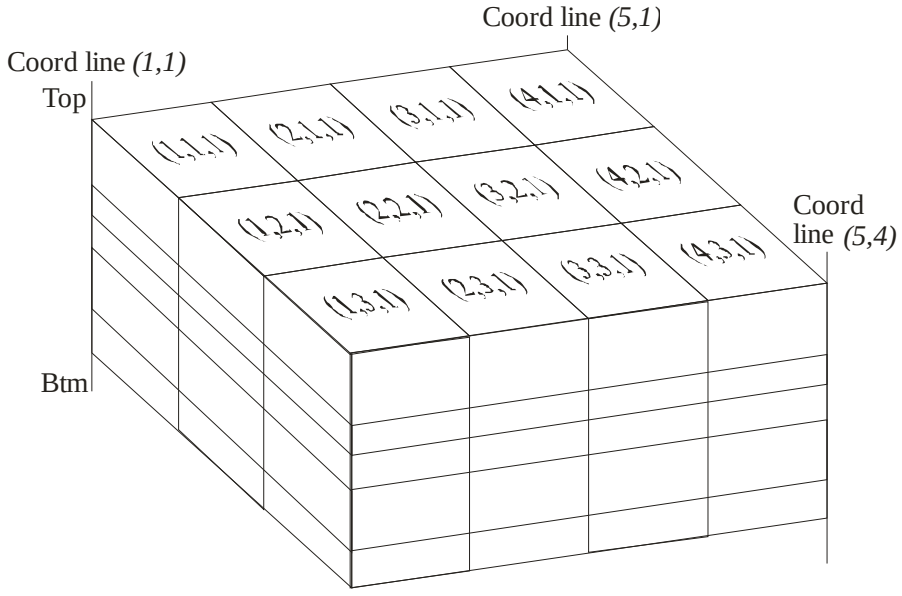


Figure 5. Cell indices and coordinate indices in a structured grid

A structured grid obeying the coordinate line restriction was originally called a *corner point grid* by the Eclipse development team. The Eclipse grid format is now a *de facto* industry standard, and corner point grids are used in a much wider context.

Examples

Ex. 1. Regular grid
The regular cartesian grid is obviously a corner point grid, and the coordinate lines are vertical. Each grid node (edge nodes excepted) is the meeting point for eight cells, and each of these cells has one corner equal to the meeting point. Since we required that all eight corners were defined for every cell, this means that in regular points like this, every node coordinate will be defined (and stored) eight times. (Waste of memory and hard disk space...)

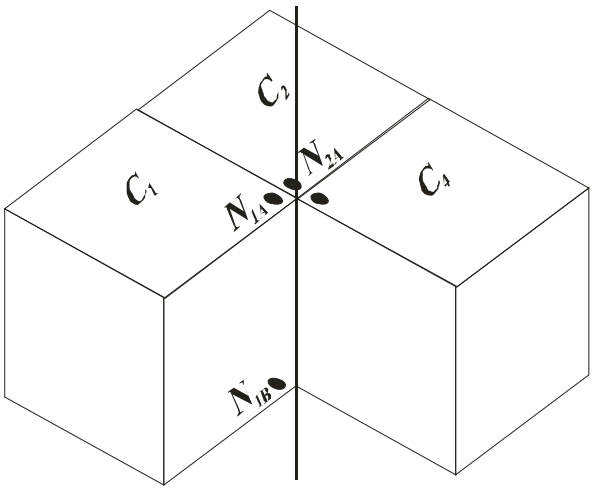


Figure 6. Three of the four cells sharing a coord line, and some corner points

Ex. 2. Fault
Modelling of fault throws is the main reason for introducing the corner point concept. Figure 7 shows a cross-section view in the *XZ* plane of a fault, with upthrown side (footwall) to the left and downthrown side (hanging wall) to the right. One way to look at this is that the grid has been constructed in the same fashion as the fault itself. Starting with a continuous grid, the cells on the hanging wall side has been allowed to slide downwards along the fault plane, which is a coordinate line. Note how the corners on the downthrown side has moved along the coordinate line, such that the depths of associated nodes are no longer equal. It is easy to convince oneself that such a modelling of faults could not be possible without the general way in which all eight cell corners are handled individually.

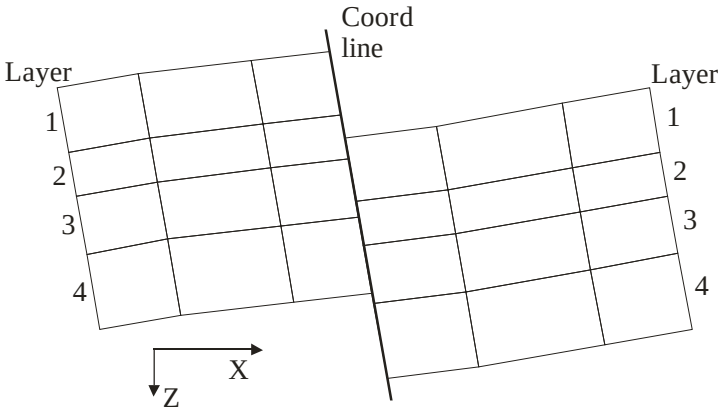


Figure 7. Cross-section view of a fault in a corner point grid

Notes:

- Since the coordinate lines must be straight lines, listric faults (curved) cannot be modelled[2].
- Obviously, the coordinate lines must form a grid skeleton which is physically realisable. This means that Y-faults and other complex shapes are impossible to model, since it would lead to crossing coordinate lines.
- As modelled, faults are discontinuity surfaces. In reality all faults have volume, a fault core and a surrounding damage zone. Such features can be modelled, but in this context we stick to the Eclipse concept.

Ex. 3. Shaly Layer

A non-permeable layer is a non-flow unit, and inactive in the grid. As such, the layer does not need to be modelled at all. Eclipse allows for gaps in the grid, i.e. the top of one cell does not need to coincide with the bottom of the cell above[3]. Although such explicit modelling of gaps is rather rare in practice, it is an example that the grid doesn't have to be continuous vertically, and hence the generality of all eight corners being handled individually is actually required (Figure 8).

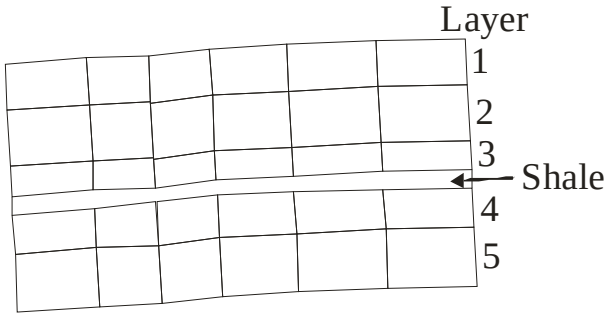


Figure 8. A shaly layer modelled as a gap (non-grid) in a corner point grid

Degenerate cells

As defined, each cell in a corner-point grid is defined by its eight corners. If all corners are different, and the cell has a "normal" appearance (topologically equivalent to a cube), the cell is a standard corner-point cell. It is however permitted and useful to allow some corners to be coinciding, in which case we say that the cell is degenerate. A degenerate cell has fewer actual corners than the eight corners it is defined by. Typically, a degenerate cell will have one or more edge thicknesses equal to zero, e.g. a wedge-shaped cell used to define a pinchout zone. The most common example is probably zero-thickness cells. When a geological layer terminates, this must be modelled in a grid by defining part of the layer with cells with thickness zero, since the number of layers must be the same in the entire grid.

Defining a corner point grid in Eclipse

Eclipse always stores the grid in corner point format. I.e., on input we must either define the cell corners explicitly, or we must give sufficient grid data that eclipse can construct the corner points itself. In practice we will either define a regular simple grid as in the example in Chapter 1, or the grid will be constructed by suited software, such that the corner point data is supplied in a black box file. Hence, a user will never input corner points manually. It is, nevertheless, useful to know the format of the corner point input to Eclipse, which is defined by the two keywords `COORD` and `ZCORN`.

COORD is used to define the coordinate lines. Each coordinate line is uniquely defined by two points, and since the line cannot be horizontal, the two points are denoted *top* and *btm* (bottom). If the number of cells in *x* and *y*-direction are *NX* and *NY*, the number of coordinate lines will be *NX+1* and *NY+1*.

[2] Actually, when Eclipse reads the grid, it doesn't check that the coordinate lines are really straight. So if the grid could be constructed by non-eclipse software that permits listric faults, and if the resulting grid was edible by Eclipse, then Eclipse could be tricked to handle quite general grids. This is far beyond the scope of these notes.
[3] Eclipse doesn't even object to overlaps, although this is obviously physically impossible.

The coordinate lines are then input in the book page format, line by line, with *I* running from 1 to *NX+1* on each line:

```
COORD
 x(1,1)TOP y(1,1)TOP z(1,1)TOP x(1,1)BTM y(1,1)BTM z(1,1)BTM
 x(2,1)TOP y(2,1)TOP z(2,1)TOP x(2,1)BTM y(2,1)BTM z(2,1)BTM
 x(3,1)TOP y(3,1)TOP z(3,1)TOP x(3,1)BTM y31,1)BTM z(3,1)BTM
  .
  .
  .
 x(NX+1,NY+1)TOP y(NX+1,NY+1)TOP z(NX+1,NY+1)TOP
 x(NX+1,NY+1)BTM y(NX+1,NY+1)BTM z(NX+1,NY+1)BTM
```

Note that no cell corners have been defined by the COORD keyword, so far only a family of lines where corners are allowed is all we have. Corner depths are defined by the keyword ZCORN. The intersection between a (non-horizontal) coordinate line and a depth value is unique, such that from coordinate lines and corner depths, all coordinates can be calculated. It would perhaps seem natural to define corners cell by cell, but Eclipse sticks strictly to the book page format, so (with *x* pointing east, *y* south) first the top northern edge of all cells are read from west to east, then the southern edge, then advancing to next row of cells. When the top of a layer has been read, the bottom is read in a similar fashion, and then we are ready for the next layer. Recall that corners associated with the same coordinate line may have different depths, but are equal in continuous areas, such that all corners must be defined – nothing is "implicitly assumed". When explaining the syntax we will use indices *NW*, *NE*, *SW*, *SE* to denote the corners (assuming a standard orientation of the grid), and *T*, *B* for Top, Bottom.

```
ZCORN
z(1,1,1)T,NW z(1,1,1)T,NE z(2,1,1)T,NW z(2,1,1)T,NE ... z(NX,1,1)T,NE
z(1,1,1)T,SW z(1,1,1)T,SE z(2,1,1)T,SW z(2,1,1)T,SE ... z(NX,1,1)T,SE
z(1,2,1)T,NW z(1,2,1)T,NE z(2,2,1)T,NW z(2,2,1)T,NE ... z(NX,2,1)T,NE
z(1,2,1)T,SW z(1,2,1)T,SE z(2,2,1)T,SW z(2,1,1)T,SE ... z(NX,2,1)T,SE
.
.
.
z(1,NY,1)T,SW z(1,NY,1)T,SE z(2,NY,1)T,SW z(2,NY,1)T,SE ... z(NX,NY,1)T,SE
z(1,1,1)B,NW z(1,1,1)B,NE z(2,1,1)B,NW z(2,1,1)B,NE ... z(NX,1,1)B,NE
.
.
.
z(1,NY,1)B,SW z(1,NY,1)B,SE z(2,NY,1)B,SW z(2,NY,1)B,SE ... z(NX,NY,1)B,SE
z(1,1,2)T,NW  z(1,1,2)T,NE  z(2,1,2)T,NW  z(2,1,2)T,NE ...  z(NX,1,2)T,NE
.
.
.
z(1,NY,NZ)T,SW z(1,NY,NZ)T,SE z(2,NY,NZ)T,SW z(2,NY,NZ)T,SE ... z(NX,NY,NZ)T,SE
```

Moderately complex grids – FILL

As noted, most or all complex grids for real field simulations are constructed with dedicated software like FloGrid or IRAP RMS. Simple cartesian grids can easily be defined in the GRID section of the data file. Often we are however faced with the challenge of constructing a grid which is not as complex as the real-case grids, but still too complex to be defined with the available keywords in the GRID section. For such cases Schlumberger offers a program called FILL, which basically is a program that constructs a complete grid including petrophysics by interpolation of sparse data. If combined with a spreadsheet like EXCEL, quite interesting grids can be constructed with FILL. FILL is considered obsolete, and probably no longer supported by Schlumberger, but old versions are always available. Ref. Eclipse documentation for further information.

4. Petrophysics (GRID section)

Petrophysics input to Eclipse is porosity, net-to-gross ratios, and three diagonal components of the permeability tensor. The general syntax is defined in the basic data input example, one value for each cell. (An easier and more flexible format will be defined in chapter 10).
The values should be "representative average values" for the cell volume. For porosity and net-to-gross ratios the simple arithmetic average is the best we can use, but for permeability the best solution is not that obvious.

Average permeability

The question is, if permeability varies with x in the interval (x_0, x_1), $K = K(x)$, is it possible to define a constant representative permeability, $K = K^*$, such that flow between x_0 and x_1 is the same if we replace the actual variation $K(x)$ with the constant K^*?

To gain insight in this problem we simplify it: Assume one-dimensional incompressible stationary one-phase flow. The governing equations are,

Darcy's law: $$u = -\frac{K}{\mu}\frac{\partial p}{\partial x} \tag{1}$$

Conservation law: $$\frac{\partial}{\partial x}\left(\frac{\rho K}{\mu}\frac{\partial p}{\partial x}\right) = \frac{\partial}{\partial t}(\rho\phi) = 0 \tag{2}$$

(The last term vanishes since stationary flow has been assumed)
Notation is standard, u is Darcy velocity, K permeability, p fluid pressure, μ viscosity (assumed constant), ρ the constant density, and ϕ porosity.

Omitting constants in Eq. (2) we have: $$\frac{d}{dx}\left(K(x)\frac{dp}{dx}\right) = 0\,, \tag{3}$$

an equation that is easily solved:

$$\left(K(x)\frac{dp}{dx}\right) = C_1 = -\mu u \Rightarrow \frac{dp}{dx} = -\frac{\mu u}{K(x)} \Rightarrow \int_{x_0}^{x_1} dp = -\mu u \int_{x_0}^{x_1} \frac{dx}{K(x)} \tag{4}$$

(C_1 determined from Eq. (1)).

The fluid flows from x_0 to x_1 such that the end point pressures are $p_0 = p(x_0)$, $p_1 = p(x_1)$.

Then from Eq. (4):

$$p_1 - p_0 = -\mu u \int_{x_0}^{x_1} \frac{dx}{K(x)}\,. \tag{5}$$

Replacing $K(x)$ by the constant permeability K^* in Eq. (5) gives:

$$p_1 - p_0 = -\mu u \int_{x_0}^{x_1} \frac{dx}{K^*} = -\mu u \frac{x_1 - x_0}{K^*} \tag{6}$$

If K^* can be used in place of the actual variation $K(x)$ the end point pressures p_0 and p_1 must be the same in equations (5) and (6). Hence,

$$p_1 - p_0 = -\mu u \int_{x_0}^{x_1} \frac{dx}{K(x)} = -\mu u \frac{x_1 - x_0}{K^*} \Rightarrow \frac{x_1 - x_0}{K^*} = \int_{x_0}^{x_1} \frac{dx}{K(x)} \tag{7}$$

Letting x_0 and x_1 be the cell edges, we see that the pressure drop across the cell is unaltered if we replace $K(x)$ with K^*.

K^* is called the *harmonic average* of $K(x)$, denoted by $\langle K \rangle_H$:

$$\frac{x_1 - x_0}{< K >_H} = \int_{x_0}^{x_1} \frac{dx}{K(x)} \tag{8}$$

In practice, we often regard the permeability as piecewise constant. Let the interval [a, b] be subdivided into N subintervals, such that $K = K_i$ for x in subinterval i, with length L_i. Then the integral (8) computes to:

$$\frac{b-a}{< K >_H} = \frac{L_1}{K_1} + \frac{L_2}{K_2} + \cdots + \frac{L_N}{K_N} = \sum_{i=1}^{N} \frac{L_i}{K_i} \tag{9}$$

We have shown that when the flow is in the same direction as the permeability variation, the appropriate average permeability to use is the harmonic average. It would however be wrong to conclude that the harmonic average is always the best to use. This can be illustrated with some examples.

Example 1. Non-permeable slab.
Assume a unit cube of porous material, with isotropic and constant permeability K = 1000mD, except for a thin slab of non-porous soil (K = 0) in the volume $0.5 \le x \le 0.51$, all y and z
Then flow in the x-direction will be completely stopped by the slab, such that the effective permeability across the cube in the x-direction is zero, which is also what we would get if we compute the harmonic average.
In the y and z-directions, the flow would, however, hardly be affected by the slab. Since the flow is along the slab, not across it, the slab will only represent a minor reduction of permeability. In this case the *arithmetic average* is the most appropriate:

$$< K >_A = \frac{1}{L} \sum_i L_i K_i = 0.5 * 1000 + 0.01 * 0 + 0.49 * 1000 = 990 mD$$

Example 2. A fracture
A rectangular grid block is 100m x 100m x 5m. A 1m wide fracture runs along the entire cell in the y-direction at x = 50m. The fracture also covers the whole cell thickness. Inside the fracture the permeability is very high, 10000mD, while the non-fractured porous media ("matrix") has a low permeability of 2mD. Then for flow across the cell normal to the fracture (in the x-direction) we use the harmonic average (Equation (9)):

$$\frac{100m}{< K >_H} = \frac{99m}{2mD} + \frac{1m}{10000mD} = 49.5001 \frac{m}{mD} \Rightarrow < K >_H = 2.02 mD$$

For flow parallel to the fracture we use the arithmetic average:

$$< K >_A = \frac{99m * 2mD + 1m * 10000mD}{100m} = 102 mD$$

So for this particular example we should define the grid cell permeability as anisotropic, with K_x = 2mD, and K_y = 102mD. I.e. the flow across the fracture is only insignificantly influenced by the fracture, while the flow in the fracture direction is strongly influenced.

Since real reservoirs have been deposited in a layer by layer fashion (roughly) and been compressed vertically, the typical reservoir has significant permeability variation vertically, but less along the layers. Also the vertical permeability will normally be lower than the horizontal, both locally due to the compression direction, and more noticeable on a larger scale, since low permeable layers (shale, clay, ...) will almost always be present. As a rule of thumb, grid cell permeabilities can be obtained from log data (measurements along a well path) by using the harmonic average in the vertical direction, and arithmetic averages in the horizontal directions (along the bedding planes).

If the permeability variation within a grid cell has no preferred direction, it has been shown that the best average value to use is the geometric average: With the same notation as above, assuming all interval lengths are equal,

$$< K >_G = \left(\prod_{i=1}^{N} K_i \right)^{1/N} \qquad (10)$$

Transmissibility

Permeability is a measure for conductance, and its inverse is a measure for resistance. If we draw the comparison to electrical resistance or conductance, it is evident that we never speak of "resistance in a point". Resistance or conductance is a property that only has meaning if referred to as "between two points", i.e. conductance and resistance is a flow property. In the same manner, it is meaningless to speak of "permeability in a point". If we regard the grid cell as a volume, we could interpret permeability as the flow property across the cell. However, viewing the cell as a volume is solely a convenient visualisation means. By the manner the simulator solves the flow equations the grid is a set of points, and all property values are to be understood as *the values in the points*. And obviously nothing can vary "in a point". (By this way of looking at the grid we are less tempted to think that it is possible to model variations within a grid cell volume. If the point of interest is really a point it is evident that no variation can be possible.) The bottom line is that the permeabilities of interest should really be measures for conductivity between different grid points.

Finite differences

In numerical methods, differentials are replaced by finite difference approximations. Since $\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x}$, we expect $\frac{\Delta f}{\Delta x}$ to be a good approximation to $\frac{df}{dx}$ when Δx is small.

E.g. to solve the differential equation $f(x) + \frac{df}{dx} = 0$ on the interval $0 \le x \le 1$, we subdivide the interval into N subintervals of equal length $h = 1/N$.

Let $x_i = ih$, and $f_i = f(x_i)$. Then an approximation to the differential equation at the point $x = x_i$ is,

$f_i + \frac{\Delta f(x_i)}{h} = 0$. We approximate $\Delta f(x_i)$ with the "forward difference", $\Delta f(x_i) \approx f(x_{i+1}) - f(x_i)$, such that the difference equation becomes,

$$f_i + \frac{f_{i+1} - f_i}{h} = 0 \Rightarrow f_{i+1} = (1-h) f_i, \text{ for } i=1,2,...,N.$$

From this last equation we can compute all the f_i.

In this context we are especially interested in the transport term for phase l (= *o,w,g;* oil water or gas) in the flow equations,

$$\nabla \cdot \left(\frac{K \cdot k_{rl}}{\mu_l B_l} \nabla p_l \right),$$

where k_{rl} is relative permeability, and μ_l and B_l are viscosity and volume factor for phase l.

Introducing *mobility* $\lambda = \frac{k_r}{\mu B}$ (where the phase subscript has been omitted for simplicity), and restricting ourselves to one-dimensional flow, this term becomes,

$$\frac{d}{dx}\left(K\lambda \frac{dp}{dx} \right) \qquad (11)$$

We seek a finite difference approximation to the term (11).

From numerical analysis it is known that the central difference $\Delta f_i = f_{i+1} - f_{i-1}$ "works better" than forward or backward differences[4]. When we firstly discretise the inner differential, for reasons that will become clear eventually, we will not use the points x_{i+1} and x_{i-1}, but intermediate points $x_{i+1/2}$ midway between x_i and x_{i+1}, and $x_{i-1/2}$ midway between x_i and x_{i-1}.

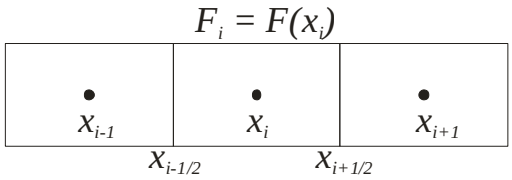


Figure 9. Discretisation notation

Using a temporary shorthand notation $F = K\lambda \frac{dp}{dx}$, the central finite difference approximation is,

$$\frac{d}{dx}\left(K\lambda \frac{dp}{dx} \right)_i = \frac{d}{dx} F_i \approx \frac{F_{i+1/2} - F_{i-1/2}}{x_{i+1/2} - x_{i-1/2}} \tag{12}$$

Now F in the "half-points" can be evaluated by a central difference in exactly the same manner:

$$F_{i+1/2} = (K\lambda)_{i+1/2} \left(\frac{dp}{dx} \right)_{i+1/2} \quad \text{and} \quad \left(\frac{dp}{dx} \right)_{i+1/2} \approx \frac{p_{i+1} - p_i}{x_{i+1} - x_i}$$

Similarly, $\left(\frac{dp}{dx} \right)_{i-1/2} \approx \frac{p_i - p_{i-1}}{x_i - x_{i-1}}$, so that we finally get,

$$\frac{d}{dx}\left(K\lambda \frac{dp}{dx} \right)_i \approx \frac{1}{\Delta x_i} \left\{ (K\lambda)_{i+1/2} \left(\frac{p_{i+1} - p_i}{x_{i+1} - x_i} \right) - (K\lambda)_{i-1/2} \left(\frac{p_i - p_{i-1}}{x_i - x_{i-1}} \right) \right\} \tag{13}$$

Using Equation (9) we see that the permeability in the "half points" should be evaluated as,

$$\frac{x_{i+1} - x_i}{K_{i+1/2}} = \frac{x_{i+1/2} - x_i}{K_i} + \frac{x_{i+1} - x_{i+1/2}}{K_{i+1}}$$

Equation (13) includes permeability and mobility evaluated at the "half points". For practical purposes we can interpret the "half points" as lying on cell edges, although this is only true if the grid is uniform. Hence, the permeability values needed by the simulator (at "half-points") must be computed from two and two cell values, and is interpreted as conductivity between the two cell centres. Pressure is, on the other hand, computed at the cell centres, all this in accordance with the introductory discussion.

Actually, the permeability cell values in the input data file are never used by the simulator at all. During the initialisation phase, these values are converted to the "half-point" inter-cell values, and in computer memory these are stored by overwriting the original permeability values, which hence are inaccessible during the simulation.

The representative permeability between two cell centres is called the *transmissibility* between the cells. In practice, all simulators include some geometric factors and in some cases also fluid properties in the definition of transmissibilities, but in this context we shall refer to the transmissibility as simply the harmonic averaged permeability between two cell centres[5].

> Be aware that this is *not* in accordance with Eclipse's way of defining transmissibility (consult manual / technical appendices), and if the user wants to define Eclipse transmissibilities explicitly Eclipse documentation should be consulted.

[4] The central difference is a second order approximation, i.e. the error is of order h^2, while the forwards and backwards approximations are of first order, with error of order h.

[5] Since the detail definition of transmissibility differs from simulator to simulator, it is unfortunately not possible to transfer transmissibilities determined by e.g. history matching from one simulator to another.

The mobility term – upstream weighting

The other "troublesome" term, $\lambda_{i+1/2} = \lambda(p_{i+1/2}, S_{i+1/2})$ must be handled differently. The details are beyond the scope of these notes, but averaging values for pressure and saturations from two neighbour cells is not a good solution. By arguing physically that mobility is a property that moves with the flow, λ is computed based on the values in the cell the flow is coming from to reach the cell edge. Hence for a one-dimensional case,

$$\lambda_{i+1/2} = \begin{cases} \lambda_i & if \ u \rightarrow \\ \lambda_{i+1} & if \ u \leftarrow \end{cases}$$

This is called upstream weighting.
(And yes, this can and does pose problems frequently.)

Averaging – smoothing

When we compute the cell average permeability, the actual permeability field is smoothened by the averaging, and the effect of local extreme values may be lost. When the simulator converts the input permeability to inter-cell transmissibilities, the data is smoothened yet another time. The permeability field that is used by the simulator is therefore typically considerably smoother than the original data. In some cases this effect can change the flow pattern such that we don't get the results that were expected by logical physical reasoning. In such cases it is recommended to compute improved transmissibilities manually, and replace simulator-computed values by these in critical areas.

Inactive cells

An inactive cell is a cell that does not contribute to fluid volumes or flow in the reservoir. By default, the simulator will flag any cell with vanishing pore volume as inactive. In addition, the user may define active cells explicitly, by the keyword ACTNUM, where all cells are listed, the active ones as "1", inactive as "0". Note that a cell with zero pore volume will be inactive irrespective of the value it receives in ACTNUM.

Another source for inactive cells is the *minimum pore volume* option. Cells with very small pore volume will contribute insignificantly to flow or production, but exactly because they have a pore volume which is very different from typical values they can have a negative effect on the numerical solution, and total computing time can become (unacceptable) large. For this reason it is possible and advisable to define a minimum pore volume value such that cells with smaller pore volume than this value will be flagged as inactive. For Eclipse, see keywords MINPV or MINPVV.

An inactive cell between two active cells will be a flow barrier. But if a cell becomes inactive by the minimum pore volume option, it would obviously be wrong to treat it as a barrier, which would not be in accordance with the original layout. Eclipse therefore has a keyword PINCH which when used causes correct transmissibilities to be computed across cells made inactive by small pore volume.

> Note that in this case the transmissibilities are not between neighbour cells, which they don't need to be. General transmissibilities can be between any cells.

5. Fluid properties (PROPS section)

Tables in Eclipse

All functional dependencies are defined by tables in Eclipse. Values between tabulated points are found by linear interpolation when needed. If values extend beyond the range in the table, the first or last point in the table is used, i.e. *no trend extrapolation*. Note that in many cases Eclipse actually needs differentials calculated from the tables, hence not only should table data be sufficiently smooth, but also the derivatives. With this background we can put up some simple rules of thumb,

- Ensure that the table range covers all values that can occur in the simulation. If in doubt, extrapolate table roughly by hand (it's going to be better than Eclipse's constant extension).
- Use enough points that transitions between table points are "sufficiently smooth". Avoid abrupt changes, e.g. don't use a step function, but define some extra points which round off the corners a little. Since Eclipse interpolates linearly, check that the point spacing is dense enough that this is good enough. On the other hand don't use too many points, as this may slow down the table look-up.

- Hysteresis will not be covered here, but if used it is recommended to define the two curves at a parting point such that the parting is smooth, *with smooth derivatives*.

In most cases the functional dependency that is tabulated has some physical constraints. Eclipse knows all about such constraints, and will check tables, and refuse to simulate an unphysical situation. (Typical monotonicity, consistency,...). Some violations will result in warnings, other in errors.

Relative permeability and Capillary Pressure

Two-phase curves (water – oil)

The standard way of defining relative permeability for an oil-water system is as a function of water saturation, $k_{rl} = k_{rl}(S_w)$, $l = o,w$. The interesting range is the mobile fluid range,
$0 \le S_{wc} \le S_w \le 1 - S_{or} \le 1$, where S_{wc} is critical water saturation and S_{or} residual oil saturation.
The endpoint relative permeabilities are,

$$k'_{ro} = k_{ro}(S_{wc}),\ k'_{rw} = k_{rw}(1 - S_{or})$$

The easiest way to input the curves to Eclipse is by the keyword SWOF. Each entry (line in the table) consists of four items,
S_w $k_{rw}(S_w)$ $k_{ro}(S_w)$ $P_{cwo}(S_w)$

Requirements:
Column 1 is in increasing order.
Column 2 is non-decreasing
Column 3 is non-increasing
Column 4 is non-increasing

In the first column, the first entry is taken as connate water, and k_{rw} must be zero on this line, while $k_{ro} = k_{ro}'$.

$S_w = 1 - S_{or}$ at the first zero-value in the third column, while the last entry in the first column is taken as maximum occurring water saturation, $S_{w,max}$.

NOTE: $S_{w,max}$ is used to initialise saturations in the water zone in the reservoir. Only in exceptional cases should this value be different from 1.0.

Some entries in columns 2, 3, and 4 can be defaulted (by 1*). Missing table values will then be computed by linear interpolation.

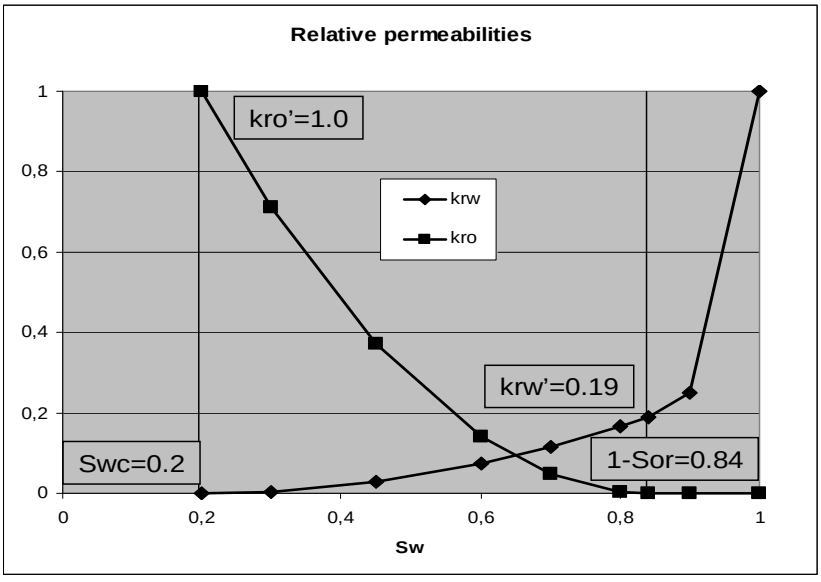


Figure 10. Typical two-phase relative permeability curves

Example 1
The relative permeability curves shown in Figure 10 can be input to Eclipse as,

```
SWOF
-- Sw    krw     kro     Pcwo
   0.2   0.0     1.0    10.0
   0.3   0.0046  0.712   2.7
   0.45  0.029   0.371   0.78
   0.6   0.074   0.141   0.3
   0.7   0.116   0.0479  0.14
   0.8   0.167   0.004   0.033
   0.84  0.19    0.0     0.0
   0.9   0.25    0.0     0.0
   1.0   1.0     0.0     0.0   /
```

From the first line we deduce that $S_{wc} = 0.2$, and $k'_{ro} = 1.0$.
The first entry where k_{ro} is 0 is at $S_w = 0.84$. Hence $1 - S_{or} = 0.84$ ($S_{or} = 0.16$), and $k'_{rw} = 0.19$. The last table entry is $S_w = S_{w,max} = 1.0$, so the water saturation in the water zone will be 1.0.

Note that we have defined a point between $1 - S_{or}$ and 1.0, such that the water relative permeability is smooth near $1 - S_{or}$.
Theoretically, $P_{cwo} \to \infty$ as S_w approaches S_{wc}. This obviously can't be input in the table, so we use a large value of the capillary pressure at the end point, such that the curve is relatively smooth.

An alternative way of defining two-phase water-oil relative permeability curves is by the keywords SWFN and SOF2. The difference from SWOF is that only one curve is defined in each keyword, water relative permeability by SWFN and oil relative permeability by SOF2. Also the curves are defined with its own saturation as independent variable, $k_{rw} = k_{rw}(S_w)$ and $k_{ro} = k_{ro}(S_o)$. (Both tables defined with the first column in increasing order.) Apart from that all rules and constraints defined for SWOF are the same. (SWFN and SOF2 are not much used anymore.)

Example 2
The relative permeabilities defined in Example 1, by SWFN and SOF2 keywords,

```
SWFN
-- Sw    krw     Pcwo
   0.2   0.0     10.0
   0.3   0.0046  2.7
   0.45  0.029   0.78
   0.6   0.074   0.3
   0.7   0.116   0.14
   0.8   0.167   0.033
   0.84  0.19    0.0
   0.9   0.25    0.0
   1.0   1.0     0.0   /

SOF2
-- So    kro
   0.0   0.0
   0.16  0.0
   0.2   0.004
   0.3   0.0479
   0.4   0.141
   0.55  0.371
   0.7   0.712
   0.8   1.0   /
```

Since we now have two tables, they must be compatible: the endpoint water saturations in SWFN (0.2 and 0.84) must correspond to the endpoint oil saturations in SOF2 (0.16 and 0.8).

Three-phase relative permeabilities

If we assume that any of the three phases can flow irrespective of the saturation of the two other phases, we will need a very complex description, that is not even agreed on in the industry. We therefore restrict ourselves to a simple model, and refer to other literature if a more realistic model is needed. The basic assumption is that the flow is in essence two-phase, with the third phase acting as a "dummy background", but not contributing actively to the flow. Then the relative permeability curves for oil are either oil-water (no gas) or oil-gas (water at connate saturation). The input can be defined in several ways in Eclipse. We present only one, and refer to the manuals for alternatives.
Three phase relative permeabilities are input by the keywords SWOF and SGOF.
SWOF is identical to the two-phase case, except that column 3, k_{row}, is now interpreted as oil relative permeability when only oil and water are present,
S_w $k_{rw}(S_w)$ $k_{row}(S_w; S_g=0)$ $P_{cwo}(S_w)$

The data for keyword SGOF also consist of lines with four entries:
S_g $k_{rg}(S_g)$ $k_{rog}(S_g; S_w=S_{wc})$ $P_{cog}(S_g)$

We see that the first entry in column 3 is oil relative permeability at minimum water and gas saturation in both tables, so those two have to be equal. The last value in column 3 must be zero for both keywords. $P_{cog}(S_g)$ must be nondecreasing.

Example 3. Three phase relative permeabilities using SWOF and SGOF

```
SWOF
-- Sw   Krw   Krow   Pcow
  0.22  0.0    1.0    7.0
  0.30  0.07   0.4    4.0
  0.40  0.15   0.125  3.0
  0.50  1*     0.065  1*
  0.60  0.33   0.0048 2.0
  0.80  0.65   0.0    1.0
  0.90  0.83   0.0    1*
  1.00  1.00   0.0    0.0   /

SGOF
-- Sg   Krg    Krog   Pcog
  0.00  0.0    1.00    0.0
  0.04  0.0    0.60    0.2
  0.10  0.022  0.33    0.5
  0.20  0.1    0.10    1.0
  0.30  0.24   0.02    1*
  0.40  1*     0.00    1*
  0.50  0.42   0.00    1*
  0.60  0.5    0.00    3.0
  0.70  0.813  0.00    3.5
  0.78  1.0    0.00    3.9  /
```

PVT data

Eclipse, as most black oil simulators, assumes isothermal behaviour in the reservoir, i.e. the PVT relations are defined for constant reservoir temperature. This may be far from reality, as e.g. when injecting cold water into a warm reservoir, but this is a limitation we have to live with.

Although the simulator must relate to the fluids as they occur in the reservoir, we as observers can only measure fluid properties at the surface. It is therefore convenient, and standard in all reservoir simulators to define fluid properties at standard conditions (also called surface conditions, or stock tank conditions), i.e. at atmospheric pressure and temperature 15.5 °C. We will use subscript *RC* for *Reservoir Conditions* and subscript *SC* for *Standard Conditions*.

Densities are then supplied to Eclipse as standard condition values (as in the DENSITY keyword in the "BASIC data input example"), and transforming these values to reservoir conditions is done by the formation volume factors, which is the ratio of a unit volume of fluid at reservoir conditions to the volume the same amount of fluid occupies at standard conditions,

$B_l = \frac{V_{l,RC}}{V_{l,SC}}$, where *l* denotes gas, oil or water.

The unit for B_l is Rm^3/Sm^3, reservoir cube metres on standard cube metres.

The dissolved gas-oil-ratio, R_s, is defined as the gas volume, measured at standard conditions, that can be dissolved in one standard conditions unit volume of oil, if both fluids are taken to reservoir conditions.

Then, since a volume of oil at reservoir conditions contains both pure (liquid) oil and dissolved gas, the formation volume factors can be expressed by the fluid densities as,

$$B_w = \frac{\rho_{w,SC}}{\rho_{w,RC}}$$

$$B_g = \frac{\rho_{g,SC}}{\rho_{g,RC}}$$

$$B_o = \frac{\rho_{o,SC} + R_s \rho_{g,SC}}{\rho_{o,RC}}$$

Oil will behave very differently depending on whether the oil pressure is above or below the bubble point. If the oil is at a higher pressure than the bubble point pressure, no gas will evaporate from the oil, and the oil behaves as a "pure" fluid, similar to water. Oil in this regime is called *dead oil* or *undersaturated oil*, while oil below the bubble point is called *live*, or *saturated oil*.
For gas, water or dead oil, increasing the fluid pressure results in compressing the fluid – it becomes heavier. Hence the volume formation factor will be decreasing with increasing pressure for such fluids. Also fluid viscosity must be expected to be non-decreasing as the fluid becomes heavier. When oil pressure is reduced below the bubble point, the behaviour is less predictable, as the oil will become heavier due to gas release, but lighter due to expansion.

Water

In the black oil model, water is pure water, and does not mix with oil or contain dissolved gas. Further, since water compressibility is low, a constant compressibility coefficient is assumed.

The coefficient of compressibility is defined as,

$$C_w = -\frac{1}{B_w}\frac{dB_w}{dp} \qquad (14)$$

When C_w is constant, equation (14) can be solved easily:

$$C_w dp = -\frac{dB_w}{B_w} = -d\log B_w \Rightarrow C_w \Delta p = \log\frac{B_w^0}{B_w} \Rightarrow$$

$$B_w = B_w^0 \exp[-C_w(p - p^0)]$$

From this expression, the volume formation factor can be calculated at any pressure from its value at a reference pressure p^0.

Water viscosity can safely be assumed constant. Hence, Eclipse only needs water PVT data at a reference pressure P_{ref}.
Eclipse syntax:

PVTW

P_{ref} [bars] $B_w(P_{ref})$ [Rm^3/Sm^3] C_w [$bars^{-1}$] $\mu_w(P_{ref})$ [cP] C_v

The term C_ν is called “viscosibility”, and is defined as

$$C_\nu = \frac{1}{\mu_w}\frac{d\mu_w}{dp}$$

It can safely be defaulted in almost all runs.

Example:

```
PVTW
-- Pref   Bw(Pref)   Cw    mu_w(Pref)  C_nu
   245.0   1.0035  2.0e-5   0.525       1*   /
```

Dead Oil

If the oil pressure stays above the bubble point pressure everywhere in the reservoir at all times, a simplified version of oil PVT data can be defined. In the permitted pressure regime, no gas is liberated from the oil in the reservoir, and hence the oil behaves as a pure liquid. Eclipse uses a special keyword for this situation, PVDO (PVT for Dead Oil). The table consists of three columns, with syntax for each line,

P_o [bars] $B_o(P_o)$ [Rm^3/Sm^3] $\mu_o(P_o)$ [cP]

P_o must be increasing down the column
B_o must be decreasing down the column
μ_o must be non-decreasing down the column

Example
(See also Figure 11)

```
PVDO
-- Po   Bo    mu_o
  27.2 1.012  1.160
  81.6 1.004  1.164
 136.0 0.996  1.167
 190.5 0.988  1.172
 245.0 0.9802 1.177
 300.0 0.9724 1.181
 353.0 0.9646 1.185  /
```

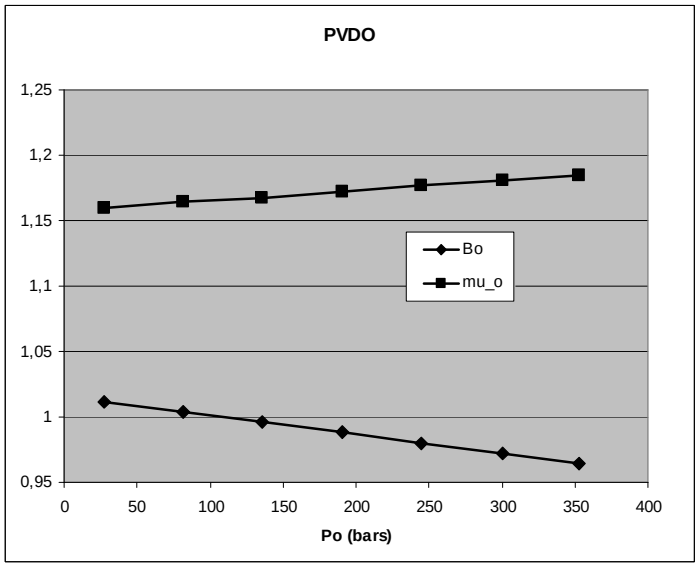


Figure 11. Volume factor and viscosity, dead oil (from example PVDO table)

For dead oil no gas vaporises from the oil in the reservoir. When the oil flows in a producing well to the surface, the oil pressure will normally be far below bubble point when the oil arrives at the

platform. The released gas volume pr. unit oil volume will however be constant, since all oil departed the reservoir as dead oil. The amount of released gas which must be accounted for on the surface can therefore be specified as a constant, which must be supplied to Eclipse in dead oil runs,

RSCONST
R_s [Sm^3/Sm^3] P_{BP} [bars]

Note that if the pressure in any grid block falls below the bubble point pressure P_{BP}, the run will terminate.

Example

```
RSCONST
--  Rs    P_BP
   180    230    /
```

Dry Gas

Gas which does not contain vaporised oil is denoted *dry gas*, and is the only kind of gas we will look at in these notes.

PVT data for dry gas are equivalent to PVT data for dead oil, and the table syntax in Eclipse is equal: Each line,

P_g [bars] $B_g(P_g)$ [Rm^3/Sm^3] $\mu_g(P_g)$ [cP]

P_g must be increasing down the column
B_g must be decreasing down the column
μ_g must be non-decreasing down the column

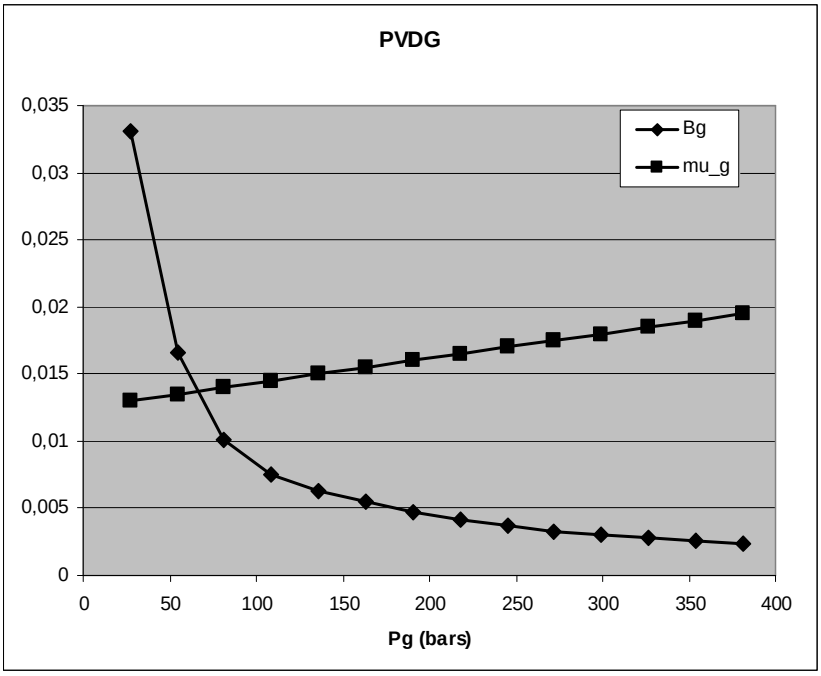


Figure 12. Volume factor and viscosity for dry gas (from example PVDG table)

Example
(see also Figure 12)

```
PVDG
-- Pg      Bg        mu_g
  27.2  0.03313   0.0130
  54.4  0.016564  0.0135
  81.6  0.010051  0.0140
 108.8  0.007500  0.0145
 136.0  0.006255  0.0150
 163.2  0.005503  0.0155
 190.4  0.004716  0.0160
 217.6  0.004155  0.0165
 245.0  0.00365   0.0170
 272.0  0.00328   0.0175
 299.2  0.00303   0.0180
 326.4  0.00275   0.0185
 353.6  0.00253   0.0190
 380.8  0.00236   0.0195  /
```

Live Oil

An undersaturated oil has a well defined bubble point pressure P_{BP}. If the oil pressure is reduced to P_{BP} the oil starts to boil, and gas evaporates from the oil. But once this process starts, the oil itself changes – when some gas has been liberated from the oil, the remaining liquid oil contains less dissolved gas than before the evaporation, such that the oil is heavier, and with a new, lower bubble point than originally. Hence, PVT relationships for live oil must describe both how the oil itself changes, and for "each type of oil" how properties vary with primarily oil pressure. As long as the "oil type" is constant the property variation is similar to that of dead oil, described by the PVDO keyword. Whereas the bubble point is constant for dead oil, for live oil it is directly related to the "oil type", which can be characterized by the gas resolution factor R_s. Hence we must specify the relationship between R_s and P_{BP}, as well as variation of oil properties – typically the liquid oil grows heavier and more viscous as the amount of dissolved gas is reduced. The table can use either P_{BP} or R_s as primary variable, in Eclipse R_s has been chosen.

The table contains two types of data,

1. Data for saturated oil, tabulated as a function of R_s. Each line contains PVT data relating to the supplied R_s –value.
2. Data for undersaturated oil, valid for a certain value of R_s, and tabulated as a function of oil pressure. This data *must* be defined for the largest R_s –value in the table, and can also be defined for other R_s –values.

One record in the table is comprised of four items if data is of the first kind, and an additional sub-table if it is of the second kind. Each record is terminated by a slash.

Data of the first kind are given as,

R_s [Sm^3/Sm^3] $P_{BP}(R_s)$ [bars] $B_o(P_{BP})$ [Rm^3/Sm^3] $\mu_o(P_{BP})$ [cP]

The R_s –values should be increasing within the table.

When data of the second kind are supplied, the record takes the form,

R_s [Sm^3/Sm^3] $P_{BP}(R_s)$ [bars] $B_o(P_{BP})$ [Rm^3/Sm^3] $\mu_o(P_{BP})$ [cP]
P_o [bars] $B_o(R_s, P_o)$ [Rm^3/Sm^3] $\mu_o(R_s, P_o)$ [cP]
.
.
.
P_o [bars] $B_o(R_s, P_o)$ [Rm^3/Sm^3] $\mu_o(R_s, P_o)$ [cP]

I.e. the first four items are defined as above, and then additional data are defined, tabulated as a function of P_o (which must be defined in increasing order, starting with P_{BP}), all valid for the current value of R_s.

Example
(See also Figure 13; viscosity is not shown as it behaves as in the dead oil regime)

```
PVTO
-- Rs P_bp(Rs) Bo(P_bp) mu_o(P_bp)
  49.0  27.2    1.1334    1.17     /
 108.6  81.6    1.1626    1.11     /
 167.1 136.0    1.1906    1.06     /
 220.8 190.4    1.2174    1.00     /
 267.2 245.0    1.2432    0.95
-- Subtable for undersaturated oil with Rs = 267.2
--      Po       Bo(Po)   mu_o(P_o)
       272.0    1.2382    0.95
       299.2    1.2332    0.95
       326.4    1.2283    0.95
       353.6    1.2235    0.95
       380.8    1.2186    0.95     /  Record end
 286.7 272.0    1.2544    0.94     /
 306.3 299.2    1.2656    0.92
-- Subtable for undersaturated oil with Rs = 306.3
       326.4    1.2606    0.92
       353.6    1.2555    0.92
       380.8    1.2505    0.92     /
/  "Empty" record marks end of table
```

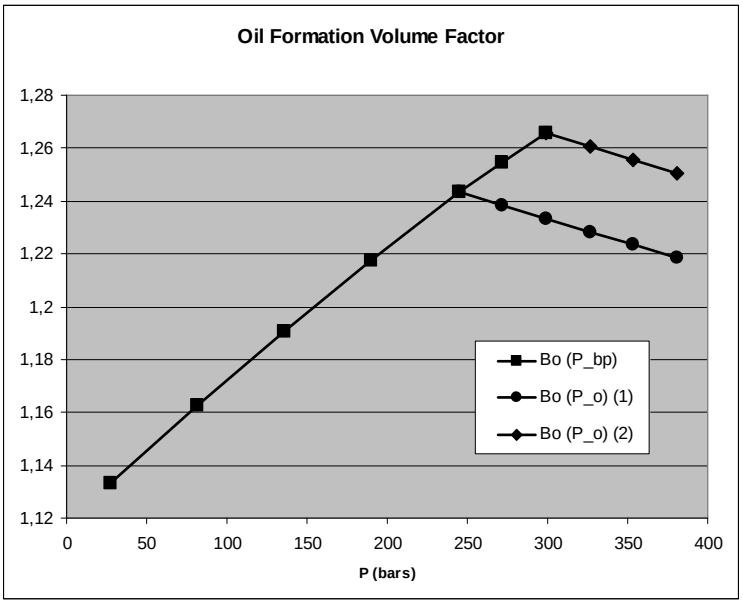


Figure 13. Variation of live oil volume factor, below bubble point and two branches above (from example PVTO table)

6. Soil compressibility (PROPS section)

In reservoir simulation soil compaction (which is actually compaction of the pore space) is modelled as a function of fluid pressure, which is a necessary simplification. The simplest available compaction model is to assume linear elasticity, i.e. a constant coefficient of rock compression (similar to the constant water compressibility discussed above), defined with keyword ROCK, with syntax,

ROCK
  P_{ref} [bars] C_r [bars^{-1}]

Example

```
ROCK
-- Pref   Cr
   310.0  5.0e-6  /
```

Then if a grid cell has an initial pore volume of $PV_0 = PV(P_{ref})$, the pore volume corresponding to cell pressure P, $PV(P)$, will be,

$$PV(P) = PV_0 \exp[C_r (P - P_{ref})]$$

Non-linear compaction with or without hysteresis can be modelled by means of the keyword ROCKTAB. The input consists of a table which specifies the pore volume multiplier (PVM) as a function of cell pressure. A permeability multiplier can also be specified. The basic syntax for each line is,

P [bars] *PVM(P)* *Perm-mult(P)*

P must be increasing down the column
Columns 2 and 3 must be non-decreasing down the columns.

In addition it may be necessary to include the ROCKCOMP keyword in the RUNSPEC section.

Example

```
ROCKTAB
-- P      PVMult Perm-mult
 100.000 0.93520   1.0
 167.854 0.95327   1.0
 203.427 0.96376   1.0
 227.475 0.97265   1.0
 251.206 0.97953   1.0
 264.976 0.98510   1.0
 280.418 0.98937   1.0
 289.369 0.99254   1.0
 293.122 0.99484   1.0
 300.651 0.99652   1.0
 305.144 0.99797   1.0
 310.000 1.00000   1.0 /
```

7. Initialisation (SOLUTION section)

At time zero, before any flow occurs in the reservoir, the saturation and pressure distribution in the reservoir must be such that the fluids are in gravity equilibrium, i.e. a no-flow situation. Although it is possible (in principle) to manually define initial grid cell values such that the no-flow condition is satisfied it is virtually impossible to do this in practice. Almost without exception, the equilibrium calculations will be performed by the simulator, which is also strongly recommended. Using Eclipse, this signifies that initialisation of the models will always be done by the keyword EQUIL. By this keyword Eclipse has sufficient data to compute capillary and fluid gradients, and hence fluid saturation densities in each cell.

The data record is comprised of one line of data values (or one for each equilibration region if needed),

DD $P(DD)$ OWC $P_{cow}(OWC)$ GOC $P_{cog}(GOC)$ $R_sP_{BP}vsD$ R_vP_dvsD N_{acc}

Datum depth

DD (Datum Depth) and *P(DD)* (Pressure at datum depth) are the fix-points in the calculations. All cell pressures will be calculated from fluid pressure gradients and the supplied pressure value at datum depth. Also, calculated pressures during the simulation can be reported as equivalent datum depth pressures. So typically datum depth will be chosen at a depth where an early pressure measurement was made. If a gas-oil contact exists Eclipse will often only accept datum depth at this contact, so it can be recommended practice to do this in general (ref. Eclipse documentation for details).

Contacts

Item 3, *OWC* is the depth of the oil-water contact in oil-water or three-phase problems, the depth of the gas-water contact in gas-water problems. (The value is ignored in single phase or gas-oil problems).
Item 5, *GOC* is the depth of the gas-oil contact (ignored in single-phase, oil-water, and gas-water problems). Capillary pressures specified at the contacts (items 4 and 6) will overrule capillary pressures defined in the relative permeability tables.
Note that if a contact is not present in the reservoir it should be defined above or below the reservoir, not defaulted. E.g. in a three-phase run with initially no free gas, the GOC should be defined at a depth above the reservoir, to assure correct fluid distribution when free gas is produced, and since Eclipse expects a gas-oil contact in a three-phase run.

$R_sP_{BP}vsD$ and R_vP_dvsD. These are flags for specifying initial variation with depth of gas solution, bubble point, vaporised oil, or dew point. Ref. Eclipse documentation for details, as the flags will not be used in these notes.

The N_{acc} parameter – accuracy of initial fluids in place calculations

The initial vertical distribution of fluids will be governed by fluid densities and capillary forces. Starting from the top and going downwards the typical distribution will be, gas, gas-oil transition zone, oil, oil-water transition, water. By initialisation we aim to define fluid saturations in each grid cell such that the system is in equilibrium, i.e. when no external forces are applied, no fluid flow should take place in the reservoir. (Appears obvious, but very often not obeyed in practice.) For simplicity we discuss only the oil-water contact, and assume no transition zone, i.e. water saturation is unity below the oil water contact and at connate water saturation above the contact. Only grid cells which contain the oil-water contact can pose problems. Recall that a grid cell is actually a point in space for the simulator, so although we can envisage a saturation variation within a cell volume, such a variation is not possible in a simulation cell – we can only assign *one* value of e.g. water saturation for the cell. So the question is, which value to choose?

The simplest and perhaps most obvious solution is to use the saturation at the cell centre, i.e.

$$S_w = \begin{cases} 1.0 & \text{if cell centre below OWC} \\ S_{wc} & \text{if cell centre above OWC} \end{cases} \qquad (15)$$

If the oil-water contact passes close to the cell centre this method will result in inaccurate fluids-in-place calculations, as the "correct" water saturations should obviously be close to the average value of connate and pure water. If a cell has total pore volume V, of which V_w is water-filled and V_o oil-filled, then the cell's initial water saturation ("exact") will be

$$S_w = \frac{1}{V}(1.0 \cdot V_w + S_{wc} \cdot V_o) \qquad (16)$$

Intuitively, calculating initial water saturations by Equation (16) will provide a more accurate initial fluid distribution and better estimates for initial fluids in place than by using Equation (15). Although Equation (16) is simple as it stands, it becomes rather complex if we try to generalise it to handle transition zones. Eclipse therefore uses an alternative approach to calculate initial cell saturations from exact fluid distribution. Imagine the cell divided into a number of horizontal slices of equal thickness. For each such slice, the slice saturation is defined as the saturation at the slice centre. Then the cell saturation is the volume-average of the slice saturations. This saturation calculation will obviously be more accurate the more slices we use. The number of such slices is defined by the N_{acc} paramter.

$N_{acc} = 0$: Use cell centre initialisation (Equation (15))
$N_{acc} > 0$: Use $2*N_{acc}$ slices and honour actual cell geometry (tilted cells)
$N_{acc} < 0$: Use $-2*N_{acc}$ slices, but treat the cells as being horizontal.

N_{acc} is limited to 20. Note that the default value is -5 (i.e. 10 slices and cells assumed horizontal), such that if cell centre initialisation is required, N_{acc} must be set explicitly to 0.

Example.
A reservoir has its highest point 1700 m SMSL (Sub mean sea level), and its deepest point at 1940 m SMSL. The reservoir was discovered by an appraisal well in which a reservoir pressure of 326 bars was measured at depth 1754 m SMSL. The oil water contact is at a depth of 1905 m, and there is no free gas present in the reservoir (so we define GOC above the reservoir).

We use the initially measured pressure as datum, and will initialise our three-phase model with the accurate fluids-in-place calculation with 20 slices, using the tilted cell option. No capillary transition zones. The appropriate EQUIL keyword is then,

```
EQUIL
-- DD    P(DD)  OWC Pcow(OWC) GOC  Pcog(GOC) RsPbpvsD RnPdvsD Nacc
  1754  326.0  1905   0      1600    0          1*      1*     10  /
```

Equilibrium – discussion – advanced issues

From a computational point of view the cell centre initialisation is the most proper option to use, at it is the only way to initialise the reservoir to complete equilibrium. The accurate fluids-in-place method will in most cases be almost in a quiescence state, but not quite, such that some fluid flow will take place if the simulation is started without any production / injection (try it!).

The full problem of proper initialisation cannot be discussed in these notes, but some basic features are presented, as they should be known by simulator users.

When building simulation models for real reservoirs in a commercial setting (i.e. in an oil company) it is normally requested that initial fluid-in-place volumes are as correct as possible, not only on reservoir scale, but also in each reservoir segment and geological unit. This can often only be achieved by using accurate fluids-in-place initialisation, so even though cell-centre initialisation has many advantages it is frequently not an option. Additionally, the initial saturation distribution in the reservoir is seldom or never as simple as a fluid contact with or without transition zone. Saturation distribution is generated from seismics and saturation vs. depth measurements in wells, and it is often required to reproduce this initial saturation field in the simulation model, although an exact reproduction will (almost) never be in equilibrium (primarily because reality includes so much physics which has been left out in the model).

The bottom line is that we often wish to initialise our simulation grid with a certain saturation field, but this results in a non-equilibrium model.

Eclipse offers some ways to come around this, by the keyword EQLOPTS. Note that these should never be used if cell centre initialisation has been chosen.
Basically, three flags can be set to control simulator behaviour:

MOBILE:

Adjusts rel-perm end points (critical saturations) such that relevant fluids are (only just) immobile at the calculated or defined initial saturations.

QUIESC:

Changes initial cell phase pressures such that a quiescent state is achieved

THPRES:

Enables the Threshold Pressure Option, which is otherwise used to prevent fluids from different equilibration regions to be in an incompatible state. In essence this option defines inter-cell transmissibilities which are just large enough to prevent flow between the cells with the existing pressure differences computed by initialisation (a sort of “pseudo-barrier”). The changes are applied throughout, not only during initialisation.

All of these flags result in a quiescent initialisation in the sense that no flow will occur if the simulation is started with no injection / production. However, no one can argue that the means have a flavour of artificialness, and are outside the control and knowledge of the user. More importantly, the simulator does change the input data, and these modifications are not only used to force a stable initialisation, but will continue to be applied throughout the run, some times with unintentional behaviour as result.

If one can live with the lack of accuracy in fluid-in-place computations it is probably the safest to stick to the simple cell-centre saturation initialisation.

8. Time dependent input data (SCHEDULE section)

So far, all the data we have defined has been static data – used to specify the model itself and initialise it. The time dependent (dynamic) data describe how the reservoir is produced. Eclipse processes the dynamic data a little different, as the input file is only read as far as necessary to compute the next time step. This can be a clue to understanding how the SCHEDULE section is organised. Basically, the section is comprised of three different kinds of data,

1. Specification of wells, and guidelines for production or injection.
2. Control of progress of the numerical computational scheme (optional)
3. Time stepping

The only way to remove or add fluids from / to the reservoir is by wells, so any simulation model will include at least one well. In reality wells can be simple or complex, and controlled by a number of devices, to meet surface constraints. Most simulators will therefore offer an extensive list of features for controlling simulator well behaviour, some of which we cover in the following section.

8.1 Well definitions and control

In Eclipse, wells are defined and controlled in three stages,

1. Well specification. Includes the static properties of the well as the well name, location, dimensions,...
2. Completion data. Although the well itself can penetrate a long part of the reservoir, it will be open for flow between well and reservoir only in limited sections (actually holes), called completions (or perforations, connections). Completion data specifies where and how the well is open for flow to / from the reservoir.
3. Production / injection control. Specification of target rates and constraints for well production or injection.

The first item is given only once for each well, and must be specified before any other well data can be defined. The second kind of data changes occasionally, while the third kind typically is updated relatively often.

In production simulation, wells are very often put under group control, which means that in stead of controlling each well individually, wells are organized in groups, and the control criteria applies to the group as such. Groups can also be organized hierarchically. Group control will however not be covered in these notes.

The well control keywords contain many flags, some of which are not needed in the simple problems we study in these notes. Such flags are marked with N/A in the following, and curious readers should consult the Eclipse documentation.

Well Specification (WELSPECS keyword)

All wells in Eclipse are referred to by their name, a name which is defined in the WELSPECS keyword. Therefore, logically, any well must be defined in WELSPECS before any further reference to the well can be done. Apart from that, the keyword can occur anywhere in the SCHEDULE section and as many times as needed.

Each well definition consists of one line terminated by a slash, and an empty record is used to terminate the keyword.

Each line is comprised of the following items,

Wname Gname I_{WH} J_{WH} Z_{BHP} *Phase N/A N/A Auto-shut-in XflowFlag N/A DensCalc ...*

Wname:

Unique well name, max 8 characters. All later reference to the well will be by Wname.

Gname:

Name of group to which the well belongs. Must be supplied even though we don't intend to use group control. The top-level group is always FIELD, but that name is reserved and cannot been used here. Advice: Use some simple dummy-name, like G1.

I_{WH} J_{WH}

I and J cell index for *well head*, typically taken as the cell where the well hits the reservoir.

Z_{BHP}

Reference depth for bottom hole pressure. The flowing bottom hole pressure, P_{BHP}, is the pressure *inside* the well bore. It varies with depth, but not that much if tubing wall friction can be neglected. It is advised to set this value to the depth of the topmost perforation, which can be done by defaulting it.

Phase

Preferred phase for the well. Can be set to OIL, WATER, GAS or LIQ. What we define here doesn't matter much in practice, since actual phase will be defined later.

Auto-shut-in

Flag for determining what to do with the well if the simulator determines to close it[6] due to some control violation. The options are, SHUT, which isolates the well completely from the reservoir, or STOP; which only plugs the well above the reservoir, and hence allows for crossflow after the well has been closed. The STOP option is normally the most realistic compared to actual field wells, but will be more calculation intensive and can cause convergence problems, a reasonable reason for choosing SHUT.

XFlowFlag

Flag for defining if crossflow is permitted through the perforations (This can and does occur in real wells, but the computational considerations discussed above are valid in this context also.) The options are, YES, crossflow allowed, NO, crossflow not allowed.

DensCalc

At this stage, can be left at default value. When friction wells are used, this item must be set to 'SEG' (chapter 15).

Note that in all cases where a character-variable flag is expected, a unique abbreviation is accepted.

Example

```
WELSPECS
-- Wname Gname IWH JWH  Z_BHP  Phase N/A  N/A ShutIn Xflow
   OP1    G1    12  23  1825.0  OIL    1*  1*  STOP   YES   /
   OP2    G1     5  17    1*    OIL    1*  1*  STOP   YES   /
   WI1    G1    19   4  1960    WAT    1*  1*  SHUT    NO   /
/
```

Well Completions (COMPDAT keyword)

The keyword is used to specify where and how the well is open for flow from / to the reservoir. Each line (record) contains data as described below. A complete description of a well's completion can and often must be defined using several records. As each record is read existing data is updated such that the most recently read specification is the valid one.
Each line is comprised of the following items,

Wnm I_C J_C $K_{C,TOP}$ $K_{C,BTM}$ Open/ShutFlag SatTblNbr N/A D_w Kh Skin N/A Dir r_0*

Wnm*

Well name as defined in WELSPECS, or well name wildcard. Eclipse will interpret a trailing * as a wildcard, e.g. OP-1* will mean all wells with well names starting with "OP-1". (Will match OP-1A, OP-11, OP-100B, OP-12,..., but not OP10). Note that the * can only be used as a wildcard at the end of Wnm*.

[6] Eclipse defines accurate usage of the terms STOP and SHUT. We will therefore use "close" when we mean either STOP or SHUT, and only use STOP / SHUT in Eclipse-correct settings

Well name wildcards are so useful that most experienced users choose the well names such that wildcards can be used in the most efficient manner.

I_C J_C

I and *J* indices for connection cells. If defaulted (by setting *I, J* to 0) the values defined in WELSPECS will be used.

$K_{C,TOP}$ $K_{C,BTM}$

K-index (layer) for topmost and bottom cell for which completions (perforations) are defined in this record. All cells in the range ($K_{C,TOP}$, $K_{C,BTM}$) will be open to flow. E.g.
I_C J_C $K_{C,TOP}$ $K_{C,BTM}$ = 5 20 6 10 means the cells,
(5, 20, 6), (5, 20, 7), (5, 20, 8), (5, 20, 9), and (5, 20, 10)
will be perforated.
Observe that the syntax is tailored for vertical wells.

Open/ShutFlag

Specifies if the connection(s) specified in this record are currently open or closed to flow. Choices are OPEN or SHUT.
(There is also a third option, AUTO. Ref. Eclipse manual)

SatTblNbr

By default Eclipse will use the relative permeability curves defined for the connection cell for well in/outflow calculations. So far we haven't even covered use of multiple relative permeability regions, but still can appreciate that there may be good reasons for defining special relative permeabilities for well flow. We contend ourselves with a simple example. Assume a well is completed down to 1960 m, with OWC at 1970 m. Further assume that in the simulation grid, both OWC and the well are in the same cell. In that case, the cell will contain mobile water such that the well will produce water immediately. Obviously in reality, the well will not produce water before the OWC has risen to the well completion level. We could model this behaviour by defining an artificially high S_{wc} in the well cell, such that the water would be seen as immobile until saturation reached a level which corresponds to the OWC having risen to the perforation.

D_w

Wellbore diameter at the connection. This property is defined here rather than in WELSPECS, to allow for the diameter to vary along the wellbore.

Kh

Permeability thickness, i.e. the product *K*h*. This value is a measure for the production (or injection) capacity of the well. The default value is to take *K* as the cell permeability and *h* as the connection length, which is the cell thickness for a vertical well. If the well is perforated across the entire cell it is normally OK to use the default value. But often the well is only perforated in part of the cell thickness, and also the average cell permeability may not be representative for the permeability in the interval that is perforated. In such cases it is better to specify the Kh explicitly.

Skin

The skin factor enters the well in/outflow calculations (see below), and can be specified here (default value is zero).

Dir

The direction the well penetrates the cell. For a vertical well this is the *Z*-direction, for a horizontal well parallel to the *x*-axis *X*. The choices are *X, Y,* or *Z.*

r_0

Pressure equivalent radius. Generally it is recommended to default this value, in which case the Peaceman equivalent radius will be used (see below). To use an alternative value, specify $r_0 > 0$.

Wellbore in/outflow calculations

We look at the simplest possible model, a vertical well in a homogeneous isotropic reservoir producing incompressible fluid at constant rate *q*. The assumptions imply cylinder symmetric flow, so that the pressure variation near the well is described by Darcy's law:

$$q = \frac{K \cdot A}{\mu} \frac{dp}{dr} \tag{17}$$

where r is the distance from the wellbore centre, and A is the area the fluid crosses at distance r, i.e. $A = 2\pi rh$. h is total completion height.

Equation (17) is easily solved,

$$\int_{p_{rw}}^{p} dp = \frac{q\mu}{2\pi Kh} \int_{r_w}^{r} \frac{dr}{r} \Rightarrow p - p_{wf} = \frac{q\mu}{2\pi Kh} \log(\frac{r}{r_w}) \tag{18}$$

Note that the lower bound of the integral is not zero, which it would be for an ideal point source, rather we integrate from a finite well radius. Expression (18) actually requires that the well has a finite non-zero radius. $p_{wf} = p(r_w)$ is the pressure value at the well radius, and we assume that $p(r) = p_{wf}$ when $r < r_w$. p_{wf} is the flowing bottomhole pressure, which we so far have denoted P_{BHP}. We also recognize the permeability thickness in expression (18).

When a well is drilled the drilling pressure (inside the well) is normally kept higher than external reservoir pressure, to avoid reservoir fluids flowing into the well during drilling. Normally drilling fluids will penetrate the reservoir, and the drill procedure itself can damage a zone near the well. Some times, a stimulating fluid is added to the damage zone to improve productivity. The bottom line is that in general there is a zone near the well where flow properties are different than those assumed when deriving Equation (18), such that actually observed pressure drop differs from that predicted by Equation (18). This pressure deviation Δp_S is conveniently defined as,

$$\Delta p_S = \frac{q\mu}{2\pi Kh} S \tag{19}$$

where S is a dimensionless constant called the skin (or skin factor).
Taking account of the skin, we update the pressure drop (Equation (18)) to,

$$p - p_{wf} = \frac{q\mu}{2\pi Kh} \left\{ \log(\frac{r}{r_w}) + S \right\} \tag{20}$$

(see Figure 14).

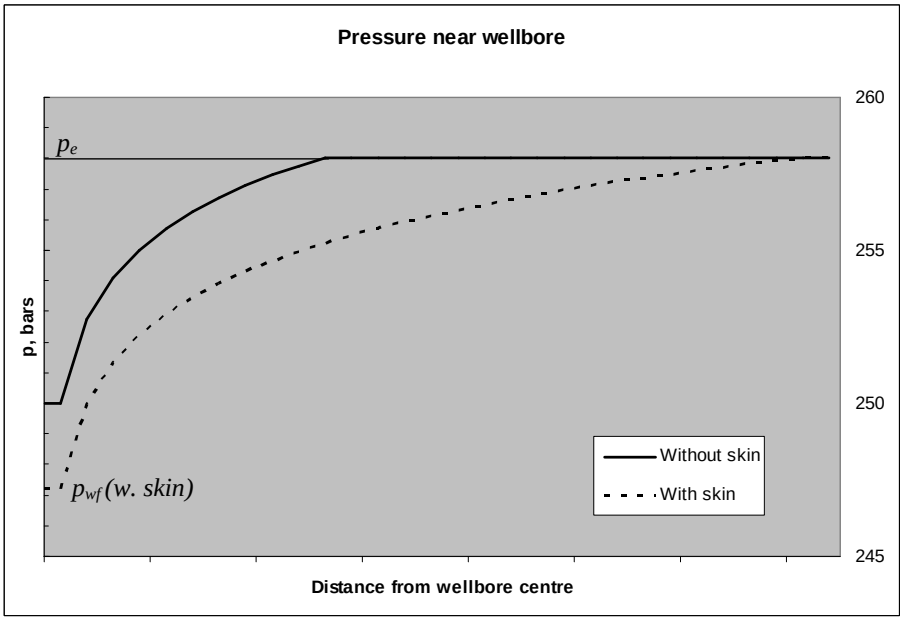


Figure 14. Pressure variation near well bore – without skin and with positive skin

If we let $r \to \infty$ in Equation (20), the pressure will also grow without bounds. This is clearly unphysical, so we define a *drainage radius* r_e, where the influence from the well can be neglected, and pressure has stabilised to a value p_e:

$$p_e - p_{wf} = \frac{q\mu}{2\pi Kh}\left\{\log(\frac{r_e}{r_w}) + S\right\} \tag{21}$$

When p_e and p_{wf} are known, Equation (21) can be solved to express maximum possible rate from the well:

$$q_{\max} = \frac{2\pi Kh}{\mu}\frac{p_e - p_{wf}}{\log(\frac{r_e}{r_w}) + S} \tag{22}$$

If we want to use Equations (18) – (22) in a numerical simulator we only have access to pressure as average cell pressure. And we cannot observe variation like the one described by Equation (20) unless we have a very fine grid near the well. What we need is some mechanism that relates the cell pressures we have computed in the simulator to the expressions above. In papers from 1978 and 1983 Peaceman showed that if the drainage radius r_e is replaced by an equivalent cell radius r_b, Equation (22) can be used to compute max. rate if p_e is replaced with the average cell pressure p_b. Peaceman suggested that for a cartesian grid and isotropic medium, r_b could be computed as,

$$r_b = 0.1h\sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{23}$$

r_b as computed by Equation (23) is called Peaceman's equivalent radius (or Peaceman-radius). Several suggestions for improving this expression have seen light during the last decades.

Example
Well PROD1 is completed in cells (6, 8, 1), (6, 8, 2), (6, 8, 3) and (6, 8, 7). All connections are open for production. Default saturation table will be used, the wellbore diameter is 0.3 m., Eclipse can compute the Kh-product, skin is zero, and the well is vertical.
Well PROD2 is completed in cells (9, 5, 4), (9, 5, 5), and (9, 5, 6). All connections are open for production. Default saturation table will be used, the wellbore diameter is 0.3 m.
Only part of the layer thicknesses are perforated, so we will define the Kh-products manually.

```
COMPDAT
-- Wnm* IC JC KTOP KBTM OPEN? SatTbl N/A Dw   Kh Skin N/A Dir r0
  PROD1  6  8   1    3  OPEN     1*   1* 0.3  1*   0   1*  Z   1*  /
  PROD1  6  8   7    7  OPEN     1*   1* 0.3  1*   0   1*  Z   1*  /
  PROD2  9  5   4    4  OPEN     1*   1* 0.3  400  0   1*  Z   1*  /
  PROD2  9  5   5    5  OPEN     1*   1* 0.3  900  0   1*  Z   1*  /
  PROD2  9  5   6    6  OPEN     1*   1* 0.3  100  0   1*  Z   1*  /
/  Empty record terminates keyword
```

Production / Injection data (Keywords WCONPROD / WCONINJE)

These keywords are used to specify production injection rates, guidelines and constraints. Eclipse will determine production / injection rate such that all constraints are fulfilled. If no constraints are violated the rate will be determined by the primary guide, defined by the Control mode item.
A production well has its data specified in the WCONPROD keyword. Each line (record) in the keyword contains production data for one well, and production specifications for a well are valid until they are redefined. The WCONPROD keyword can occur as many times as needed in the SCHEDULE section. Each record is comprised by,

Wnm Open/ShutFlag CtrlMode Orat Wrat Grat Lrat Resv BHP THP VFP-data ...*

Wnm*

As in COMPDAT

Open/ShutFlag

This is an open/shut flag for the *well*, in contrast to the flag in the COMPDAT keyword which was for individual connections. The choices are,

OPEN: Well is open for production (default)

STOP: Well is plugged (stopped) above the reservoir, may be open for crossflow.

SHUT: Well is isolated from formation

(AUTO: Opened automatically by triggering options in the drilling queue.
Ref. Eclipse documentation.)

CtrlMode

Primary control mode for the well. The specified mode is the guiding mode that the well will be controlled by if no other constraints are violated. The available flags are, (no default)

ORAT: Oil rate target
WRAT: Water rate target
GRAT: Gas rate target
LRAT: Liquid rate target
RESV: Reservoir fluid volume rate target
BHP: Minimum allowed bottom hole pressure
THP: Minimum allowed tubing head pressure
GRUP: Well is under group control

Orat Wrat Grat Lrat Resv BHP THP

These are the actual guides / targets / constraints that will be used to determine producing rate. For all the rate constraints, the default is "no limit". Default BHP is atmospheric pressure, but it is recommended to always define a minimum BHP, not less than the lowest value in the PVT tables.

VFP-data....

Tables for Vertical Flow Performance Calculations are important for doing real field simulations, but will not be covered here.

Constraints / targets / control modes

The rate targets are surface rates or constraints, which all have to be fulfilled, but some can be defaulted, which means "no limit".
RESV is a special target. In principle it refers to a fluid rate measured at reservoir conditions. If we e.g. want injection to balance production such that liquid out = liquid in, we cannot do this by setting injected water equal to produced liquid, since the equality ceases to be valid when the fluids are moved to reservoir conditions. But the flag RESV tries to come around this problem. Unfortunately, the calculation of equivalent reservoir fluid volumes are based on average reservoir pressure, not the pressure of the actually produced fluid. So the RESV fluid volumes are only approximately correct.

When the fluid flows from the reservoir to the surface in the well, the fluid pressure will decrease as the fluid moves upwards. The pressure drop is proportional to the completion depth, and more or less equal to hydrostatic pressure. Obviously, if the fluid leaves the reservoir at a pressure which is lower than the pressure drop it experiences on the way to the surface, it has too low energy to reach the surface at all. Therefore, if fluids are to be produced without additional means of lifting, it is required that the fluid pressure is above some minimum level when they enter the well. This is the minimum required bottom hole pressure BHP. From Equation (21) we see that the BHP will be lower when the rate is increased. A BHP violation can therefore often be met by reducing rate.

Constraints on THP (tubing head pressure) act just as the BHP constraints, but are more realistic. The reason for this is that THP is the pressure measured at the tubing head, i.e. at the surface end of the well. THP is hence an accessible measure, while BHP, measured in the reservoir is only available when such measurements are done, which will be relatively infrequent if at all. To use THP data in the simulation we need to be able to compute THP values from BHP and vice versa. This is done by VFP curves (Vertical Flow Performance), which are curves which do the desired conversions. The VFP relations are complex, involving depths, tubing lengths and fluid saturations, and are beyond the scope of these notes. We will therefore stick to the BHP constraints.

Example

```
WCONPROD
-- Wnm* Open? CtrlMode Orat Wrat Grat Lrat Resv BHP THP ...
  PROD1 OPEN    ORAT   3000 2000  1*  4500  1*  220  1*  /
/
```

Production data has been specified for a single well, PROD1, which is open at the current time. The primary control mode is by oil rate, and the guides will be applied as follows:
First Eclipse checks if it possible to fulfil the primary goal. If oil can be produced at a rate of 3000 Sm^3/day and at the same time all the other constraints are OK:

- BHP > 220 bars
- Water rate < 2000 Sm^3/day
- Liquid rate (water + oil) < 4500 Sm^3/day
- (No constraint on gas rate)

then the oil rate is set to the target rate. If any of the items in the bullet list are violated, then Eclipse will attempt to reduce the oil rate until all the constraints are OK. If it is not possible to fulfil the constraints with a positive oil rate, then Eclipse will write a message and close the well for the current time step. But at later time steps the well will be checked to see if it can be reopened.

Observation: Violation on the secondary targets will generally not be discovered before at the end of the computations for the current time step. Eclipse will therefore have to redo the entire calculations with redefined input, and iterate until a solution satisfying all constraints has been found.

Injection well – WCONINJE

The WCONINJE keyword for controlling injection wells is very similar to WCONPROD. The syntax for each record (line) is,

Wnm InjType Open/ShutFlag CtrlMode Rate Resv BHP THP N/A*

Wnm*

As before

InjType

This is either WATER (water injector) or GAS (gas injector)

Open/ShutFlag

As in WCONPROD

CtrlMode

RATE:	Surface flow rate target
RESV:	Reservoir volume rate target
BHP:	Bottom hole pressure
THP:	Tubing head pressure
GRUP:	Under group control

Rate

Target or upper limit for injected fluid surface flow rate (default no limit)

Resv

As in WCONPROD (default no target)

BHP

Target or maximum bottom hole pressure. Note that for an injector, the argument is turned upside down, such that we now need a high surface pressure to get the fluid into the reservoir. (default 6895 bars (=10^5 psi), which for practical purposes means no limit)

THP

Comments as in WCONPROD.

Example

```
WCONINJE
-- Wnm*  InjTyp  Open?  CtrlMode  Rate  Resv BHP  THP
'INJ*'   WATER   OPEN    RATE     4000    1* 450   1*  /
 INJ4    WATER   OPEN    BHP      4000    1* 480   1*  /
/
```

First all wells with name starting with INJ are defined as water injectors. Constraints:
The wells will inject water at a surface rate of 4000 Sm^3/day as long as the bottom hole pressure stays below 450 bars. If this is not possible, injection rate will be reduced so that the bottom hole pressure constraint is fulfilled.
Next the well INJ4 is defined. Although it also fits in the INJ* wildcard, the last definition will be the valid one. INJ4 will be controlled by bottom hole pressure, i.e. it will inject water with a constant BHP of 480 bars, if this can be done with a water injection rate not exceeding 4000 Sm^3/day. If this is not possible, the BHP will be reduced until injection rate is at most 4000 Sm^3/day.

Economic well constraints (keywords WECON, WECONINJ)

The constraints we discussed for the WCONPROD and WCONINJE keywords are physical constraints. If these are violated, production / injection is not possible. Another kind of constraints are those that are based on economy. A natural requirement for any well would be that the value of produced hydrocarbons from the well should as a minimum pay for the costs of maintaining the well. Such constraints can be specified by the keyword WECON (and also WECONINJ which is rarely used). The syntax of WECON is,

WECON
Wnm MinORAT MinGRAT MaxWCUT MaxGOR MaxWGR Workover EndRunFlag*

Wnm*
: Well name wildcard as before

MinORAT, MinGRAT, MaxWCUT, MaxGOR, MaxWGR
: Economic constraints that all have to be satisfied if the well shall continue to stay open. The constraints are in order, minimum oil production rate, minimum gas production rate, maximum water cut, maximum gas-oil ratio, maximum water-gas ratio. (Default values = "don't use constraint")

Workover
: What to do if any of the economic constraints are violated. If the specified minimum oil rate or gas rate cannot be satisfied, the well will be closed unconditionally. If one of the other constraints is the offending one, the following options are available,

NONE	Do nothing (default)
CON	Shut the worst-offending connection
CON+	Shut the worst-offending connection and all those below it.
WELL	Shut / stop the well
(PLUG	Plug back well, ref. Eclipse documentation)

EndRunFlag
: Request for whether the simulation shall be terminated if the well is shut or stopped. The choices are YES or NO. If YES the simulation will continue till the next report step and then terminate. (Default is NO).

The keyword has several additional items, ref. Eclipse documentation.

Example

For all wells with name starting with OP we require a minimum oil rate of 200 Sm^3/day and a maximum water cut of 0.9. If these requirements are not met the offending well will be closed, but the run not terminated.

```
WECON
-- Wnm*  MinORAT  MinGRAT  MaxWCUT  MaxGOR  MaxWGR  Workover End?
 'OP*'      200       1*       0.9       1*      1*       WELL    NO  /
  /
```

It is only natural that we set the EndRunFlag to NO in the example, since we wouldn't want to terminate the run as long as there are producing wells left. But when the last well has been shut, there is no point in continuing the simulation.

For such cases we have available the keyword GECON – group economic control. The keyword is similar to WECON, with the slight variation we need:

GECON
Gname MinORAT MinGRAT MaxWCUT MaxGOR MaxWGR Workover EndRunFlag*

Gname*
Group name wildcard. For our purpose this will always be FIELD, in which case the constraints apply to the field as whole.

MinORAT MinGRAT MaxWCUT MaxGOR MaxWGR
These are exactly as in WECON, except they are now understood as group (field) quantities, not single well (default values as in WECON).

Workover

NONE	Do nothing (default)
CON	Shut worst-offending connection in worst-offending well
CON+	As CON, but also shut all connections below
WELL	Shut / stop worst-offending well
(PLUG	As WECON)

EndRunFlag

YES	Terminate run at next report step if all producers are stopped or shut
NO	Continue run regardless (default)

Often both WECON and GECON will be used, they are *not* mutually exclusive.

Other often used Well control keywords

WELTARG
When well control data has been defined by WCONPROD or WCONINJE, typically only target rates are changed later. In stead of repeating all the data every time a small change is made to already defined constraints, the keyword WELTARG can be used. It is suited for situations where only one parameter for a previously defined well shall be changed. The syntax is,

WELTARG
Wnm ControlToChange NewValue*

Example
All constraints have been defined for well OP1, but at a certain time we wish to change the target oil rate from its previous value to 3000 Sm^3/day.
This can be done by,

```
WELTARG
 OP1   ORAT  3000  /
/
```

The advantage goes beyond a shorter syntax, as it is clear to the reader that only the one constraint was changed.

WCONHIST / WCONINJH
These keywords are very similar to WCONPROD / WCONINJE, but are used to define actual historical rates and pressures. The advantage is primarily in the post-processing phase, when target data can be compared to simulated results.

WEFAC
Wells are never on production 24 hours a day, 7 days a week. Some down-time is unavoidable. If a well is operative 80% of the time with a producing rate *Q*, simulation production rate could of course be defined as *0.8Q*, ensuring correct grand total. This would however not be entirely correct, as the bottomhole pressure calculations would be based on an erroneous rate. A correct solution can be obtained by use of the WEFAC keyword (well efficiency factor), which is used to specify the factor of producing time to total time.

8.2 Time stepping

As previously noted, the numeric solution is not continuous in time, but advanced in time in discrete steps of length depending on the problem. In general, implicit solution schemes (which Eclipse uses by default) allow for larger time steps than explicit schemes. The term “time step” in simulation is more or less used to describe different kinds of time stepping. Although we will also adopt this sloppy terminology, we will firstly define a more accurate notation.

1. The simulator will advance the solution from one point in time to another. The size of the step is governed by the problem and the solution procedure, but some guidelines are often provided by the user. These solution-dependent time steps are called *ministeps* by Eclipse.
2. The user will specify milestones (times or dates) of special interest. These milestones may be e.g. times when a new well is put on production, when a well rate is changed, or simply to force the simulator to dump output. These are the points in time which are referred to as time steps (TSTEPS) in Eclipse, but are more correctly denoted milestones.
3. As we shall see when we discuss writing of results for visualisation purposes, a “snapshot” of the simulation, i.e. a list of “all” relevant data in all grid cells, can be pretty large. Therefore, such snapshots are only produced on user specified times (normally much larger intervals than the milestones described above). For reasons which will become clear, these “snapshot times” are called *restart times*.

In this section we discuss the milestones (TSTEPS).

Eclipse writes a progress report of the simulation, in a file called <root>.PRT (where the original data file was <root>.DATA). How much Eclipse shall write to this file is determined by the user, and described later, at this stage we only note that Eclipse writes data to the .PRT file at the milestones.

In addition, any change to well data (or other dynamic data) can only happen at the milestones.
An obvious and minimum required use of the milestones is to tell the simulator how far forward in (real) time to simulate.

The simplest syntax is by the TSTEP keyword:

TSTEP
step1 (step2 step3 ...) /

The specified steps are lengths of the time steps (time between milestones, not the milestones)

Example

```
TSTEP
 5 10 15 30 5*50  /
```

tells the simulator to advance the simulation 5 days, then 10 days, then 15, then 30, and finally take 5 time steps of 50 days each. By this scheme the milestones would be after
5, 15, 30, 60, 110, 160, 210, 260, 310, and 360 days.

In practice we often know the milestones by date, not by time intervals, so it is more common to define the milestones by the DATES keyword. The milestones are then defined as
`day month year`, where

- *day* is a number 1,...,31,
- *month* is JAN, FEB, MAR, APR, JUN, JUL (or JLY), AUG, SEP, OCT, NOV, DEC
- *year* is the numerical year (no abbreviation).

Eclipse knows about month lengths and leap years, so DATES is an easier way to define the milestones at the start of a year or month.

Each line in the DATES keyword contains exactly one date, and is terminated by a slash.

Example

```
DATES
  1 JAN 1998  /
 15 JAN 1998  /
  1 MAR 1998  /
  1 JUL 1998  /
  1 JAN 1999  /
/
```

Order of actions

When determining where to put e.g. a well action in the SCHEDULE section to get it to happen at the desired time, it can be helpful to observe that Eclipse reads the section only as far as necessary. I.e. once it encounters a TSTEP or DATES, it will start to advance the solution to the specified next milestone, using the data it regards as current.

Example

Start date has been defined as 1. Jan 2004 in the RUNSPEC section (ref the basic data input example). An order of input could then be,

```
SCHEDULE

WELSPECS
...
COMPDAT
...
WCONPROD
...
-- The defined wells and rates are valid from start date, and will
-- now be used when advancing the solution to 1. FEB 2004
DATES
 1 FEB 2004 /
/

WELTARG
...
-- The current date is 1.FEB 2004. So the change defined in WELTARG
-- will happen at this date

-- The solution will now be advanced first to 1.MAR 2004, then
-- further to 1.APR 2004, using the current rate defined in WELTARG
DATES
  1 MAR 2004  /
  1 APR 2004  /
/
.
.
```

8.3 Convergence Control I (keyword TUNING)

This subject will be covered later, but deserves some mentioning here.
Only exceptionally can the solution be advanced from one milestone to the next (as described in 8.2) in a single ministep. Eclipse has a default handling of its solution procedure, in this context mainly by adjusting the ministep size. The TUNING keyword is used when the user wants to override Eclipse default behaviour.
When advancing the solution we want to take as large ministeps as possible (to minimize computing time). On the other hand there is an upper limit to the possible ministep size, which depends on the solution procedure and the solution itself. The way Eclipse tries to optimize this procedure is by looking at the ministep it has just finished. If the solution was found without problems, Eclipse will take this as a signal that the length of the ministep can be increased. If on the other hand the solution

was only found after convergence problems, Eclipse will take that as a hint that the ministep size should be reduced (chopped).
There are some parameters that governs this behaviour,

TSINIT	Max. length of *first* following ministep (after the keyword) (default 1 day)
TSMAXZ	Max. length of any ministep (default 365 days)
TSMINZ	Min. length any ministep (default 0.1 day)
TSMCHP	Min. length any choppable ministep (default 0.15 day)
TSFMAX	Max. ministep increase factor (default 3)
TSFMIN	Min. ministep cutback factor (default 0.3)
TSFCNV	Cut factor after convergence failure (default 0.1)
TFDIFF	Max. increase factor after convergence failure (default 1.25)

Example of how these parameters are used, assuming the default values.
When the simulation starts, the length of the ministep is 1 day (TSINIT). If the solution is found without problems, the length of the ministep will be increased, by a factor TSFMAX, so the following ministep will be 3 days. As long as the convergence is OK, the ministeps will be increased by a factor 3 (TSFMAX) every time, so the next ministeps would be 9, 27, 81 days. (Evolved time after these ministeps would be 1 day, 4 days, 13 days, 40 days, 121 days.) The solution at 121 days, following the 81 days time step was found after convergence problems, so the ministep size is reduced, hence the following step will be 81 * 0.3 (TSFMIN) = 24.3 days long. If the solution was not found at all, we had a convergence failure, and the next step would be 81 * 0.1 (TSFCNV) = 8.1 days. In this manner the length of the ministep will fluctuate between TSMINZ and TSMAXZ, increasing length when all is well, chopping when problems are encountered.
Often we observe that every time the ministep length is increased beyond some value, the following step has convergence problems, and the step length is chopped. In such cases it is much more optimal to redefine TSMAXZ to a value which allows for OK convergence, and hence the simulation will progress without convergence problems and chopping (which are computer time demanding). In the example above, we observed that 81 days was too much. If we later saw that every time the ministep length became larger than 60 days, the following step was chopped, then it would be wise to set e.g. TSMAXZ = 50 days.
What these numbers should be is very case-dependent, and no rules of thumb can be given. The essential information is in the run log or .PRT file. So this output should always be examined after a run!

The keyword TUNING is used to set convergence flags. It is comprised of three records, the first one handles primarily time step control, the second one contains internal convergence controls, and should as a general rule never be modified. The third record contains controls on the iteration schemes, which will be covered later.

At this stage we only look at the first record, and leave record 2 and 3 at default values.
The syntax of the first record is then,

TUNING
TSINIT TSMAXZ TSMINZ TSMCHP TSFMAX TSFMIN TSFCNV TFDIFF ... /

Note: TSINIT is the first ministep following the TUNING keyword, not the first step in the simulation as such.

Example,
Set max length of any ministep to 45 days, reduce increase factor after OK convergence to 2.0, and enter the other default values explicitly:

```
TUNING
--TSINIT TSMAXZ TSMINZ TSMCHP TSFMAX TSFMIN TSFCNV TFDIFF
   1.0    45.0    0.1   0.15    2.0    0.3   0.1    1.25   /
 /
 /
```

9. Regions

So far we have tacitly assumed that the reservoir can be regarded as a single entity in all aspects. This will seldom or never be true. A single set of relative permeability curves will rarely be valid everywhere, compaction will normally be dependent on soil type etc.
In general, Eclipse allows for subdividing the reservoir into sub-sets, or *regions*. The regions serve two different purposes. One is to subdivide the reservoir into separate regions which are described by different characterizations of various kinds. The most common of these are,

- Regions of validity for relative permeability curves, SATNUM
- Regions for different sets of PVT-data, PVTNUM
- Regions where different equilibration parameters exist, EQLNUM
- Regions for different compaction data, ROCKNUM

The other kind of regions is for reporting purposes only. We may be interested in e.g. how much water flows from one part of the reservoir to another. The user may subdivide the reservoir into as many regions as desired in any possible manner, and request separate reports for each region (see later). The standard way of doing this is by the keyword FIPNUM (Fluid-In-Place regions), but additional user defined regions are possible, ref. Eclipse documentation.

The way the regions are defined and used is very similar for the different kinds of regions, so we take SATNUM as an example.

The SATNUM keyword defines the saturation regions. Each grid cell is assigned one SATNUM number between 1 and N_{SATNUM}, the total number of regions. Then if a certain grid cell belongs to SATNUM region number *M*, this means that the relative permeability data for this grid cell will be obtained from table number *M*. Hence instead of defining a single family of relative permeability curves, we must now specify N_{SATNUM} tables (of each kind).

Example
Assume our grid is 10 x 20 x 6 grid cells. One set of relative permeability curves is to be used in the two upper layers (*K*=1 and *K*=2), another set in layers 3 and 4, and a third set in layers 5 and 6. N_{SATNUM} = 3, and we would specify the SATNUM regions (in the REGIONS section), as

```
SATNUM
 400*1  400*2  400*3 /
```

whereby the two upper layers (10x20x2 = 400 cells) are assigned to SATNUM region 1, the next two layers to SATNUM region 2, and the last two layers to region 3.

Instead of the single relative permeability table we used in chapter 4, we will now need three, as,

```
SWOF
-- Sw    krw     kro     Pcwo
   0.2   0.0     1.0    10.0
   0.3   0.0046  0.712   2.7
   0.45  0.029   0.371   0.78
   0.6   0.074   0.141   0.3
   0.8   0.167   0.004   0.033
   0.84  0.19    0.0     0.0
   0.9   0.25    0.0     0.0
   1.0   1.0     0.0     0.0   /  End of table 1
   0.28  0.0     1.0    10.0
   0.3   0.0046  0.712   2.7
   0.45  0.029   0.371   0.78
   0.6   0.074   0.141   0.3
   0.8   0.167   0.004   0.033
   0.84  0.19    0.0     0.0
   0.9   0.25    0.0     0.0
   1.0   1.0     0.0     0.0   /  End of table 2
   0.35  0.0     0.8    10.0
   0.45  0.029   0.371   0.78
```

```
0.6   0.074   0.141   0.3
0.8   0.167   0.004   0.033
0.84  0.19    0.0     0.0
0.9   0.25    0.0     0.0
1.0   1.0     0.0     0.0   /  End of table 3
```

During the simulation, the first table will be used for cells in layers 1 and 2, the second table for cells in layer 3 and 4, and the third table for cells in layers 5 and 6. Eclipse needs to be notified that we now have three saturation tables, by the flag NTSFUN in the TABDIMS keyword in the RUNSPEC section.

10. Simplified input and modification of Eclipse arrays

We have earlier defined the default Eclipse input format for an array (e.g. PORO) as a list comprised of one value per grid cell, with possible use of the *n** - option.

Eclipse offers an alternative input format, that is more user-friendly, and often more efficient, by the keywords **BOX, EQUALS, ADD, MULTIPLY, COPY, ...**
In general these commands work by first defining a *box* where the associated input applies, and then defining the input. The BOX keyword is used to set such a box explicitly, but the box can also be defined as a part of the input.

NOTE: When a box has been defined, it remains valid until a new box is defined.

When Eclipse reads data it *immediately* performs an action on the data array it is processing. This action can be to overwrite an existing value, or do arithmetic to an existing value. With this knowledge we can build data arrays by repeated assigning and reassigning values, assuring that the end result is the final operations we performed.

Breakdown of the different edit-keywords,

BOX

A *box* is an index-cube, i.e. bounded by constant indices in all directions.
A cell with index *(ix, jy, kz)* is in the box *(ix1 – ix2; jy1 – jy2; kz1 – kz2)* if
$ix1 \le ix \le ix2,\ jy1 \le jy \le jy2,\ kz1 \le kz \le kz2$.

Defining the box above would be by the syntax,

BOX
ix1 ix2 jy1 jy2 kz1 kz2 /

When such a box has been defined, Eclipse takes array input exactly as usual, except only necessary and sufficient data to fill the box are expected. (The standard array input format is hence a special case where the box is the entire reservoir.)

Example
By the keyword

```
BOX
-- ix1 ix2  jy1 jy2  kz1 kz2
    3   6    10  12   5   5   /
```

we specify that until the box is redefined, input data will be for cells (I, J, K), with
$I = 3, 4, 5, 6$; $J = 10, 11, 12$; $K = 5$.
To set porosity $\phi = 0.26$ in this box:

```
PORO
  12*0.26  /
```

We have defined 12 entries, since the box is comprised of 4*3*1 = 12 cells.

The keyword **ENDBOX** (no associated data, no slash) terminates the current box definition, and sets the current box to the entire grid.

EQUALS

This keyword is used to define a single value to a property in a box. The "property" must be identified by its Eclipse array name.
Syntax for each line in the EQUALS keyword,

ArrayName Value BOX,

terminated by a slash. A terminating slash ends the keyword.

Example 1
An alternative way to define the porosity in the example above is,

```
EQUALS
-- Array   Value   ix1 ix2  jy1 jy2  kz1 kz2
   PORO    0.26     3   6    10  12   5  5  /
/
```

Notes

1. The two ways we have done this are *equivalent*, i.e. the current box (3 – 6, 10 – 12, 5 – 5) is valid until redefined also in the last example.
2. Since the box is valid until redefined, the BOX can be omitted in the EQUALS line, and will then default to the last defined box
3. The EQUALS keyword can be used in several places in the data file, but the array which is defined or updated must belong to the correct section (So the current example with PORO goes in the GRID section, an EQUALS for SATNUM would go in the REGIONS section etc.)

Example 2
For a grid with NX = 10, NY = 20, NZ = 5, define three different regions, region 1 for layers 1 and 2, region 2 for J = 1 – 10 in layers 3, 4, 5, and region 3 for J = 11 – 20 in layers 3, 4, 5.
Petrophysics for the three regions:

Region	**1**	**2**	**3**
ϕ	0.21	0.24	0.23
$\mathbf{K_x}$	100	800	650
$\mathbf{K_y}$	100	800	650
$\mathbf{K_z}$	20	60	48

In the GRID section we specify,

```
EQUALS
-- array value ix1 ix2  jy1 jy2  kz1 kz2
  PORO   0.21   1   10   1   20   1   2   /  Region 1
  PERMX  100  /  defaults to last defined box
  PERMY  100  /
  PERMZ   20  /
  PORO   0.24   1   10   1   10   3   5   /  Region 2
  PERMX  800  /  defaults to last defined box
  PERMY  800  /
  PERMZ   60  /
  PORO   0.23   1   10  11   20   3   5   /  Region 3
  PERMX  650  /  defaults to last defined box
  PERMY  650  /
  PERMZ   48  /
   /             Empty record terminates EQUALS keyword
```

and in the REGIONS section:

```
EQUALS
-- array value ix1 ix2  jy1 jy2  kz1 kz2
  FIPNUM   1     1  10    1  20   1   2   /
  FIPNUM   2     1  10    1  10   3   5   /
  FIPNUM   3     1  10   11  20   3   5   /
   /
```

Note that we don't have to specify the number of cells in the box when we include the box definition in the EQUALS keyword.

ADD, MULTIPLY

The ADD and MULTIPLY keywords are very similar to EQUALS, but instead of specifying a value, we modify an existing value, that must have been defined earlier.

Example
In a grid with *NX* = *NY* = 9, *NZ* = 3, there is a region with bad properties in the middle. Region 2 is defined by *IX* = 4 – 6, *JY* = 4 – 6, *KZ* = 2. The rest is region 1.
In region 1, porosity is 0.3, horizontal permeability 2000 and vertical permeability 250. In region 2, porosity is 0.2, and permeabilities are ¼ of the values in region 1.

In this example we will first define data for the whole grid, then redefine in the central area. Note that there are many different ways to do this, and no "correct" or "best" solution. First defining one value for a large box (here the entire grid) and then redefining a sub-box is often efficient.

In the GRID section:

```
PORO
 243*0.3  /  (9*9*3 = 243)

PERMX
 243*2000 /

PERMY
 243*2000 /

PERMZ
 243*250  /

ADD
-- array value ix1 ix2  jy1 jy2  kz1 kz2
   PORO  -0.1   4   6    4   6    2   2  /
 /

MULTIPLY
-- array value ix1 ix2  jy1 jy2  kz1 kz2
   PERMX 0.25  4   6    4   6    2   2  /  repeat box in new keyword
   PERMY 0.25  /        defaults to box above
   PERMZ 0.25  /
 /
```

In the REGIONS section

```
EQUALS
-- array value ix1 ix2  jy1 jy2  kz1 kz2
  FIPNUM  1     1   9    1   9    1   3  /
  FIPNUM  2     4   6    4   6    2   2  /
  /
```

COPY

This keyword is used to copy an entire array (or a box) to another array. Thereafter the modification keywords may be used if needed.
Syntax for each line of keyword:

SourceArray TargetArray (Box)

Each line is terminated by a slash, and an empty record (slash alone) terminates the keyword.

Example
We repeat the example above, this time using the COPY keyword:
First we define PORO and PERMX for the whole grid.
Then we modify the values for region 2

In the GRID section:

```
-- Real or dummy data for whole grid
PORO
 243*0.3  /  (9*9*3 = 243)

PERMX
 243*2000 /

-- Modify to correct values in Region 2
ADD
-- array value ix1 ix2  jy1 jy2  kz1 kz2
   PORO  -0.1   4   6    4   6    2   2  /
 /

MULTIPLY
-- array value ix1 ix2  jy1 jy2  kz1 kz2
   PERMX 0.25  /  defaults to box defined above
 /

-- PERMY and PERMZ are copies of PERMY in entire grid
COPY
-- From   To   BOX
 PERMX  PERMY  1 9  1  9  1  3  /
 PERMX  PERMZ  /
 /

-- Lastly, PERMZ is modified to be 12.5% of hor. permeability
MULTIPLY
 PERMZ  0.125 /
/
```

COPYBOX

While COPY is used to copy one array to another in the same cells, COPYBOX is used to make an exact copy of the same parameter in a box, in another, non-overlapping box. Obviously the two boxes must be of the same size in all three directions.

Example

```
COPYBOX
-- Parameter  -----   From Box   -----    ------   To Box   ------
--            if1 if2  jf1 jf2  kf1 kf2   it1 it2  jt1 jt2  kt1 kt2
 PORO           1  10    1  10    1   2     1  10    1  10    4   5  /
 /
```

11. Eclipse output, formats and files

A complete specification of the reservoir state at any time, i.e. a list of all available data values in all grid cells is huge even for moderate-sized grids. Outputting "all" data at all time steps (ministeps) is therefore simply not an option.

> **NOTE:** Eclipse does not write any results "by default". *Only* data explicitly requested by the user are written to result files. If a needed data vector was not requested in result specifications, the simulation has to be re-run to obtain the desired data.

Basically, Eclipse output can be divided into three different types,

1. Textual output. Contains run summaries, feedback messages, warnings, and errors. Optionally text listing of data arrays or aggregate values.
2. Output aimed at visualisation of time-dependent behaviour, i.e. single-valued functions of time.
3. Output aimed at keeping track of the reservoir state (data array values in all cells) at user defined times.

There are several options for how the files are written. At this time it is of special interest to note the difference between *unified* and *non-unified* files. Non-unified, which is the default, means that Eclipse writes a new file at every time step where output is requested. With unified files Eclipse stores all the relevant results in a single file. There is no difference in contents, total storage space or input/output efficiency by the two formats. Unified has the advantage of fewer files in a folder, while some external software may require non-unified files. In addition, the Eclipse suite contains the program CONVERT, which can change non-unified files to unified and vice versa.
Specification of whether unified or non-unified files are requested / used is done in the RUNSPEC section with keywords UNIFOUT (request unified output) and UNIFIN (use unified input files).
If not specified non-unified files are used.

File names

The data file, which is the one we refer to when we say that "a data set is run by Eclipse" always has a name that ends with ".DATA" or ".DAT". This ending, or extension is called the file name *tail*. The part of the name before the separating period is called the file name root. So if we run a case called OSEBERG1.DATA, the root is "OSEBERG1".
For historical reasons and compatibility it is recommended to use not more than 8 characters in the file name root.
In the following discussion, we will denote the file name root that was used in the dataset name by <root>. I.e., the data set we use as reference is <root>.DATA. (Obviously this cannot be the actual name of the data file.)

Textual output

Eclipse will always write a file called **<root>.PRT** ("report file"), which contains simulation reports. The first part of this file is a listing of the data input and interpretation phase. In addition to simple progress reports, any irregularities in input will be reported here, at three levels of seriousness,

MESSAGE
: normally just information from Eclipse, as e.g. "the grid contains 14678 active cells".

WARNING
: Something can be wrong, but not necessarily, so the run won't terminate. The user is encouraged to check the origin of warnings. A typical warning is that pressure-dependent tables will be extrapolated if pressure goes above or below so-or-so. If the user is confident that such pressure levels will not occur, the warning is not a problem.

ERROR
: Error in input data that prevents the run to start or continue. If possible, a dataset with errors will be read and initialised as far as possible, but the simulation proper will not start. If an error is found in the dynamic part (SCHEDULE), the run will terminate immediately. (Note that such errors are not found before they are encountered, which may be far into the run. Can

be very annoying, but can be avoided by use of NOSIM in the RUNSPEC section.)
If the simulation will not run, or stops unexpectedly, always suspect an error, and consult the report file, where hints to what went wrong normally can be found.

By default all input is written back to the report file. This is normally not desired, e.g. by default Eclipse will write all geometry data and petrophysics back, also if the data was read from an included file (which is more often than not rather large).
Fortunately, Eclipse has a keyword, **NOECHO** (no slash) which turns off writing of data to report file, and **ECHO**, which turns it back on. So routinely, NOECHO is inserted at the start of the GRID section, and ECHO at the end, to prevent writing of the voluminous GRID data (see "Basic data input example").

When the model has been initialised Eclipse writes user-specified initial data to the report file (see "RPT*XXX* keywords" below).

In the dynamic phase, a short convergence summary is written at each ministep. This includes current time, length of current time step (ministep), why this length was chosen, how convergence performed, average reservoir pressure and water cut. For details ref. Eclipse documentation.

NOTE: Eclipse batch run and LOG-files.
When starting Eclipse from the GeoQuest Launcher panel in a Windows-environment, a report log (short summary) including the messages, warning, errors, and convergence reports will be written to a console window. Sometimes we would prefer to have this output to a file instead. This is possible in the MS Windows environment (in linux use standard file redirection) if we create a DOS batch file, e.g. called `ECL.BAT` (tail *must* be BAT). In the simplest case, the file contains only one line which executes eclipse: (assuming dataset name `GF_CASE1`),

```
CALL $eclipse -file GF_CASE1 > GF_CASE1.LOG
```

This command starts eclipse with data file `GF_CASE1.DATA` and writes the log to the file `GF_CASE1.LOG` instead of to the command window. To execute the command, run the file `ECL.BAT` as any other windows program (note: `ECL.BAT` must be in the same directory as the data file, in this case `GF_CASE1.DATA`).
The concept can easily be extended to enabling automatic starting of several Eclipse runs in a row, by inserting more lines of the same kind (different data sets of course) in the .BAT file, or running other programs in between the Eclipse runs, to check or modify results etc. (The syntax of the execution line, starting with "CALL" ensures that a command line will not be executed before the previous command has finished.)

The RPT*XXX* keywords

In addition to the data referred to above, which is always written to the report file, the user may specify which arrays to write to the file, and at which time steps if relevant. Most users find that results are best processed graphically, hence keeping textual output to a minimum. But some times the numeric values are needed, so it is nice to know how they can be made available. The appropriate keywords are,

RPTRUNSP
RPTGRID
RPTPROPS
RPTREGS
RPTSOL
RPTSMRY
RPTSCHED

(In addition there is RPTRST which will be covered later.)

RPTRUNSP, RPTREGS, and RPTSMRY are rarely used, ref. Eclipse documentation if needed.
Each of the other keywords is associated with a list of flags that turn writing of the specified feature on or off. The number of flags is very long, so here we concentrate on the most common flags – for complete specification ref. Eclipse documentation.

RPTGRID

All of the standard grid generation and petrophysics keywords may be specified.

Example

To output X-direction grid block sizes, grid block top depths, porosities and all permeabilities,

```
RPTGRID
 DX TOPS PORO PERMX PERMY PERMZ /
```

RPTPROPS

Rel. perm tables: SWOF, SGOF.
PVT-tables: PVDG, PVTG, PVDO, PVTO, PVTW.
Rock compaction: ROCK, ROCKTAB

Example

To output oil-water relative permeability, live oil and dry gas PVT tables, and compaction tables,

```
RPTPROPS
  SWOF  PVTO  PVDG  ROCKTAB  /
```

RPTSOL

Output of equilibration data: EQUIL
Fluid in place reports are created by use of the flag FIP, which may take four values:
FIP=0 No FIP-reports
FIP=1 Initial fluid in place volumes are reported for the whole field
FIP=2 As FIP=1, and in addition initial FIP-volumes are reported for all FIPNUM regions
FIP=3 As FIP=2, in addition initial FIP-volumes are reported for all FIP regions (not covered here)

If an initial restart file is requested it must be specified in RPTSOL (see RESTART section). The appropriate flag is RESTART.
For normal use, set RESTART=2 (Other values, ref. Eclipse documentation).

Example

In most runs, RPTSOL will be used to request an initial restart file and initial fluid in place reports for all FIPNUM regions, as,

```
RPTSOL
 RESTART=2  FIP=2  /
```

RPTSCHED

Output of results from the simulation. Can also be used to control writing of restart files, which will be covered later. As noted above, textual output of dynamic arrays (like pressure, saturation) is only exceptionally needed or desired, as they are normally better analysed by a visualisation program. About any array can be output to the report file, ref. Eclipse documentation. A couple of other useful flags are,

NOTHING
: clears the report flags (since RPTSCHED can occur as often as needed in the SCHEDULE section, it is convenient to have a flag that clears previous flags).

FIP
: Fluid in place reports, equal to the flag in RPTSOL, but now reported at every milestone timestep.

RESTART
: Controls generation of restart files, as an alternative to the RPTRST keyword, which will be covered later.

WELLS=*n*
: Output of well reports (injection / production rates and totals)
n can take one of the following values:

1. report well flows
2. report well and connection flows
3. report layer totals
4. report layer totals and well flows
5. report layer totals, well and connection flow

Example
To request Fluid-in-place reports for all FIPNUM-regions, and well reports including well totals, and from each connection, set

```
RPTSCHED
  FIP=2  WELLS=2  /
```

Time dependent vectors – SUMMARY data

Most if not all single-valued results can be stored for later visualisation in programs like GRAF or Eclipse Office. The most common is to show evolution of some variable vs. time, but just about any parameters can be used as *X*- or *Y*-axes, e.g. total water production vs. total oil production. Such data are called SUMMARY data, and Eclipse will store one value of each data item at each ministep in *summary* files (see below). How to visualise results in a post-processing program will not be covered here. The important fact to notice is that Eclipse stores only those variables the user explicitly requests. The data to store are listed in the summary section, a complete list of permitted variables can be found in the Eclipse reference manual. We concentrate on the general syntax, and some of the most common data.

SUMMARY data syntax

Most of the variable names are defined by four-letter abbreviations with strict rules. These names are so common that all Eclipse users should know at least the most used of these rules.
Some exceptions occur, but as a general rule,

First character:
F Field
G Group
W Well
C Connection
R Region
B Block (Grid cell)

Second character:
O Oil
W Water
G Gas
L Liquid
P Pressure

If second character denotes a phase,
Third character:
P Production
I Injection

Fourth character:
R Rate
T Total

By these rules we can build many of the most common summary keywords, some examples,
FOPR Field Oil Production Rate
WWIT Well Water Injection Total
FGPT Field Gas Production Total
GLPR Group Liquid Production Rate

Some common keywords don't follow the rules exactly:

Keyword *X*PR	(*X* = F, R, or B)	Pressure (average), (Field, Region or Block)
Keyword *X*WCT	(*X* = F, G, W, C)	Water Cut
Keyword *X*GOR	(*X* = F, G, W, C)	Gas Oil Ratio
Keyword *X*WGR	(*X* = F, G, W, C)	Water Gas Ratio
Keyword *X*GLR	(*X* = F, G, W, C)	Gas Liquid Ratio
Keyword WBHP		Well Bottom Hole Pressure

For **regions**, the relevant data is not production / injection, but remaining fluids, ratios, and inter-region flow. Since the field is one aggregate region, most of these keywords are valid for field or region: (*X* can take the value F or R)

*X*OIP	Oil in Place
*X*WIP	Water in Place
*X*GIP	Gas in Place

Inter-region flow is defined by,

First character:	Always R
Second character:	O (oil), W (water), G (gas)
Third character:	Always F (flow)
Fourth character:	R (rate) or T (Total)

(Not all combinations are supported; ref. ECLIPSE Techn. App.)

Then there are some useful keywords which are derived from those above. OIP = "oil in place", and subscript 0 means "at time zero", while no subscript means "now" (*X* is F or R).

*X*OE Oil Efficiency = $\frac{OIP_0 - OIP}{OIP_0}$

*X*OEW Oil Efficiency from wells = $\frac{\text{Oil Prod. from wells}}{OIP_0}$

Lastly we mention a group of summary keywords for convergence and CPU (computer processor) use. For completeness we list all (of the common ones), although some of the terms have yet not been defined (time step means ministep).

ELAPSED	Elapsed time in seconds
MLINEARS	Number of linear iterations for each time step
MSUMLINS	Total number of linear iterations since start of run
NEWTON	Number of Newton iterations for each time step
MSUMNEWT	Total number of Newton iterations since start of run
NLINEARS	Average number of linear iterations per Newton iteration, for each time step
TCPU	Current CPU usage in seconds
TCPUTS	CPU time per time step in seconds
TCPUDAY	CPU time per day
TIMESTEP	Time step lengths

Data associated with the summary keywords

Some of the summary data keywords require additional data. Which and what is really logical from the context, and some can be defaulted. Also note that Eclipse always requires a terminating slash when keyword data is defined.

Field keywords have no associated data, and hence no terminating slash. Statement valid also for all of the convergence report keywords.

Well keywords need the names of the wells, as e.g.

```
WOPR
 A-1H  A-OP1  B-OP2  /
```

Note that wildcards are not permitted here.

An empty well list means, “store data for all wells”, so

```
WBHP
  /
```

will store bottomhole pressure for all wells in the run.

Connection keywords require both a well name and a connection. The connection must be a grid block that has actually been defined as a connection in the SCHEDULE section. Since each record is composite (both well name and connection), each must be terminated with a slash, and an empty record terminates the keyword.

Example (Connection Oil Flow Rate)**:**

```
COFR
 A-1H  12 15 4 /
 A-1H  12 15 5 /
 /
```

Well names cannot been defaulted, but if the connection indices are defaulted for a well, results will be stored for all the connections in that well.

Group keywords are not of concern in these notes, as we don’t handle groups, but the rules are the same as for wells.

For **Region** keywords, if the keyword data is associated with **one region** only, the region numbers must be listed, as e.g. region oil in place:

```
ROIP
 1  2  3  4  6  /
```

If an empty list is given, data is stored for all regions.

If the keyword data is associated with data from **two regions**, typically flow from one region to another, then a list of region-pairs must be defined, each pair terminated by a slash, e.g. Region water flux total:

```
RWFT
  1  2  /
  1  3  /
  2  3  /
  /
```

Block keywords include a list of slash-terminated block indices. An empty record terminates the keyword, e.g. Block Pressure,

```
BPR
 3  3  6  /
 5  8  1  /
 1  1 10  /
 /
```

Summary files

Summary data are organized in summary files (which are binary, not text files). The file `<root>.SMSPEC` contains information on how the data is organized, while the actual results reside in a file `<root>.UNSMRY` if unified. If non-unified files are used, the files are called `<root>.S0001, <root>.S0002, <root>.S0003,...`
The file `<root>.S0001` contains results from all mini steps up to the first milestone, `<root>.S0002` results from the first to the second milestone etc.

The EXCEL keyword

If we want access to the summary output data to e.g. manipulate them afterwards, the summary files are not appropriate (not text files). The keyword `EXCEL` (no associated data) is then available, whereby Eclipse writes a file in format readable by a spreadsheet program as Excel.

Restart data and restart files

A restart file is a "snapshot" of the reservoir state at a certain time. I.e. saturation and pressure values for every grid cell are saved, in addition to other data characterising the reservoir and fluids at the time. The reason for the name "restart data", and the original motivation for saving the reservoir state in such a manner, is to enable a "restart" of the simulation. In an ordinary run, when the simulation has arrived at some date, to advance the solution a time step to next milestone, it takes the "current reservoir state" and solves some problem to get the required solution. Whether the "current reservoir state" is stored in computer memory as in a normal run, or read from a file shouldn't matter at all, as long as the "current reservoir state" is exactly the same in the two situations. The definition of the restart file is just to ensure that the contents of the file matches this requirement.

The reason for wanting to do such restarts is most easily given by an example. Assume we have run a simulation case for 20 years with a defined well scenario. We now want to compare the case to another case where we put an additional injection well on injection after 15 years. Obviously the first 15 (simulated) years are equal in the two cases, so we would have to perform an identical simulation for 15 years before we do the change. If total computing time for the run is in the order of minutes this wouldn't bother us at all, but if for example simulation of the first 15 years took a couple of days of computing, then we would obviously see the rerun as a waste of time and seek alternative ways to do this. If we have stored the reservoir state after 15 years in a restart file, then we could pick up the state from there, add another well and run only the last five years anew. (Details on how to do this will come later.)

Since restart files contain a complete description of the reservoir state, they are also well suited for graphic visualisation of the state, e.g. the saturation distribution in the reservoir at the time. And by visualising several such restart data in a sequence, we can get a picture of how the reservoir state changes with time.

The dates at which restart data are stored in the restart files are called *restart dates*. Since restart files tend to be large they are normally generated less frequently than the milestones. Also, whether the objective is to use the restart files for proper restarts, or visualisation, they don't need to be generated too often, as they serve their purpose best when the difference between consecutive files is not too small. Recommended intervals between restart dates are case-dependent, and would be based on how rapid the reservoir state changes, and the size of each file. Note that it is possible to specify that the restart files are only needed for visualisation purposes (not actual restarts), which results in somewhat smaller files.

Control of restart files can be done in RPTSCHED, but the alternative RPTRST is tailored for this purpose, so in these notes we will use RPTRST.

RPTRST is, as the other RPTXXX keywords controlled by a series of flags,

BASIC=*m*

Requests output of basic (standard) restart files ("report time" = milestone)

m = 1 Restart files are created at every report time, but only the last one is kept. (For unified files this switch is not available, and *m* = 2 will be used)

m = 2 Restart files are created on every report time, and all are kept.
m = 3 Restart files are created every n^{th} report time. n is defined in flag FREQ.
m = 4 Restart files are created at the first report step of every year, or every n^{th} year if n has been set in flag FREQ.
m = 5 Restart files are created at the first report step of every month, or every n^{th} month if n has been set in flag FREQ.
m = 6 Restart file is created every time step (ministep)

FREQ=*n*

Controls the frequency of creation of restart files when BASIC = 3, 4, 5.

NORST=*m*

Requests a graphics-only restart file (smaller, but rarely an issue with today's memory)
m = 1 Standard graphics restart file
m = 2 Well arrays not included

ROCKC

Output modified (i.e. actual values used at the time step) values of cell pore volumes and transmissibility multipliers.

In a standard black oil simulation, pressure, gas and oil saturations, and gas resolution is written to the restart file by default. These parameters are therefore not necessary to request.

Restart files will only be created at the requested dates if a time step is available. If for example milestones are defined every second year, restart files will not be created more often than every second year irrespective of what was requested in the RPTRST keyword. Also restart files will always be created on a milestone time step. E.g., if an excerpt of the DATES keyword was,

```
DATES
 1  AUG 1994  /
 1  SEP 1994  /
 1  NOV 1994  /
 1  MAR 1995  /
 ...
```

and restart files are requested for the start of every year (BASIC=4), then the restart file will not be written 1. Jan 1995, which is not a report time, but the first report time after that, namely 1. Mar. 1995.

Example

To request that graphics-only restart files are written every fourth month,

```
RPTRST
 NORST=1  BASIC=5  FREQ=4  /
```

Note that RPTRST can be used as often as we wish in the SCHEDULE section, so we don't need to keep the same restart creating frequency in the whole run, but can change it when needed.

Initial file and initial restart file

The initial file, or INIT-file contains all cell-value static data, like porosity, permeability,... It is useful to have this information available for graphic visualisation, so it is recommended to always request writing of an initial file. This is done in the GRID section, with the keyword INIT. (See example in BASIC data input example.) The resulting file is named `<root>.INIT`.

Also, the RPTRST keyword will *not* write a restart file at time zero (initial restart file). Such a restart file contains information about all the dynamic data before any flow starts, and is normally useful. It must be requested in the RPTSOL keyword by flag RESTART=2 (in SOLUTION section).

Restart files

If unified output has been requested, the restart data will be written to the file `<root>.UNRST`, and a file `<root>.RSSPEC` will contain information about the files. If the files are non-unified, they follow a similar naming convention to the summary files, with names,
`<root>.X0000, <root>.X0001, <root>.X0002,...`

Note that the numbering of the X-files is synchronized with the summary files. I.e. the X-files are not numbered sequentially, but given a number that coincides with the corresponding S-file. Hence, a sequence of X-files can be, `<root>.X0000, <root>.X0010, <root>.X0015, <root>.X0023,...`
`<root>.X0000` is always the initial restart file (if requested). `<root>.X0010` is the first restart file in the sequence (not counting the initial), and is created at milestone 10. The files `<root>.S0010` and `<root>.X0010` are written at the same date. (Note that this was different in early versions of Eclipse.)

12. Restarting a simulation

As described above, a restart is what we do when we pick up the simulated state at some date (where we have a restart file available), and start a new simulation from that date. Development of the state up to the point where we do the restart is then assumed to be described by the run we restart from.
In Eclipse, restarts can be done as “standard” or from SAVE files. The “standard” way is more flexible, but the SAVE way is normally faster. We will not discuss the SAVE way here.

In a “standard” restart, the simulation is defined in the usual manner, but initialised by the restart data. I.e., the restart run needs all the standard initialisation data except the equilibration which is no longer valid. The restarted run must be given a case name different from the original run. Then the procedure to create a restart run is,

Assume the original run is called `CASE1.DATA`. We will restart this run from time step (milestone) 20, where the state has been saved in a file `CASE1.X0020`.
We will name the restarted run `CASE1_R1.DATA`. First create `CASE1_R1.DATA` as a copy of `CASE1.DATA`. Then edit the contents of `CASE1_R1.DATA`:

- In section SOLUTION delete equilibration keywords (and enumeration and analytic aquifers if used).
- Insert a keyword `RESTART` (in SOLUTION section) to specify which run to restart from, and at which time step. In our example,
  ```
  RESTART
     CASE1  20   /
  ```
- In the SCHEDULE section delete all well and time-stepping keywords up to and including the restart time. (But see below, the SKIPREST keyword)
- Start run (from Eclipse panel you would choose `CASE1_R1` as the case name)

The Eclipse post-processing software knows all about restarts, so if the last restart is given as file name, it will automatically nest itself backwards to the correct beginning.

The SKIPREST keyword

The modifications to be done in the SCHEDULE section are error-prone, and many users have had bad experiences with omitting too much or too little data. Eclipse has therefore provided a solution which is highly recommended:
If a keyword `SKIPREST` is inserted at the beginning of the SCHEDULE section, Eclipse will skip all data that is not needed for the defined restart, and enter the section at precisely the correct position.

13. Fault modelling – Non-neighbour connections

It is clear from Figure 7 (or equivalent Figure 16 below) that one characteristic of faults is that, which grid cells are neighbours is different when a fault is present than not. The fault plane (in reality a volume) is typically an area of crushed rock, with poorer properties than the non-fault volume. To describe flow across a fault it is therefore necessary to define the possible flow paths, and the effective (reduced) fault permeability.

The 7-point stencil

The numerical approximation to the differential $\frac{\partial f}{\partial x}$ in cell (I, J, K) was, using central differences,

$$\frac{\partial f}{\partial x}(I,J,K) \approx \frac{f(I+1,J,K) - f(I-1,J,K)}{C(I+1,J,K) - C(I-1,J,K)} \tag{24}$$

where *C(I, J, K)* means the cell centre coordinate for cell *(I, J, K)*, and *f(I, J, K)* means "*f* evaluated at the centre of cell *(I, J, K)*".

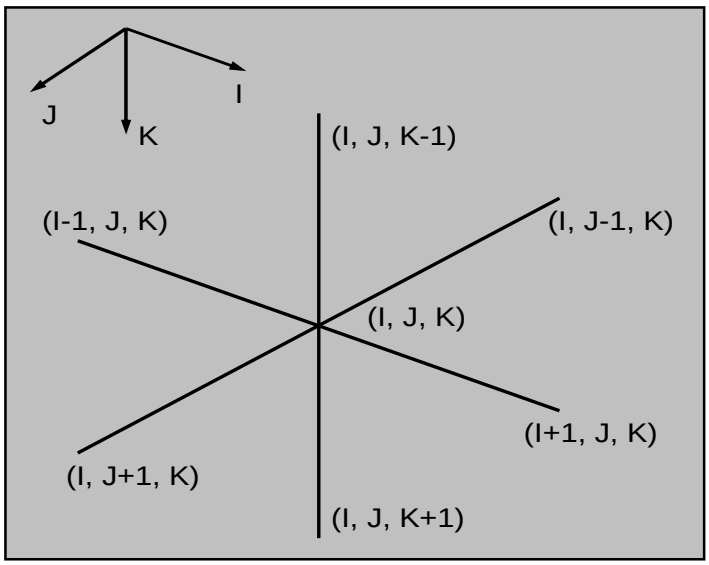


Figure 15. The 7-point stencil schematic

We see that $\frac{\partial f}{\partial x}(I,J,K)$ involves cells *(I-1, J, K)* and *(I+1, J, K)*. In the same manner, $\frac{\partial f}{\partial y}(I,J,K)$ involves *(I, J-1, K)* and *(I, J+1, K)*, and $\frac{\partial f}{\partial z}(I,J,K)$ involves *(I, J, K-1)* and *(I, J. K+1)*.

In total, evaluating *f* and its differentials in all three directions in the cell *(I, J, K)* involves the cell itself and all the direct neighbours, *(I±1, J, K), (I, J±1, K), (I, J, K±1)*, but not any *diagonal* neighbours. This computational scheme is called the 7-point stencil, and is shown schematically in Figure 15. The 7-point stencil is the basic computational molecule for Eclipse. Other computational schemes exist, with greater accuracy obtained by involving more points. In these notes we study only the 7-point stencil.

For a cell *(I, J, K)*, the six other cells in the computational molecules are called its (logical) neighbour cells.

The fault layout – non-neighbour connections

In Chapter 4 we saw that Eclipse converts input permeabilities to inter-block transmissibilities, which are the connectivity parameters that are actually used during computations. The inter-block transmissibilities are computed for all cell-pairs that are neighbours, as in the 7-point stencil. When the grid contains a fault, these concepts will however need some modification.

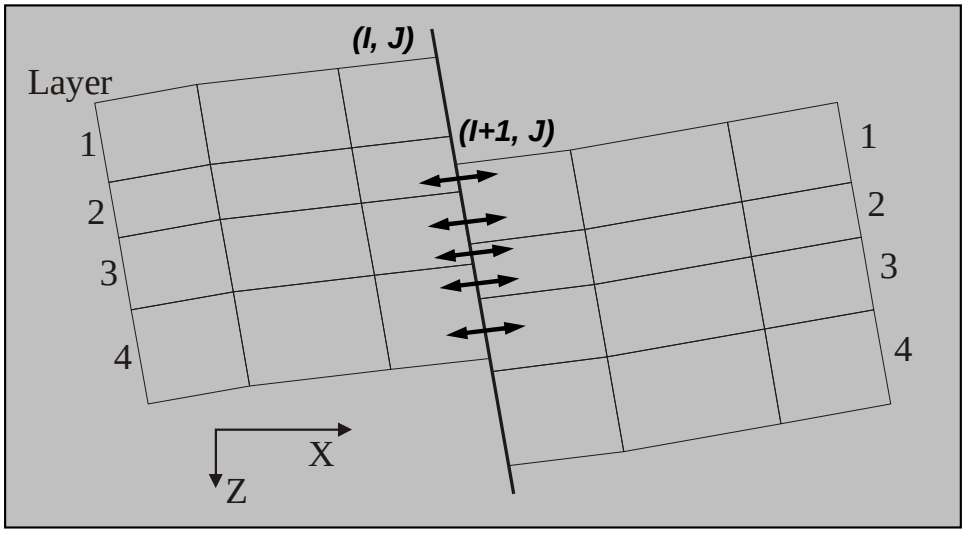


Figure 16. Sand to sand communication across a fault (vertical cross section)

In Figure 16 cell columns *(I, J)* and *(I+1, J)* are separated by a fault. A consequence of the fault is that cells *(I, J, 1)* and *(I+1, J, 1)* are no longer in contact with each other, and hence the transmissibility between these cells should be zero. On the other hand there is a direct flow path between cells *(I, J, 3)* and *(I+1, J, 1)*, so the transmissibility between these two cells should be nonzero.

The arrows in Figure 16 show all possible flow paths between cells that are physical neighbours across the fault. Cells that are thus connected are said to be in *sand-to-sand contact* across the fault. The sand-to-sand contacts in the Figure are,

	Upthrown side	Downthrown side
Layer	2	1
	3	1
	3	2
	4	2
	4	3

During the initialisation phase, Eclipse will discover all sand-to-sand contacts, and modify the necessary transmissibilities accordingly. So in this case the affected transmissibilities that enter the 7-point stencil ((*I, J, K)* to *(I+1, J, K)*, for *K*=1,2,3,4) will be set to zero, and nonzero transmissibilities will be computed for all the cell pairs in the table.

Note that transmissibilities can be computed between any cell pair, not only cells which are logical neighbours. The most common case is when cells are in sand-to-sand contact, but there is no restriction at all regarding for which cell pairs nonzero transmissibilities can be defined.

Eclipse uses the term *non-neighbour connection* for such a case: The cells are *not* logical neighbours (i.e. their cell indices are not neighbours), but the transmissibility between the cells is non-zero (i.e. there is flow connectivity between the cells).

In the Figure 16 – example, cells *(I, J, 2)* and *(I+1, J, 1)* would be a non-neighbour connection, as would the cells *(I, J, 4)* and *(I+1, J, 2)*.

We use the shorthand notation *NNC-cells* for cells which are connected by a non-neighbour connection.

> **Note:** When defining NNC-cells across a fault, we are not restricted to cells which are in sand-to-sand contact, any cell pair can be an NNC. In reality fault flow is much more complex than just sand-to-sand flow, so it will often be necessary to define other NNCs than the ones Eclipse automatically constructs.

We could e.g. define flow from cell *(I, J, 1)* to *(I+1, J, 1)* in the Figure, even though these cells are not physical neighbours. Note also that this is an exceptional case, as these two cells are not non-neighbours, as they are neighbours in the sense of indexing. Flow between the cells *(I, J, 2)* and *(I+1, J, 3)* would however, be defined by an NNC. These are examples of more advanced modelling, and are outside the scope of these notes.

Fault transmissibility multipliers

It is often needed to modify (mainly reduce) the flow conductivity across the fault. As we have seen above, the fault flow is more complex than standard inter-cell flow, so modifying cell permeabilities is not flexible enough to control fault flow – we need to work with the transmissibilities directly. Eclipse offers several ways to do this, but we will only look at the method that is best suited for modifying transmissibilities between sand-to-sand contact cells. Eclipse computes initial values for all sand-to-sand transmissibilities honouring cell permeabilities and geometry (primarily the area of the shared surface). While doing this Eclipse also builds a table of which cells are in contact with which.

Again referring to Figure 16, where the flow across the fault is in the *X*-direction, the fault transmissibilities can be modified by the keyword **MULTX**. The keyword is most often used in the GRID section, but can also be used in EDIT or SCHEDULE sections (ref. Eclipse documentation). The easiest way to use MULTX is in combination with the EQUALS keyword. MULTX can be specified as many times as needed, and the syntax of each line is,

MULTX *multiplier BOX* /

The BOX is a cell box where the multipliers apply. When we specify a BOX for the MULTX keyword, the affected cell pairs are constructed by, for each cell in the box the cell's *X*-face (i.e. the right hand face) is examined for sand-to-sand contact cells, and transmissibilities for all such contacts are modified. This is easier to envisage by an example. Still referring to Figure 16, assume now that the column denoted *(I, J)* is actually *(6, 8)*. I.e. there is a fault between *I*=6 and *I*=7 for *J*=8. In practice faults usually are several cells long, so assume that the shown fault picture is valid for *J=4,5,...,10*. To modify transmissibilities associated with layer 3 on the upthrown side, we would use the keyword,

```
EQUALS
  MULTX  0.025  6  6  4 10  3  3  /
  /
```

Now, the transmissibility between cells *(6, J, 3)* and *(7, J, 1)*, and between cells *(6, J, 3)* and *(7, J, 2)* will be multiplied by 0.025, for all *J = 4 – 10*.
This is the really nice feature with MULTX compared to other, more manual methods to modify fault transmissibilities. MULTX acts on *all* sand-to-sand contact cell pairs emerging from the cells defined in the box, and we as users do not even need to know which cells are going to be affected – everything is taken care of automatically.

If we wanted to modify the whole fault in the same fashion, we would use,

```
EQUALS
  MULTX  0.025  6  6  4 10  1  4  /
  /
```

Note that we have included layer 1, even though layer 1 has no sand-to-sand contact across the fault (layer 1 on upthrown side – MULTX *always* acts in the direction of increasing *I*-index!), as Eclipse has no problems figuring out this itself, and arrive at the correct solution.

A commonly occurring situation is that the flow conductivity is dependent on the layers on *both* side of the fault. E.g. if layers 1 and 2 connect with the fault, we want the multiplier to be 0.1, but if layers 3 and 4 connect to the fault we want a multiplier of 0.02. The keyword would be,

```
EQUALS
  MULTX  0.1   6  6  4 10  1  2  /
  MULTX  0.02  6  6  4 10  3  4  /
  /
```

and the resulting multipliers:

	Upthrown side	Downthrown side	Multiplier
Layer	2	1	0.1
	3	1	0.02
	3	2	0.02
	4	2	0.02
	4	3	0.02

The examples so far and Figure 16, have dealt with faults and multipliers in the *X*-direction, and MULTX applied to a group of cells and cells connected to this group in the direction of increasing *I*. The complete family of transmissibility multiplier keywords cover all cases. The syntax is identical, but the practical consequences are obviously different for the different cases.

MULTX Multipliers in +*X* direction (increasing *I*-index)
MULTY Multipliers in +*Y* direction (increasing *J*-index)
MULTZ Multipliers in +*Z* direction (increasing *K*-index)
MULTX– Multipliers in –*X* direction (decreasing *I*-index)
MULTY– Multipliers in –*Y* direction (decreasing *J*-index)
MULTZ– Multipliers in –*Z* direction (decreasing *K*-index)

The MULTZ and MULTZ– keywords are seldom used, and are not relevant for faults.

Note that if the negative index multipliers are used, they must be enabled by the keyword *GRIDOPTS* in the RUNSPEC section.

Multipliers are cumulative, i.e. if the same cells are modified several times, the effect will be cumulative. We demonstrate advanced use by an example. In the Figure 16 fault, we want the transmissibility multipliers to be 0.1 between 3 (upthrown) and 1 (downthrown), but 0.01 between 3 and 2. So a MULTX on layer 3 won't work, since we want to treat *3 to 1* different than *3 to 2*. For layers *2 to 1* we want a multiplier of 0.5, and connections from layer 4 should have no multiplier (i.e. multiplier = 1.0). The following combination of MULTX and MULTX– does it, (refer to the Figure while studying the example.)

```
EQUALS
  MULTX   0.1    6  6  4 10  3  3  /
  MULTX-  0.1    7  7  4 10  2  2  /
  MULTX   10.0   6  6  4 10  4  4  /
  MULTX-  0.1    7  7  4 10  3  3  /
  /
```

Line 1: Multiplier set to 0.1 from layer 3, i.e. from *(6, J, 3)* to *(7, J, 1)* and *(6, J, 3)* to *(7, J, 2)*
Line 2: Multiplier set to 0.1 from layer 2 in cells *I=7*, and affecting cells with decreasing index, i.e. cells with *I=6*. So the cell pairs affected are *(7, J, 2)* to *(6, J, 3)* and *(7, J, 2)* to *(6, J, 4)*. Since *(6, J, 3)* to *(7, J, 2)* already had a multiplier of 0.1 defined in line 1, this multiplier is now further multiplied by 0.1, and therefore the current value is 0.01 (as desired). The connection to layer 4 is wrong at this stage.
Line 3: Multiplier for cells *(6, J, 4)* to *(7, J, 2)* and *(6, J, 4)* to *(7, J, 3)* are set to 10. Since *(6, J, 4)* to *(7, J, 2)* was 0.1 it is now 1.0 (as desired)
Line 4: Fixes the connection *(6, J, 4)* to *(7, J, 3)*, which was 10.0, now 1.0.

In real models it is generally not possible to model all faults as simple as e.g. "between cells *I*=6 and *I*=7 for *J* = 4 – 10." To honour real fault geometry it is frequently necessary to let the fault trace be a

zig-zag line in the grid. In such cases the multipliers even for a single fault can become a long sequence of lines, where all combinations of MULTX, MULTX–, MULTY, and MULTY– occur. In some cases, editing and maintenance of the data file can be simplified by use of keywords FAULTS and MULTFLT, which are described below. Note that these keywords do nothing different than the "traditional" approach with the MULTX etc. keywords.

FAULTS – MULTFLT keywords (GRID section)

The FAULTS keyword is used to define the geometry of one or several named faults. Fault transmissibility multipliers can then later be defined by the MULTFLT keyword, which assigns a multiplier to the entire fault defined by the FAULTS keyword.
Note that the faults defined need not be real, complete faults. It is possible and recommended, to single out layers or groups of layers of whole or parts of a real fault, such that these can receive individual treatment later.

Syntax FAULTS:
Any number of lines, building the faults successively, each line terminated by a slash.
Syntax of each line,

Fault name ix1 ix2 jy1 jy2 kz1 kz2 Face

Fault name	Max 8 characters, used to identify fault
ix1, ix2	Lower and upper *I*-index for the fault segment defined here
jy1, jy2	Lower and upper *J*-index for the fault segment defined here
kz1, kz2	Lower and upper *K*-index for the fault segment defined here
Face	Face of the fault, which is the same as the direction the transmissibility multiplier would be. Face *X* corresponds to using MULTX, and faces *Y* and *Z* similar. (Z seldom/never used). *X–*, *Y–*, *Z–* are also permitted (must set GRIDOPTS in RUNSPEC).

There are some restrictions on the index bounds which are really obvious. If for example the face is *X*, then *ix1* must be equal to *ix2*, etc. (purpose just to exclude impossible configurations).

Example
Assume we have a fault which looks as in Figure 17 (seen from above)
The model has 6 layers, and we want to define a transmissibility multiplier of 0.1 in the positive index direction from layers 1-3 across the fault, and a multiplier of 0.015 from layers 4-6 across the fault.
We describe two different ways for defining the multipliers.

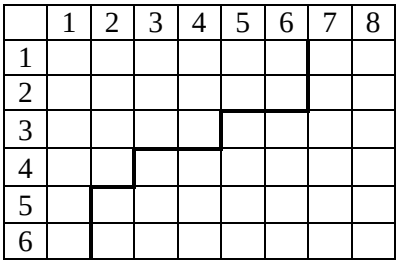


Figure 17. Example XY-view of fault.

1. Using multipliers

```
EQUALS
--array  value ix1 ix2  jy1 jy2  kz1 kz2
  MULTX   0.1   1   1    5   6    1   3  /
  MULTY   0.1   2   2    4   4    1   3  /
  MULTX   0.1   2   2    4   4    1   3  /
  MULTY   0.1   3   4    3   3    1   3  /
  MULTX   0.1   4   4    3   3    1   3  /
  MULTY   0.1   5   6    2   2    1   3  /
```

```
  MULTX   0.1   6   6    1   2    1   3  /
  MULTX   0.015 1   1    5   6    4   6  /
  MULTY   0.015 2   2    4   4    4   6  /
  MULTX   0.015 2   2    4   4    4   6  /
  MULTY   0.015 3   4    3   3    4   6  /
  MULTX   0.015 4   4    3   3    4   6  /
  MULTY   0.015 5   6    2   2    4   6  /
  MULTX   0.015 6   6    1   2    4   6  /
 /
```

2. Using FAULTS and MULTFLT keywords

```
FAULTS
-- f-name  ix1 ix2  jy1 jy2  kz1 kz2  Face
  Flt_top   1   1    5   6    1   3    X   /
  Flt_top   2   2    4   4    1   3    Y   /
  Flt_top   2   2    4   4    1   3    X   /
  Flt_top   3   4    3   3    1   3    Y   /
  Flt_top   4   4    3   3    1   3    X   /
  Flt_top   5   6    2   2    1   3    Y   /
  Flt_top   6   6    1   2    1   3    X   /
  Flt_btm   1   1    5   6    4   6    X   /
  Flt_btm   2   2    4   4    4   6    Y   /
  Flt_btm   2   2    4   4    4   6    X   /
  Flt_btm   3   4    3   3    4   6    Y   /
  Flt_btm   4   4    3   3    4   6    X   /
  Flt_btm   5   6    2   2    4   6    Y   /
  Flt_btm   6   6    1   2    4   6    X   /
 /

MULTFLT
-- f-name  Multiplier
   Flt_top    0.1    /
   Flt_btm    0.015  /
  /
```

We see that if this is a one-time job, the effort in setting up the fault and multiplier data is about the same in the two different approaches. The advantage with the latter method is seen if we want to change the multipliers to e.g. test sensitivities. Then we only have to change a single line in method 2, while more elaborate (and error prone) editing is required by method 1.

Defining a fault manually – the ADDZCORN keyword

As noted earlier, almost all grids of some complexity are constructed by dedicated software. This certainly includes fault definition and final representation of faults on the grid. Nevertheless, there are times when we would like to edit faults manually, either to do a minor fix that shouldn't require a complete grid rebuild, or non-production grids, where we would like to construct generic faults without the need for the dedicated software. The means to do so in Eclipse are limited, and can seem cumbersome and complicated, but still the users should be aware of the possibilities that exist.

The keyword ADDZCORN is used to move individual corners in cells. It can be set to continuous mode, in which case adjacent corners are moved simultaneously, or cell corners can be moved without affecting neighbour corners.

Using ADDZCORN to move all corners of a cell (or box of cells) simultaneously
The data is given in three groups,

1. The distance to move corners, DZ
2. The box to move, ix1 ix2 jy1 jy2 kz1 kz2
3. Continuity flags ix1A ix2A jy1A jy2A (default: No fault discontinuity introduced)

In addition there is a flag for keeping the top or bottom layers unchanged,

ALL: Move all corners (default)
TOP: Don't move bottom of bottom layer
BOTTOM: Don't move top of top layer.

Syntax: ADDZCORN can contain arbitrary many lines, each line with syntax,

DZ ix1 ix2 jy1 jy2 kz1 kz2 ix1A ix2A jy1A jy2A Top/Bottom-flag

Use of the continuity flags:
ix1A must be either the left hand boundary *ix1*, or the cell to the left of it (*ix1-1*). The value can be interpreted as, the first corner in the range to move. Hence, if *ix1A* is equal to *ix1*, cell *ix1* is moved, but cell *ix1-1* is left unchanged, whereby a discontinuity (fault) has been constructed. On the other hand, if *ix1A* is equal to *ix1-1*, then the right hand side of cell *ix1-1* is moved (all associated corners are moved simultaneously), but the left hand edge of the cell is left unchanged. No discontinuity has hence been introduced. See Figure 18 for a visualisation of the use of the left hand (lower index) flag in the *x*-direction. The other flags act similar.

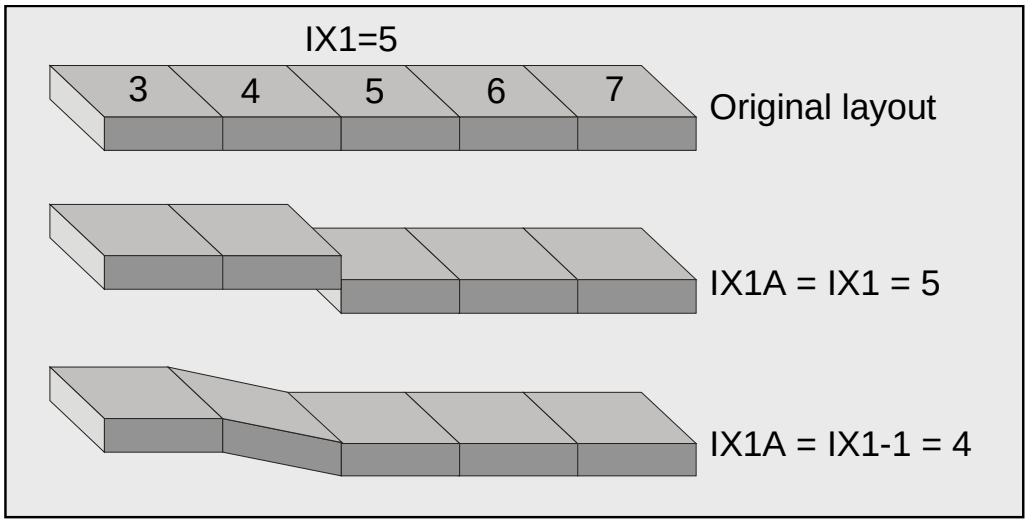


Figure 18. Example use of continuity flags in ADDZCORN, left hand x-direction

Example
We wish to move part of the grid downwards 20 m. The box to move is *I = 5-10, J = 3-8*. When moving the box, the left and right hand boundaries shall be discontinuous (fault) like in the middle example in Figure 18. The other two edges (*J*-direction) shall be continuous, like the lower example in the figure. The syntax is,

```
ADDZCORN
-- DZ  ix1 ix2  jy1 jy2  kz1 kz2 ix1A ix2A jy1A jy2A TB-flag
  20.0  5   10   3   8    1   6   5     10   2    9   /
  /
```

Notes:
1. The continuity flags are only used when the box edge is continuous before the move. If a box edge prior to the move is discontinuous, like the middle example in Figure 18, then the box will be moved as discontinuous, and the rest of the grid left unchanged irrespective of the value of the continuity flag
2. All the indices *ix1, ix2,..., jy1A* must be within the grid index limits. I.e. if e.g. *ix2* is at the right hand edge of the grid, it is an error to define *ix2A = ix2+1*. When the box extends to the edge of the grid the concept of continuity at the edge of the box becomes irrelevant anyway.
3. Normally all the layers are moved simultaneously. If only some layers are moved, the user must take care to avoid unintended overlaps or gaps after the move.
4. Note that the default values of the continuity flags imply continuity (no fault).

Using ADDZCORN to move individual corners of a single cell

The way we used ADDZCORN above, all the eight corners of the cell were moved rigidly. At times it is needed to move corners individually, either a single corner, or some of the corners. In Eclipse this is done by a rather special syntax in the ADDZCORN keyword. If at least one of *ix1, ix2, jy1, jy2* is set to zero, this is a signal that we are operating in single-cell, individual corners mode.
When specifying *ix1 ix2* or *jy1 jy2* a nonzero value means, "move this corner of cell with defined index", where "this corner" is defined by the position of the nonzero index. This is best explained by examples, where IV means a nonzero index value:

ix1	*ix2*	
IV	0	Move left hand corner of cell with *ix = IV*
0	IV	Move right hand corner of cell with *ix = IV*
IV	IV	Move both left and right hand corners of cell with *ix = IV*
jy1	*jy2*	
IV	0	Move "back" (northernmost) corner of cell with *jy = IV*
0	IV	Move "front" (southernmost) corner of cell with *jy = IV*
IV	IV	Move both "front" and "back" corner of cell with *jy = IV*

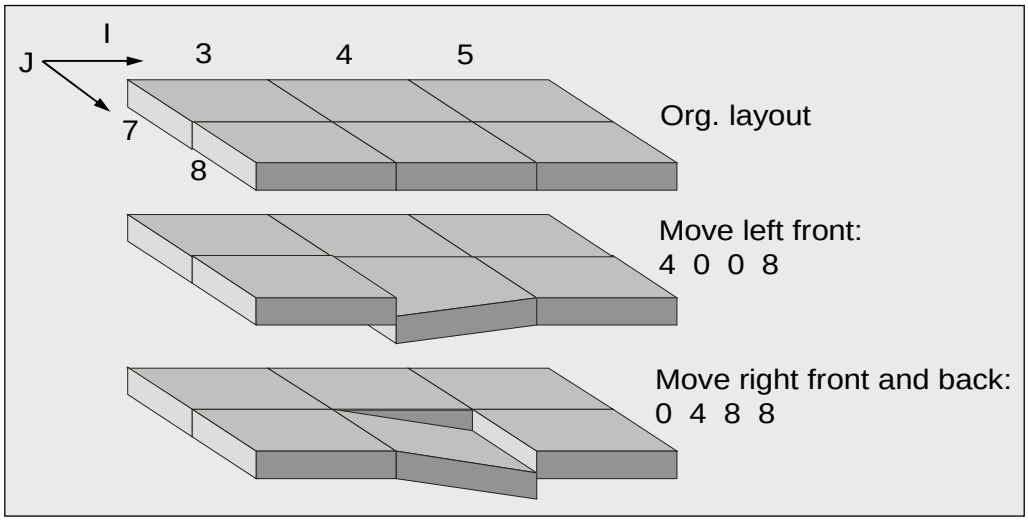


Figure 19. Examples of use of combinations of cell index and zeros to move individual corners

Some examples of use are shown in Figure 19. In the figure it has been assumed that only the defined cell is affected. The continuity rules to surrounding grid are valid as in the section above, i.e. if the continuity flags are set to the current cell, the movement will not affect neighbour cells, while if the flags are set to the neighbour cell index, the movement will be continuous.

So the complete syntax for the middle example in Figure 19 would be, assuming all six layers are moved,

```
ADDZCORN
-- DZ  ix1 ix2  jy1 jy2  kz1 kz2 ix1A ix2A jy1A jy2A TB-flag
  15.0  4   0    0   8    1   6   4     5    7    8    /
```

14. Aquifer Modelling (GRID section)

When a reservoir is produced by pure depletion, the only energy supply available is fluid and rock expansion. The total fluid expansion energy is obviously proportional to the total fluid volume. Hence if it as an example is possible to produce 5% of total fluid volume by pure expansion energy, then the recovery fraction will be much lower if all of the fluid is oil (5% of oil present) than if the total volume is to a large extent water (recovery 5% of oil + water).

Fortunately (in a recovery perspective) many reservoirs are connected to an aquifer, i.e. a huge volume of water. When including such aquifers in the simulation model, it is of course possible to model the water-filled volume explicitly, building the grid sufficiently large to cover the entire water volume. This will normally require many grid blocks, and can increase the model size considerably.

An alternative is therefore to model the aquifer by some artificial means. There are several ways to do this, but we will concentrate on the so-called *numerical aquifers*, which also can be modelled in Eclipse. The reasons for using a special model for the aquifer, are

- Practical: it saves memory space and computer time
- No loss of accuracy. Since aquifers are large one-phase volumes, the computations are simple, and as accurate on large grid blocks as on small.
- Detail water flow pattern is not needed in the water zone, only the energy (pressure) support from aquifer to reservoir, and water flux between the two are needed.

Actually, the flux and energy support can be computed with sufficient accuracy in a one-dimensional abstraction of the aquifer, which is how it is implemented. With this knowledge we see that a sufficient description of an aquifer doesn't require shape, only volumes. However, the interface between the aquifer and the reservoir requires a shape definition, which is only logical.

Hence there are two aspects of setting up an aquifer, the *aquifer definition* (volume, properties), and how it *interfaces* to the reservoir.

Aquifer definition

The way Eclipse does it, ordinary grid cells in the reservoir are used to define aquifers. The cells will have all their properties redefined, but apart from that they are still part of the grid, and cannot be regarded as having been removed from the grid. E.g. connectivity between grid cells is not changed just because a cell is defined as an aquifer cell. This means that aquifer cells should be completely isolated from the rest of the reservoir, i.e. completely surrounded by inactive cells.

Most aquifers can be satisfactory defined by one grid cell only. However, if the aquifer properties and/or depth have large variation, it can be a good idea to try to capture this variation by using more cells to define the aquifer. More than three or four cells should seldom or never be necessary.

> To verify this it is a nice exercise to build a simple grid including a large water zone which is explicitly modelled. Then model the water zone as a numerical aquifer using one, two, four and ten cells. Most probably, the results from these cases will be as good as equal.

Figure 20 shows some examples of ways to use grid cells to define aquifers, and the connectivity to expect.

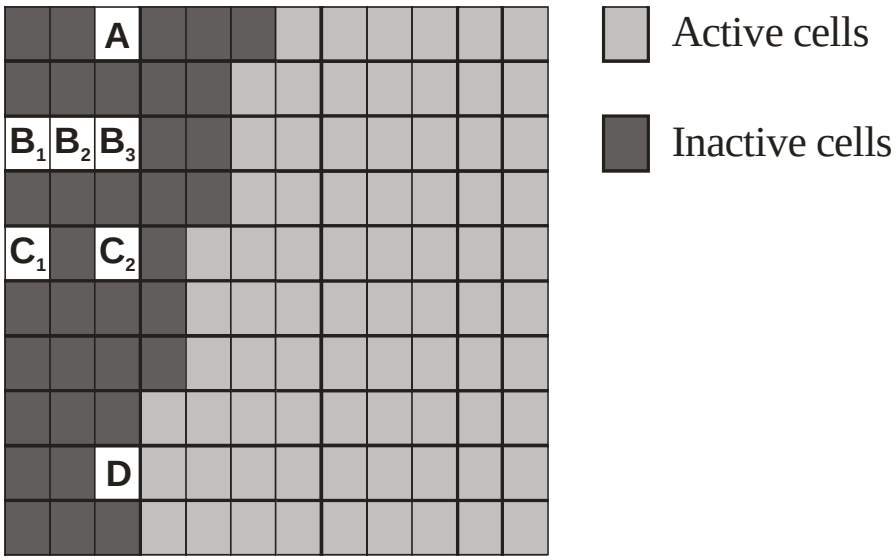


Figure 20. Examples of use of inactive cells to define aquifers (grid viewed from above)

Cell A is a typical example of using an isolated inactive cell to define an aquifer. The cell has no connection to the surrounding world at all, so the necessary flow connections will be defined by the user.

Cells B_1, B_2, B_3 are connected to each other, but isolated from the rest of the grid. The three cells can either be used to define a single aquifer, which obviously will have internal communication, or the cells could define two or three different aquifers, such that the aquifers communicate with each other.
Cells C_1 and C_2 are isolated from each other, and from the rest of the world. They can be used to define two different aquifers. Using such a configuration for a single aquifer would be meaningless, as there would be no connectivity between the two different parts of the aquifer.
Cell D is *not* suited for use as an aquifer cell, since it is in direct contact with an active cell in the grid. In addition to the connectivity defined by the user, fluid in cell D will flow directly into the reservoir, which is probably not intended.

Having decided which (inactive) cells from the grid to use as aquifer cells, the keyword to *define* the aquifer is AQUNUM (numerical aquifer). The keyword data is comprised of arbitrary many lines, each line containing definition for exactly one aquifer cell on the original grid. The syntax for each line is,

AQ-id AI AJ AK X-sect_area Length Poro Perm Depth P_init PVT-table SatNum

AQ-id
: an *integer* identifying the aquifer. If several lines are used to define the same aquifer, only the cell defined in the first line of definition will be in contact with the reservoir, as defined by the AQUCON keyword below. (**Note:** Aquifers are identified by integers, not names)

AI AJ AK
: The (*I, J, K*)-index for the grid cell to use as aquifer cell

X-sect_area Length
: The gross volume of the aquifer cell is the product of these two numbers. Recall that the aquifer is modelled as one-dimensional – flow is assumed to be in the direction defined by *Length*, and flux out of the aquifer is through the cross sectional area defined by *X-sect_area*.

Poro Perm Depth
: Normally we will define porosity, permeability and a typical depth for the aquifer, taken from known properties of the water zone. If these items are defaulted, the values will be taken from the grid cell as it is defined in the grid.

P_init
: Initial pressure in the aquifer cell. *It is recommended to default this value*, whereby it will be computed from the fluid gradients and the depth defined above.

PVT-table SatNum
: These items are best defaulted. (Ref Eclipse documentation if alternative is needed.)

Example.
We will define four different aquifers, corresponding to examples A, B, and C in Figure 20. Cell A has aquifer-id 1, aquifer-id 2 is used for cells B, and aquifer-id 3 and 4 for cells C. The view in Figure 20 is assumed to be the top layer (*K*=1), at the edge of the reservoir (*I*=1 at left end, *J*=1 at top).

```
AQUNUM
-- AQ-ID  AI AJ AK  X-sect  Length  Poro  Perm  Depth P_init
     1     3  1  1   2E7     3000   0.3    800   3000   1*   /
     2     3  3  1   6E6     2500   0.28  1000   2500   1*   /
     2     2  3  1   6E6     2500   0.28   800   2800   1*   /
     2     1  3  1   6E6     2500   0.28   500   3100   1*   /
     3     1  5  1   5E7     6000   0.25   300   3500   1*   /
     4     3  5  1   4E6     4000   0.24   100   2700   1*   /
  /
```

Note that the dimensions, petrophysics, and depth of the aquifer cells bear no relation to what was defined in the grid originally at all.
Flow between aquifer 2 and the reservoir is only to/from aquifer cell (3, 3, 1), since this cell is the first to be defined in the keyword. Also note the reason for using three cells for aquifer 2, to capture the depth increase (higher fluid pressure, larger compressive energy) and decrease in permeability with depth.

Aquifer connection to reservoir

Each aquifer connects to the reservoir in one aquifer cell only. This cell will, however typically connect to a large number of grid cells in the reservoir. To mimic the aquifer to reservoir flux, also the side of the reservoir grid blocks which the influx is coming must be specified.

The AQUCON keyword is used to specify which cells (in the reservoir) a given aquifer (defined by the *aquifer id*) connects to. The data is comprised of arbitrary many lines (building up the entire definition of all connections), each line with syntax,

AQ-id ix1 ix2 jy1 jy2 kz1 kz2 Face TranMult (4)*

AQ-id refers to the id used in AQUNUM
ix1 ix2 jy1 jy2 kz1 kz2 defines the *grid* box which the aquifer connects to. Normally the two indices are equal in one of the directions, so that the box is not a volume, but corresponds to a surface.
Face is the *reservoir* grid block face to which the aquifer is connected. The choices are,

- ***I–*** Left hand face of cell (face in negative I-direction)
- ***I+*** Right hand face (positive I-direction)
- ***J–*** Back face (negative J-direction)
- ***J+*** Front face (positive J-direction)
- ***K–*** Top face (negative K-direction)
- ***K+*** Bottom face (positive K-direction)

TranMult is used to define a transmissibility multiplier between the aquifer cell and reservoir. The relevant transmissibility will be computed from the aquifer cell permeability and the receiving grid cell permeability, and can be modified by setting *TranMult* different from 1.0 (the default value).

Example
Aquifer 1 (cell A in Figure 20) shall be connected to the northern edge of the reservoir grid. The relevant cells are the row J = 1, I = 7-12, and assuming the grid is comprised of six layers which all connect to the aquifer, K = 1-6. Since the aquifer connects to the northern face of the cells, the appropriate face is '*J–*'. We set the transmissibility multiplier to 0.75. The syntax for this connection would be,

```
AQUCON
-- AQ-ID  ix1 ix2  jy1 jy2  kz1 kz2  Face  TranMult
     1     7   12   1   1    1   6    J-     0.75   /
  /
```

By normal use, the reservoir grid surface defined by the index box and 'face' is a reservoir edge, or connects to inactive blocks only. This condition should always be satisfied (although a flag exist to relax the condition (not recommended!), see Eclipse documentation).
The keyword **AQUDIMS** must be set in the RUNSPEC section.

15. Local Grid Refinement

In many problems we need a higher resolution (finer grid) than our grid permits. An example is where we model gas coning near a horizontal well. With a high resolution description as in Figure 21, we can track the gas front accurately, and give a good estimate for time and position of the gas breakthrough in the well. Also, the cells are sufficiently small that they can be classified as either gas filled or oil filled.

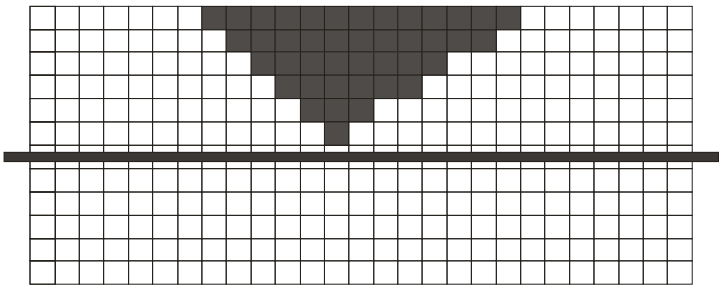


Figure 21. Gas cone near a horizontal well, fine grid, vertical cross section.

When the same problem is modelled on a coarser grid, we see that the shape of the cone is completely lost, and the front is no longer clearly defined, as all the relevant cells have an intermediate gas saturation (saturation larger in darker cells). Also neither the time of the breakthrough nor the exact place where it happens can be deducted from the simulation.

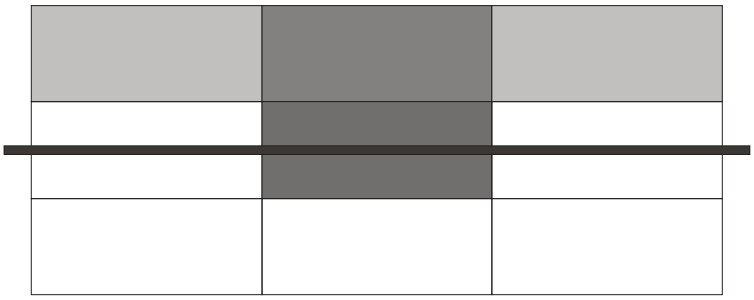


Figure 22. As figure 21, but on a coarse grid.

Using the resolution of Figure 21 on the entire grid is typically not possible due to memory limitations and computing time. One possibility is to extend the fine grid in all directions with coarser cells, as in Figure 23. This is, however, not a recommended solution, since the resulting long and narrow cells are sources of computational errors, especially when the size difference between large and small cells in the grid becomes too large.

In such situations it is much better to use local grid refinement (LGR). As the name implies, this means that part of the existing grid is replaced by a finer one, and that the replacement is done locally. An example of how this looks can be seen in Figure 24.

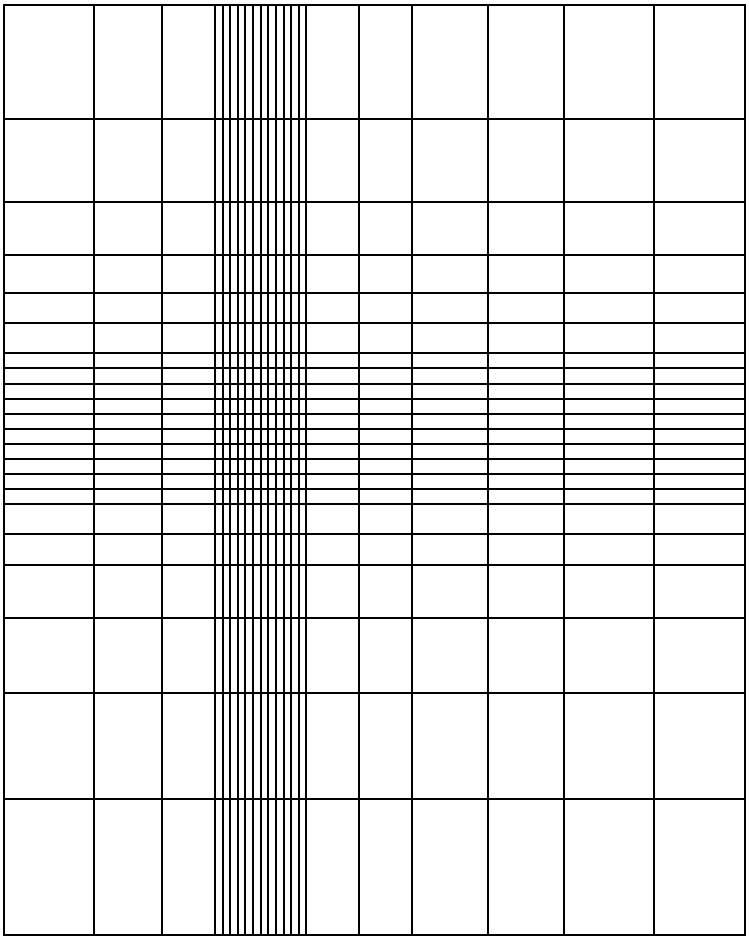


Figure 23. Extending resolution of fine cells non-uniformly, XY-view

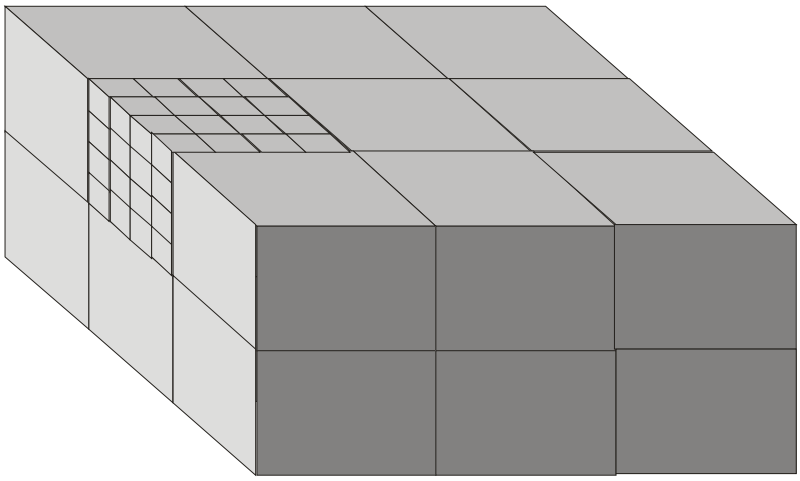


Figure 24. Example where one cell in original grid has been replaced with an LGR

The LGRs which will be discussed in this context are regular cartesian. The appropriate keyword is then CARFIN (Cartesian refinement). Basically a box in the grid is replaced by another box with more cells, but there are some constraints (see *nx ny nz* below). The keyword is followed by *one* line of data, terminated by a slash. Note that only one LGR can be defined in one CARFIN keyword. The keyword must be repeated for each new LGR. Keyword ENDFIN terminates current CARFIN.
The syntax is then,

```
CARFIN
--             -- Box to refine --  Cells in LGR
-- LGR-name    I1 I2  J1 J2  K1 K2   nx  ny  nz  MaxWells
```

LGR-name

Each local grid must have a unique name. All later references to the LGR is by its name

The "Box to refine" is a standard grid box which we have encountered many times now.

nx ny nz

These numbers define the total number of cells in each direction in the LGR. The numbers can however not be chosen entirely freely. Basically, the constraint is that the cell edges in the old (coarse) grid must coincide with edges in the refined grid. Or, the number of refined cells in *each* of the old cells must be an integer. So e.g. two coarse cells cannot be divided into five fine cells, as that would give 2.5 fine cells in each coarse block.

MaxWells

As we shall see eventually, most well keywords must be redefined when wells are defined on a local grid. Therefore, the maximum number of wells permitted on the LGR must be given already when the local grid is defined.

Example

The box *I = 13-15, J = 4-5, K = 3* shall be refined. On the *coarse* grid, the dimensions of the box are, *NCI* = 3, *NCJ* = 2, *NCK* = 1. Therefore *nx* must be divisible by three, and *ny* must be divisible by two. It's often easier to think in terms of how many fine cells go into each coarse cell. In this case, we divide each coarse cell into three cells in the *x*-direction, 5 cells in the *y*-direction, and 4 cells in the *z*-direction. That way, *nx = 9* (three coarse cells, three fine in each), *ny=10* (two coarse cells, five in each), and *nz=4*. So the CARFIN keyword becomes,

```
CARFIN
--             -- Box to refine --  Cells in LGR
-- LGR-name    I1 I2  J1 J2  K1 K2   nx  ny  nz  MaxWells
    ExLGR1     13 15   4  5   3  3    9  10   4     1     /
```

We see that an LGR must be defined on a BOX (no irregular shape allowed). Moreover LGRs must not overlap. They are not even allowed to touch each other without special treatment (see below). LGRs can be nested, but that is not discussed here.

When the simulation includes local grid refinements, this must be notified in the RUNSPEC section by the *LGR* keyword (see page 82).

15.2 LGR on an irregular volume – Amalgamation

If we want an LGR on a volume that is not a regular BOX, this can be done by amalgamating several local grids into one. Each of the LGRs must be defined in the standard manner, and hence be on a regular BOX. There is no limit to the number of LGRs that can be amalgamated.

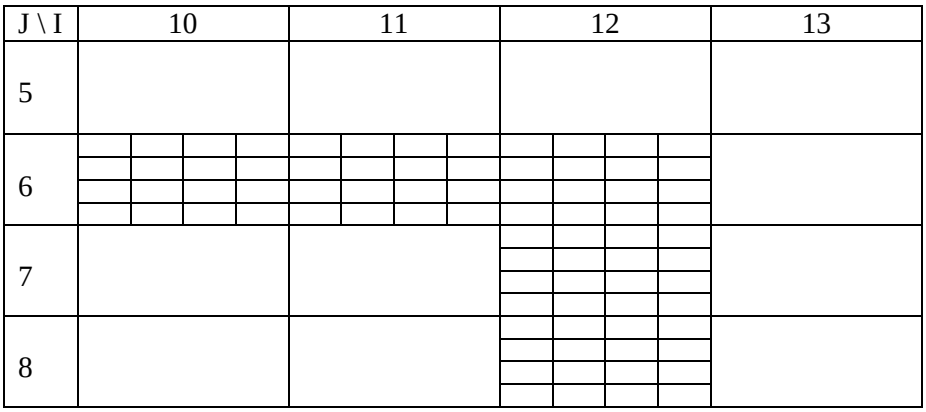


Figure 25. Example LGR on irregular area (XY view)

To define an LGR on an area as in Figure 25, first define one LGR for each regular area, such that the boxes touch each other but do not overlap. Then amalgamate these, as,

```
CARFIN
--             -- Box to refine --  Cells in LGR
-- LGR-name    I1 I2  J1 J2  K1 K2   nx  ny  nz  MaxWells
    LGR1       10 12   6  6   1  1   12   4   4     1     /

CARFIN
--             -- Box to refine --  Cells in LGR
-- LGR-name    I1 I2  J1 J2  K1 K2   nx  ny  nz  MaxWells
    LGR2       12 12   7  8   1  1    4   8   4     1     /

AMALGAM
  LGR1  LGR2  /
  /
```

Wildcards are permitted in the AMALGAM keyword, so the last keyword could have been written,

```
AMALGAM
  'LGR*' /
  /
```

The LGRs must be compatible, as in the example both LGRs have *nx* = 4.
It is possible to define PORO, PERM, ... on LGRs, and also more complex geometry. The default is to inherit the properties from the parent cell, which we will be content with here.

15.3 Wells on local grids – Horizontal wells

The keywords for defining wells and completions for wells on a local grid are equivalent to the standard keywords, except,

1. The name of the LGR must be supplied (so Eclipse knows where it is working)
2. The completion indices are with reference to the LGR, not the original grid.

Example
A horizontal well is defined on a standard grid as in Figure 26. Assume the well is at a depth equal to the centre of layer 2.
Figure 27 shows the same configuration using local grid refinement. We set up the appropriate well keywords for both cases.

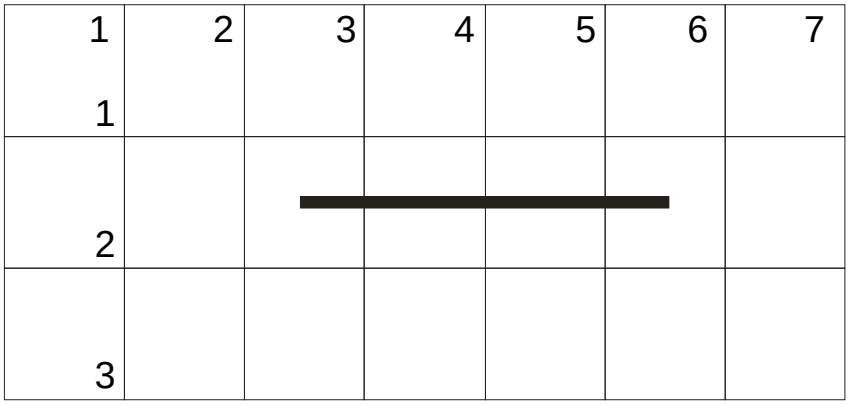


Figure 26. A horizontal well on a standard grid (XY view)

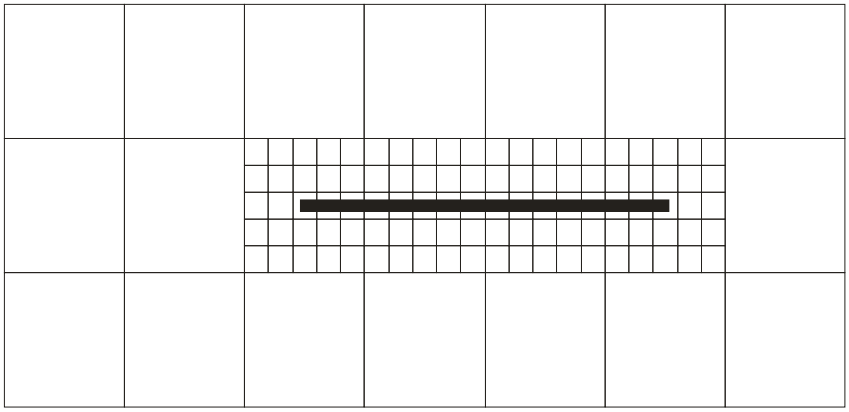


Figure 27. As Figure 26, but using LGR near the well

For the standard grid (Figure 26) we would use (with well name WH1),

```
WELSPECS
-- Wname  Gname  IWH  JWH  Z_BHP  Phase  N/A  N/A  ShutIn  XFlow
    WH1    G1     3    2   1840.0  OIL    1*   1*   STOP    YES   /
  /

COMPDAT
-- Wnm*  IC JC KTOP KBTM  OPEN?  SatTbl  N/A  Dw  Kh  Skin  N/A  Dir  r0
   WH1    3  2   2    2   OPEN     0      1*  0.3  1*   0    1*   X   1*  /
   WH1    4  2   2    2   OPEN     0      1*  0.3  1*   0    1*   X   1*  /
   WH1    5  2   2    2   OPEN     0      1*  0.3  1*   0    1*   X   1*  /
   WH1    6  2   2    2   OPEN     0      1*  0.3  1*   0    1*   X   1*  /
  /

WCONPROD
...
...
```

We note that the COMPDAT keyword is tailored for vertical wells, where a range of completions can be defined at once, by the *KTOP – KBTM* interval. For horizontal wells, there is no such shortcut, so the COMPDAT keyword frequently grows to a large number of lines when horizontal wells are modelled. This will get even worse on local grids.

It may be surprising that Eclipse does not have a similar shortcut for horizontal wells as for vertical. The truth is, “nobody” inputs this data for hand anyway. Since well and production data are so complex and voluminous in real-life simulations these data are almost always generated by external software, typically the SCHEDULE program. Since “nobody” needs a simple input syntax for horizontal wells anyway, the Eclipse team hasn’t taken the trouble to implement it.

For horizontal wells, the well head index *(IC, JC)* refers to the heel of the well (which is the end of the well that is closest to the surface)

For the LGR case (Figure 27), the appropriate keywords would be (first the LGR definition, then the well keywords)

In the RUNSPEC section add a keyword

```
LGR
-- MAXLGR MAXCLLGR MAXAMLGC MAXAMLGF MAXLGRAM LSTACK INTERP?
     1      500       1*       1*      1*       1*      1*  /
```

(Max nbr. of LGRs in model; Max nbr of cells in one LGR; Max nbr. of amalg. coarse cells; Max nbr. of LGR amalgamations; Max nbr. of LGRs in one amalgamation; NSTACK for LGRs (NSTACK is defined on page 100); Pressure interpolation? (INTERP or NOINTERP (default)))

In the GRID section,

```
CARFIN
--            -- Box to refine --  Cells in LGR
-- LGR-name   I1 I2  J1 J2  K1 K2   nx  ny  nz  MaxWells
    WH1LGR     3  6   2  2   2  2   20   5   5     1     /
```

In the SCHEDULE section,

```
WELSPECS
...

-- LGR wells must be defined with special keywords WELSPECL and COMPDATL
WELSPECL
-- Wname  Gname  LGRname IWH JWH Z_BHP  Phase  N/A  N/A  ShutIn  XFlow
   WH1     G1     WH1LGR  3   3  1840.0  OIL    1*   1*   STOP    YES  /
 /

COMPDAT
...

COMPDATL
--Wnm* LGRname IC JC KTOP KBTM OPEN? SatTbl  N/A  Dw  Kh Skin N/A Dir  r0
  WH1   WH1LGR  3  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
  WH1   WH1LGR  4  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
  WH1   WH1LGR  5  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
  WH1   WH1LGR  6  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
  .
  .
  .
  WH1   WH1LGR 14  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
  WH1   WH1LGR 15  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
  WH1   WH1LGR 16  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
  WH1   WH1LGR 17  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
  WH1   WH1LGR 18  3   3    3  OPEN    0     1*   0.3  1*  0   1*  X   1* /
/

WCONPROD
...
```

Note that all references to cell indices are relative to the local numbering scheme in the LGR.
Note: For complex well paths ECLIPSE sometimes gets the well path all wrong. The user can override the default well-path-construction by the keyword COMPORD, whereby the completion order can be set exactly as defined by the user.

15.4 Horizontal wells and friction

So far we haven't mentioned the effect of friction between well casing and flowing fluids. Vertical or moderately deviated wells are so short that the friction doesn't matter, but in long horizontal wells it may have a significant impact on the well flow. The friction results in an additional pressure drop along the well, which is calculated based on the well diameter and the roughness of the casing. The length of the well is always measured from the bottomhole reference point, which for a horizontal well was the heel.

Note: When a well is defined as a friction well, item 12 in the WELSPECS keyword must be set to 'SEG' (the segmented density calculations must be used). (If the well was defined by WELPSECL, the relevant item is item 13.)

The keyword for defining friction data is WFRICTN. This keyword can be used for wells on the standard grid, and for wells on a single LGR. For wells on an amalgamated LGR, keyword WFRICTNL must be used (this keyword can also be used on single LGRs).

Each WFRICTN (or WFRICTNL) is used to define one (and one only) friction well, so the keyword must be repeated for each well.

The keyword is divided into two parts, the first part for the well as such , the second part for data along the well. Each record is terminated with a slash, an empty record terminates the keyword.

We describe WFRICTN here, the difference for WFRICTNL is treated at the end.

Record 1,
WellName TubingDiameter Roughness FlowScaleFactor /

TubingDiameter is normally the same as the diameter defined in the COMPDAT keyword, but since the diameter is used for different calculations in the two cases it is more flexible to define this item also here. The diameter can vary along the well, in which case it can be redefined in the next record.

Roughness, a number, obtained from measurements

FlowScaleFactor, a scaling factor. In this context we set it default, which is 1.0

Subsequent records

I0 J0 K0 Distance0 Distance1 WellDirection IJK_end TubingDiameter /

I0 J0 K0
Indices for the current cell or start of a range of cells. ("Start" here always means "nearest the reference point", i.e. the heel)

Distance0
Distance from the reference point to the start of the perforations defined in this record

Distance1
Distance from the reference point to the end of the perforations defined in this record

WellDirection
The direction the well penetrates the cell(s) defined in this record (*X, Y, Z*; alt. *I, J, K*)

IJK_end
If a range of cells is defined (not a single cell), the direction of the range is defined in the item above. So the end of the range is defined by a single index, not a triplet.

TubingDiameter
Specify here if different from the value defined in record 1.

Notes
When Eclipse computes the effect of the friction, it needs the effective friction lengths. These are obtained from the *Distance0* and *Distance1* items. With a fixed reference point *P0*, *Distance0* is the distance from *P0* to the start of the perforations, and *Distance1* is the distance from *P0* to the end of the perforations. The friction length is the difference *Distance1 – Distance0*. If *Distance0* is defaulted, it is set to *Distance1* of the previous record. If *Distance1* is defaulted it is set to *Distance0* (this record)

plus the diameter of the cell, which is the length of the cell in the direction set in *WellDirection*. I.e., if the distances are defaulted, Eclipse computes them as if the well path crosses the cell in the direction defined by WellDirection, and the well is continuously perforated (the perforation length is equal to the cell diameter).
If the actual well path is aligned along one of the cell directions, the default calculation is correct. In such cases, the automatic *range* definition can be used, by specifying *IJK_end*. Once a well direction has been specified, Eclipse knows which cells are affected. If e.g. WellDirection is set to '*X*', then *J0* and *K0* will be the same along the segment, only *I* will change, so the *IJK_end* will be the *I*-index for the cell at the end of the range.
Default-values: Those values which are not needed can be defaulted. (If *Distance0* and *Distance1* are given, *WellDirection* and *IJK_end* are not needed, if *Distance1* is defaulted, *WellDirection* is needed, and *IJK_end* if appropriate).
The *Distance0* and *Distance1* terms are especially relevant if the grid approximation to the well path deviates much from the actual well path, so that Eclipse-computed friction lengths will come out wrong.

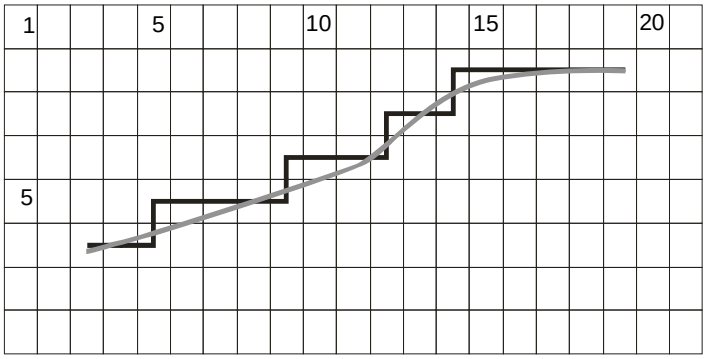


Figure 28. Actual well path (grey) and grid approximation (black) (XY-view)

A typical approximation is shown in Figure 28. The actual well path must be approximated by a path passing through only cell centres. The best fit is therefore something like the black zig-zag-line. We see that both perforation lengths and the length of the well itself is quite different if measure along the zig-zag path than along the actual path. Assuming that the left hand end of the well is the heel, we see that in the last few cells before the toe (right hand end) actual distance is equal to grid well distance, but for the rest of the well there is some difference. When this well is defined as a friction well, the appropriate friction lengths should therefore *not* be auto-computed by Eclipse.
The grid blocks are 100 m in the *x*-direction, and 150 m in the *y*-direction, and the well is assumed horizontal in layer 3.
We let cell (3, 6, 3) be the well heel, i.e. the bottomhole reference point, i.e. the point which we measure all lengths from.

Then one way of defining the friction in the well would be,

```
WFRICTN
-- Wellname  TubingDiam  Roughness  Scale
   WH1          0.35        3E-4    /
-- I0 J0 K0 Dist0  Dist1  WellDir  IJK_end  TubingDiam
    3  6  3   0.0   50.0  /
    4  6  3  50.0  155.0  /
    5  5  3 155.0  265.0  /
    6  5  3 265.0  375.0  /
    7  5  3 375.0  485.0  /
    8  5  3 485.0  595.0  /
    9  4  3 595.0  705.0  /
...
...
   14  2  3  1*     1*      X        19  /
 /
```

In all records, except the last we have explicitly defined the tubing lengths, which are a little longer than the cell diameter of 100m, since the well path is partly diagonal to the grid. Since the distances have been defined, the rest can be defaulted.

In the last record, we are starting on the well segment which is parallel to the *x*-axis, in grid cell (14, 2, 3). When *Dist0* is defaulted, the "current" friction length is taken as the last known value (*Dist1* from the record above). When *Dist1* is defaulted, it is computed as:
First the cell diameters are computed. Since the well direction is *X*, the diameter is 100m. Then we have defined a range – with direction *X* and start cell (14, 2, 3), the end index 19 means I=19, such that the end cell becomes (19, 2, 3). The range is hence six cells, and the total length is 600m (6*100), which is added to the *Dist0*.

Amalgamated grids
When WFRICTNL is used, the only difference is that in all records after the first, the first item is the LGR-name. The rest of the items are as before. If a range is used it must not span more than one LGR.

16. Numerical Solution of the Flow Equations

The objective is to give a "bird's eye view" of how the flow equations are solved, without going into too much details. For such a discussion, the actual flow equations are not needed – it suffices to look at a representative equation containing the necessary features. The model equation will be,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \tag{25}$$

Using central differences for the spatial approximation and forward difference in time, the finite difference approximation to Equation (25) is,

$$\frac{u_{i,j,k}^{n+1} - u_{i,j,k}^{n}}{\Delta t} = \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{(\Delta x)^2} + \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{(\Delta y)^2} + \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{(\Delta z)^2} \tag{26}$$

In Equation (26), superscripts have been used to denote the time step.
However, no time step has been indicated on the right hand side of the equation.

If the right hand side is evaluated at time step *n*, then the equation can be solved *explicitly* for the solution $u_{i,j,k}^{n+1}$, which makes finding the solution at time *n+1* for all cells *(i, j, k)* easy.

On the other hand, if the right hand side is evaluated at time *n+1*, then all the terms except $u_{i,j,k}^{n}$ are unknown, and the system defined by Equation (26) when *i,j,k* take all values must be solved all at once, which is probably not straightforward. We denote this an *implicit* formulation of the problem, with an implicit solution procedure.

From numerical analysis it is known that the implicit solution is much more stable than the explicit solution, and the time steps can be larger. So even if an implicit solution scheme is harder to implement than an explicit scheme, it is generally worth the effort. As an example of the benefit, a *stable* scheme means that it will be *self-correcting*. I.e., if the solution at one time step has become erroneous for some reason, then the numerical solution will approach the "correct" solution at subsequent time steps. An explicit scheme does not in general have this property, which means that once an error starts to develop, it will grow with time, so that the solution gets worse and worse.

Both explicit and implicit schemes have been used in reservoir simulation through the years, but explicit schemes are not widely used any more. The default scheme in Eclipse is the implicit scheme, but IMPES (see below) is available as an option.

From Equation (26) it is apparent that the cells that enter each calculation are given by the 7-point stencil. I.e. to compute *u* in cell *(i, j, k)*, we need the values in cells *(i±1, j, k), (i, j±1, k),* and *(i, j, k±1)*.

As an example, we will build the system structure for a grid with 4 cells in the *x*-direction, 3 in the *y*-direction, and 2 in the *z*-direction, i.e. *nx=4, ny=3, nz=2*. When we organize the system of equations we need to organize all the cells in one vector. We do this by the *natural ordering*, which is the same as the book order.

Then cell *(1, 1, 1)* will depend on *(2,1,1), (1,2,1),* and *(1,1,2).* But in the natural ordering that is equivalent to cell 1 depends on cells 2, 5 and 13. In the same manner, cell *(2,1,1)* will depend on *(1,1,1), (3,1,1), (2,2,1),* and *(2,1,2),* i.e.: cell 2 depends on cells 1, 3, 6, and 14. (Ref. Figure 29)

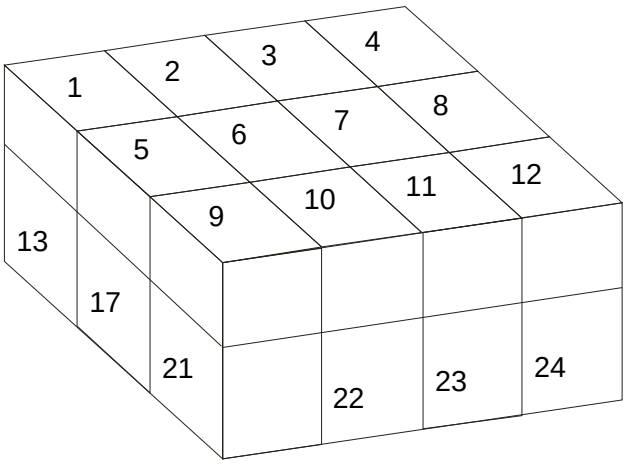


Figure 29. Natural ordering of a 4 x 3 x 2 grid

Continuing this way we can draw a table (matrix) which depicts all the connections.

	1				**5**					**10**					**15**					**20**				
1	x	x			x								x											
	x	x	x			x								x										
		x	x	x			x								x									
			x	x				x								x								
5	x				x	x			x								x							
		x			x	x	x			x								x						
			x			x	x	x			x								x					
				x			x	x				x								x				
					x				x	x											x			
10						x			x	x	x											x		
							x			x	x	x											x	
								x			x	x												x
	x												x	x			x							
		x											x	x	x			x						
15			x											x	x	x			x					
				x											x	x				x				
					x								x				x	x			x			
						x								x				x	x	x			x	
							x								x			x	x	x			x	
20								x								x			x	x				x
									x								x				x	x		
										x								x			x	x	x	
											x								x			x	x	x
												x								x			x	x

Figure 30. Coefficient matrix for a 4 x 3 x 2 grid (x means "nonzero element")

This structure is characteristic for problems using the 7-point scheme and natural ordering. The features are,

- The matrix is *sparse*, most of the entries are zero
- Along the main diagonal there are three rows, i.e. a three-diagonal main.
- At each of the sides of the main diagonal there are two single bands – for this reason this kind of matrix is called a 7-diagonal matrix.
- The main diagonal is "blocked", each block comprised of *nx* x *nx* lines / rows (*nx = 4* in this example).
- The first side-band is at a distance *nx* (4 in example) from centre.
- The second band is at a distance *nx*ny* (12 in example) from centre. (This band is missing in two-dimensional problems.)

Verify that the set-up leads to this structure always, and picture the scheme for larger problems.

Comments

- The structure in Figure 30 is a result of the way we numbered the cells (natural ordering). Other orderings are also commonly used, the objective being to get a structure which is optimally suited for some solution procedure. We won't pursue this aspect, but note that all such (sensible) ordering schemes results in a sparse banded matrix.
- Solution of systems of equations with a sparse banded structure has received a great deal of attention through the years, and many efficient and specialized routines exist.
- The black oil equations are actually a little more complex than our model example, e.g.
    - Each "element" in the matrix is actually a 3 x 3 matrix (2 x 2 for two phase problems), as the primary variables are not scalars, but typically the vector (S_w, S_g, p_o).
    - The black oil equations contain strongly nonlinear terms, as e.g. the mobility

      $$\lambda_l = \frac{k_{rl}(S_l)}{\mu_l(p_l)B_l(p_l)},\ l = o, w, g$$

      The system in Figure 30 is therefore not a *linear* system of equations as known from linear algebra.
- The "nice" structure of the matrix is a result of strict adherence to the 7-point stencil. Recall from the discussion on faults that honouring sand-to-sand contact flow gives rise to non-neighbour connections, which in this context are characterised by *not* adhering to the 7-point stencil. NNC-terms will generate non-zero elements outside the 7 standard bands, and also zero elements on the tridiagonal, which can have a deteriorating effect on convergence properties. For grids with very many faults, so many off-diagonal terms are generated that the matrix often can no longer defend to be called *sparse*.
  The specialized methods which have been developed to solve the 7-diagonal system are therefore often not suited for the real problems that evolve in grids with NNCs. The solution method used by e.g. Eclipse is perhaps not the fastest for all problems, but it has proven itself to be *robust*, i.e. it manages to solve most problems that arise.

Before we continue to describe the solution scheme used by Eclipse, we will look at another scheme that has been, and still is, very popular in reservoir simulators.

The IMPES method

IMPES is an acronym for **im**plicit **p**ressure, **e**xplicit **s**aturation. The method is hence an intermediate between explicit and implicit methods, and has some of the advantages of pure implicit methods but with less labour. As the acronym implies, IMPES is a numeric method where saturations are updated explicitly, while the pressure equation is solved implicitly.

First we generalize the expression Equation (13) (pg. 28) a little. Firstly, in that example we assumed one-dimensional flow and disregarded gravity. In three dimensional problems with gravity the term $\frac{\partial p}{\partial x}$ becomes $\nabla p - \gamma \nabla z$ (γ is the gravity term). This term is defined as the *fluid potential* ψ, $\nabla \psi = \nabla p - \gamma \nabla z$ (as usual with subscripts o, w, g when referring to phases).

Further, using the fluid potential, the right hand side of Equation (13) generalized to three dimensions becomes (only *x*-direction term shown here)

$$\frac{1}{\Delta x_i}\left\{\left(\frac{K_x\lambda}{\Delta x}\right)_{i+1/2}(\psi_{i+1}-\psi_i)-\left(\frac{K_x\lambda}{\Delta x}\right)_{i-1/2}(\psi_i-\psi_{i-1})\right\}\Delta y_j\Delta z_k \tag{27}$$

As a shorthand notation we define a variant of transmissibility, T_x by this expression as,

$$\Delta_x T_x \Delta_x \psi = \frac{1}{\Delta x_i}\left\{\left(\frac{K_x\lambda}{\Delta x}\right)_{i+1/2}(\psi_{i+1}-\psi_i)-\left(\frac{K_x\lambda}{\Delta x}\right)_{i-1/2}(\psi_i-\psi_{i-1})\right\}\Delta y_j\Delta z_k \tag{28}$$

(Δ with a subscript means "difference in the direction of the subscript", e.g.

$$\Delta_t f = f(t+\Delta t)-f(t) = f(t^{n+1})-f(t^n))$$

The definition (28) is for the *x*-direction differences. Similar terms for the *y*- and *z*- direction differences are obtained by cycling *x*, *y*, and *z* in the definition. The total flow is then given by,

$$\Delta T\Delta\psi = \Delta_x T_x\Delta_x\psi + \Delta_y T_y\Delta_y\psi + \Delta_z T_z\Delta_z\psi \tag{29}$$

For simplicity the phase subscripts have been omitted, these will be used when necessary. Then the black oil difference equations become,

$$\Delta T_l\Delta\psi_l + q_{l,ijk} = C_{ijk}\Delta_t(\phi S_l b_l),\ \ l=o,w \qquad \text{(water, oil)} \tag{30}$$

$$\Delta T_g\Delta\psi_g + \Delta R_s T_o\Delta\psi_o + q_{g,ijk} = C_{ijk}\Delta_t(\phi S_g b_g + \phi R_s S_o b_o) \qquad \text{(gas)} \tag{31}$$

In Equations (30), (31), $b_l = \frac{1}{B_l}$ and $C_{ijk} = \frac{V_{ijk}}{\Delta t} = \frac{\Delta x_i \Delta y_j \Delta z_k}{\Delta t}$ (for a rectangular cartesian grid).

It is easily verified that the product rule for differentiating can be applied to differences, as in

$$\Delta_t(F\cdot G) = F^{n+1}\Delta_t G + G^n\Delta_t F \tag{32}$$

$$\begin{aligned}\Delta_t(AB) &= (AB)^{n+1}-(AB)^n = A^{n+1}B^{n+1}-A^nB^n = \\ &A^{n+1}B^{n+1}-A^{n+1}B^n+A^{n+1}B^n-A^nB^n = \\ &A^{n+1}\Delta_t B + B^n\Delta_t A\end{aligned}$$

Using Equation (32) on the right hand side of Equations (30), (31) gives,

$$\Delta_t(\phi S_l b_l) = (\phi b_l)^{n+1}\Delta_t S_l + S_l^n\Delta_t(\phi b_l) \qquad l=o,w \tag{33}$$

$$\Delta_t(\phi R_s S_o b_o) = (\phi R_s b_o)^{n+1}\Delta_t S_o + S_o^n\Delta_t(\phi R_s b_o) \tag{34}$$

Now we want to eliminate $\Delta_t S_l,\ l=o,w,g$ from Equations (30), (31), using (33) and (34). We also eliminate gas saturation by $S_w + S_o + S_g = 1$

$$\frac{1}{C_{ijk}}(\Delta T_w\Delta\psi_w + q_{w,ijk}) = (\phi b_w)^{n+1}\Delta_t S_w + S_w^n\Delta_t(\phi b_w) \tag{35}$$

$$\frac{1}{C_{ijk}}(\Delta T_o\Delta\psi_o + q_{o,ijk}) = (\phi b_o)^{n+1}\Delta_t S_o + S_o^n\Delta_t(\phi b_o) \tag{36}$$

$$\begin{aligned}&\frac{1}{C_{ijk}}(\Delta T_g\Delta\psi_g + \Delta R_s T_o\Delta\psi_o + q_{g,ijk}) = \\ &(\phi b_g)^{n+1}(-\Delta_t S_w - \Delta_t S_o) + (1-S_o^n-S_w^n)\Delta_t(\phi b_g) + (\phi b_o R_s)^{n+1}\Delta_t S_o + S_o^n\Delta_t(\phi b_o R_s)\end{aligned} \tag{37}$$

This is a system of three equations in the two unknowns $\Delta_t S_o$ and $\Delta_t S_w$ that can be solved by a linear combination of Equations (35) – (37). One such combination is,

$$\frac{1}{b_w^{n+1}}[Eq.(35)]+\left\{\frac{1}{b_o^{n+1}}-\frac{R_s}{b_g^{n+1}}\right\}[Eq.(36)]+\frac{1}{b_g^{n+1}}[Eq.(37)].$$

Performing this summation and reorganizing, the right hand side of the sum becomes,

$$\Delta_t\phi+\phi^n\left\{\frac{S_w^n}{b_w^{n+1}}\Delta_t b_w+\frac{S_o^n}{b_o^{n+1}}\Delta_t b_o+\frac{S_g^n}{b_g^{n+1}}\Delta_t b_g+\frac{S_o^n b_o^n}{b_g^{n+1}}\Delta_t R_s\right\} \tag{38}$$

In this expression, saturations only occur at time step *n*, i.e. explicitly.

By the chain rule, $\Delta_t b_l=\frac{\partial b_l}{\partial p_l}\Delta_t p_l$.

Since capillary pressure is a function of saturation, and the saturations are treated explicitly (i.e. the saturation is the same over a time step), capillary pressure will not change during a time step, so that $\Delta_t p_w=\Delta_t p_o=\Delta_t p_g=\Delta_t p$, the last equal sign being a definition.

The three first terms in expression (38) are very similar, so define a value C_t by,

$$C_t=\left\{\sum_{l=o,w,g}\frac{S_l^n}{b_l^{n+1}}\frac{\partial b_l}{\partial p}\right\}+\frac{S_o^n b_o^n}{b_g^{n+1}}\frac{\partial R_s}{\partial p} \tag{39}$$

Using (39), Equation (38) can be rewritten as,

$$\Delta_t\phi+\phi^n C_t\Delta_t p \tag{40}$$

Equation (40) has the form of "change of porosity due to compressibility". Now introduce rock compression C_r by $\Delta_t\phi=C_r\phi^n\Delta_t p$ (41)

Then Equation (40) becomes $(C_r+C_t)\phi^n\Delta_t p$

Developing the left hand side of the sum that led to Equation (38) and using (41) we finally arrive at the *pressure equation*,

$$\begin{aligned}B_w^{n+1}\Delta T_w\Delta\psi_w+(B_o^{n+1}-B_g^{n+1}R_s^{n+1})\Delta T_o\Delta\psi_o+B_g^{n+1}\left(\Delta T_g\Delta\psi_g+\Delta R_s T_w\Delta\psi_o\right)+\\ +B_w^{n+1}q_w+(B_o^{n+1}-B_g^{n+1}R_s^{n+1})q_o+B_g^{n+1}q_g\\ =\frac{V_{ijk}}{\Delta t}(C_r+C_t)\phi^n\Delta_t p\end{aligned} \tag{42}$$

In Equation (42), saturations and all variables that are saturation-dependent have been found at time step *n*. Equation (42) is then solved implicitly for pressure (including pressure-dependent variables).

The explicit nature of the saturation computations will normally dictate relatively small time steps, especially when saturation changes rapidly.

Mass Balance

In production reservoir simulation, conservation of mass is a must. It is therefore standard procedure to use mass balance as a convergence control criterion. E.g. for the solution procedure above, one would not eliminate gas saturation as we did, but compute all saturations independently. Then the consistency requirement $S_w+S_o+S_g=1$ would be used as a control criterion for mass conservation. If the sum of saturations was not sufficiently close to unity, the procedure would be iterated until convergence was satisfactory.

Solution of Non-linear Equations – the Newton-Raphson method

As noted, the black oil equations are non-linear, so standard techniques for solving linear systems are not applicable to the difference form of the equations. We will first look at the Newton-Raphson method, a popular and powerful technique for solving non-linear problems, and as an introduction describe this method for scalar problems.

We want to solve the non-linear equation $f(x) = 0$, where $y = f(x)$ is some function, e.g. the one shown in Figure 31.

The method is best described as an algorithm,

1. Choose some arbitrary starting point x^0. Find $y^0 = f(x^0)$.
2. Find the tangent to $y = f(x)$ in the point (x^0, y^0).
3. Find the intersection x^1 between the tangent from (2) and the x-axis.
4. If $f(x^1) = 0$, or sufficiently close to 0, x^1 is the sought solution.
   Else continue procedure as in (2), (3), (4) until convergence, defining points $x^2, x^3, x^4,...$ approaching the solution.

In Figure 31 we can see how the points $x^0, x^1, x^2,...$ approach the zero of $y = f(x)$,and can easily convince ourselves that this procedure will converge to the solution.

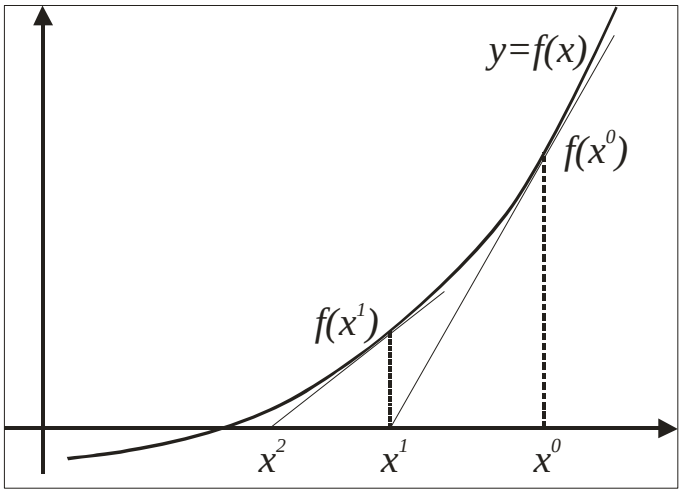


Figure 31. Solving a nonlinear problem with the Newton-Raphson method

The tangent equation is

$$y = f(x^0) + f'(x^0) \cdot (x - x^0)$$

The point x^1 is found by setting $y = 0$ in this equation, so that

$$x^1 = x^0 - \frac{f(x^0)}{f'(x^0)} \qquad (43)$$

(We must require the derivative $f'(x^0)$ nonzero, but this is only logical since a zero derivative means the tangent is horizontal, and hence cannot intersect the x-axis.)
The "stop-criterion" is $|f(x^i)| < \varepsilon$, where ε is some predefined error tolerance.

The algorithm as a computer code would look like,

```
x = x0;
do
   x = x – f(x) / diff_f(x);
while(abs(f(x)) > epsilon);
```

At each iteration, we can define the difference between "this" and "previous" x as Δx^k. E.g. at the first iteration, $\Delta x^0 = x^1 - x^0$. Then Eq. (43) can be written

$$f'(x^k)\Delta x^k + f(x^k) = 0, \tag{44}$$

which is a more suitable form to generalize to several dimensions

Newton-Raphson for systems

The method described above can be generalized to systems of non-linear equations. The notation and implementation become more complex, but the underlying idea is very much the same.

The nonlinear problem is,

$$\begin{aligned} &F_1(\mathbf{x}) = 0 \\ &F_2(\mathbf{x}) = 0 \\ &\vdots \\ &F_n(\mathbf{x}) = 0 \end{aligned} \tag{45}$$

where $\mathbf{x} = x_1,...,x_n$, and the initial guess is $\mathbf{x}^0$.

The system equivalent of the derivative of f is the total differential of the F_i. So assuming the $\mathbf{x}$-value after k iterations is $\mathbf{x}^k$, the equivalent of the tangent in Equation (44) is,

$$\frac{d\mathbf{F}}{d\mathbf{x}}(\mathbf{x}^k)\cdot\Delta\mathbf{x}^k + \mathbf{F}(\mathbf{x}^k) = 0, \quad \text{or expanded,}$$

$$\begin{aligned} &\frac{\partial F_1}{\partial x_1}(\mathbf{x}^k)\Delta x_1^k + \frac{\partial F_1}{\partial x_2}(\mathbf{x}^k)\Delta x_2^k + \cdots + \frac{\partial F_1}{\partial x_n}(\mathbf{x}^k)\Delta x_n^k + F_1(\mathbf{x}^k) = 0 \\ &\frac{\partial F_2}{\partial x_1}(\mathbf{x}^k)\Delta x_1^k + \frac{\partial F_2}{\partial x_2}(\mathbf{x}^k)\Delta x_2^k + \cdots + \frac{\partial F_2}{\partial x_n}(\mathbf{x}^k)\Delta x_n^k + F_2(\mathbf{x}^k) = 0 \\ &\vdots \\ &\frac{\partial F_n}{\partial x_1}(\mathbf{x}^k)\Delta x_1^k + \frac{\partial F_n}{\partial x_2}(\mathbf{x}^k)\Delta x_2^k + \cdots + \frac{\partial F_n}{\partial x_n}(\mathbf{x}^k)\Delta x_n^k + F_n(\mathbf{x}^k) = 0 \end{aligned} \tag{46}$$

The equivalent of Equation (44) is to solve Equation (46) with respect to the $\Delta\mathbf{x}^k = (\Delta x_1^k,...,\Delta x_n^k)$. Then the solution vector can be updated by $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}^k$, and the procedure continues until $\left\|\mathbf{F}(\mathbf{x}^k)\right\| < \varepsilon$, where $\|\cdot\|$ is some norm, e.g. $\|\mathbf{F}\| = \max|F_i|$, $\|\mathbf{F}\| = \sum|F_i|$, or $\|\mathbf{F}\| = \left\{\sum (F_i)^2\right\}^{1/2}$.

Unfortunately, solving Equation (46) with respect to the $\Delta\mathbf{x}^k$ generally requires a lot more work than solving Equation (44). Equation (46) is a linear system of equations, so the multitude of methods developed to solve such systems are available.

Application to the black oil equations

To get the difference form of the black oil equations as in (45), we rewrite Equations (30), (31) slightly, Define functions F_1, F_2, F_3 by,

$$F_1 = \Delta T_w \Delta\psi_w + q_{w,ijk} - C_{ijk}\Delta_t(\phi S_w b_w) \tag{47}$$

$$F_2 = \Delta T_o \Delta\psi_o + q_{o,ijk} - C_{ijk}\Delta_t(\phi S_o b_o) \tag{48}$$

$$F_3 = \Delta T_g \Delta\psi_g + \Delta R_s T_o \Delta\psi_o + q_{g,ijk} - C_{ijk}\Delta_t(\phi S_g b_g + \phi R_s S_o b_o) \tag{49}$$

Then Equations (30), (31) are equivalently formulated as

$$F_i = 0,\ i=1,2,3 \tag{50}$$

(with the F_i defined by Equations (47) – (49)).

We take phase pressures and fluid saturations for the three phases as primary unknowns, so that our independent vector variable in a three phase problem is, $\mathbf{v} = (p_w, p_o, p_g, S_w, S_o, S_g)$.
(In practice we could use $\mathbf{v} = (p_w, S_w, S_g)$, and determine p_o and p_g from capillary pressure, and S_o from $S_w + S_o + S_g = 1$.)
The first equation in the system (46) then becomes,

$$\begin{aligned}
&\frac{\partial F_1}{\partial p_w}\Delta p_w + \frac{\partial F_1}{\partial p_o}\Delta p_o + \frac{\partial F_1}{\partial p_g}\Delta p_g + \frac{\partial F_1}{\partial S_w}\Delta S_w + \frac{\partial F_1}{\partial S_o}\Delta S_o + \frac{\partial F_1}{\partial S_g}\Delta S_g + F_1 = 0 \\
&\Leftrightarrow \left(\frac{\partial F_1}{\partial p_w}, \frac{\partial F_1}{\partial p_o}, \frac{\partial F_1}{\partial p_g}, \frac{\partial F_1}{\partial S_w}, \frac{\partial F_1}{\partial S_o}, \frac{\partial F_1}{\partial S_g}\right) \cdot \Delta\mathbf{v} + F_1 = 0 \qquad (51) \\
&\Leftrightarrow \nabla_{\mathbf{v}} F_1 \cdot \Delta\mathbf{v} + F_1 = 0
\end{aligned}$$

Assume now that we are iterating on $\mathbf{v}$, and the solution at the current iteration is $\mathbf{v}^k$. We use the shorthand notation $c \in CC$ to mean "cell c is in the computational cube", i.e. the 7-point stencil centred at c.
Then the "next" Newton-Raphson iteration towards the solution to the problem (50) is,

$$\sum_{c \in CC} (\nabla_{\mathbf{v}} F_m \cdot \Delta\mathbf{v})_c^k + F_m^k = 0, \quad m = 1,2,3 \qquad (52)$$

This is a system of linear equations we can solve, e.g. with the techniques described in the next chapter.

Overview of equation solving (time step advancement) in Eclipse

The complete procedure for advancing the solution one time step by the Newton-Raphson method is then,

1. Formulate the problem (50), with current values of the **v**-vector as start-values
2. Solve problem by Newton-Raphson
    a. Formulate "tangent equation", Equation (52)
    b. Solve Equation (52) for $\Delta\mathbf{v}^k$.
3. Check error criterion $\left\|\mathbf{F}(\mathbf{v}^k)\right\| < \varepsilon$ (53)
4. If solution has converged, OK, else update **v**-vector and repeat from 1).

The solution is hence found by iteration according to the Newton-Raphson scheme. These iterations are called *non-linear* iterations, *Newton* iterations (in Eclipse terminology) or *outer* iterations. Note that when the error criterion (53) is fulfilled, the solution procedure *has* converged, and the correct solution has been found. We stress this before introducing a new player in the game.

Point 2b), Solve Equation (52) is *not* a trivial task. Typically, about 80% of the total processor time used when doing a simulation is spent on this problem. Details will come later, but for now we just state that the only viable way to solve the linear problem is by iteration. So in *each* Newton iteration, the linear problem is solved by doing a number of *linear* iterations, also called *inner* iterations. The linear problem has converged when a similar error criterion to (53) is fulfilled. But there is a great difference. In this case only a sub-problem to the actual task has been solved, and convergence of the linear problem only means we are one step closer to the final solution. In that view, it is not always critical if the linear equations are not completely converged, as long as Equation (53) is fulfilled in the end. We will return to this discussion in the section on convergence and time step control.

The fact that we operate with two different kinds of iterations can be confusing. The algorithm below is an attempt to clarify,

```
// Pseudo-algorithm for Newton-Raphson for systems

initialise(v);
do {
   //Non-linear iterations
   formulate_non_linear_system(v);
   make_total_differential(v);
   do {
      // Linear iterations:
      update_linear_system_variables(v);
   }
   while((linear_system_has_not_converged(v));
   update_non_linear_system_after_linear_convergence(v);
}
while((non_linear_system_has_not_converged(v));
```

17. Iteration methods for linear systems

As noted above, each non-linear iteration requires the solving of a linear system of equations, and a substantial part of the total computing time is spent in solving this system. Through the years a great deal of effort has been put into development of methods that solve linear systems more efficiently, and it is fair to say that a major reason Eclipse quickly became so popular was its linear solver, which made it possible to do real-size three dimensional reservoir simulations with implicit solvers that permitted large time steps. In this chapter we will describe the technique (developed by Appleyard, Cheshire and others in the 1980s), after first looking at fundamental methods that preceded it.

Notation:
Boldface capitals will denote (*n x n*) matrices.
If **A** is a matrix, a_{ij} will be the element of **A** in column *i*, row *j*.
Boldface small letters are used for vectors.

The problem to solve is always

$$\mathbf{A}\cdot\mathbf{x}=\mathbf{b} \tag{54}$$

where **A** is a sparse *n x n* matrix, and the unknown vector is **x**, so that we seek,

$$\mathbf{x}=\mathbf{A}^{-1}\cdot\mathbf{b} \tag{55}$$

If **A** is tridiagonal, the problem (55) can easily be solved exactly by the Thomas algorithm. Apart from that special case, iteration techniques are the only applicable candidates for solving (55), and are what we will concentrate on.

Equation (54) expanded becomes,

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,\quad i=1,...,n\text{; or solved for } x_i\text{: } x_i=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j-\sum_{j=i+1}^{n}a_{ij}x_j\right) \tag{56}$$

Direct, simple approach

Update all x_i by previous iteration values. Working from row 1 and downwards, we get (with superscript denoting the iteration counter)
Start with some initial value ("guess") for **x,** called $\mathbf{x}^0$. Then for *k=1,2,3,...*

$$x_i^{k+1}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^k-\sum_{j=i+1}^{n}a_{ij}x_j^k\right) \tag{57}$$

Continue until convergence, i.e. the system has been solved to required precision.
The method (57) has very poor convergence (and is probably not used at all).

The Gauss-Seidel method

A simple improvement to scheme (57) is to use the updated *x*-variables as soon as they become available. This approach is actually easier to implement.
The difference from Equation (57) is small but important,

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right) \qquad (58)$$

This scheme, called Gauss-Seidel, has much better convergence properties than (57).

Accelerators – the point SOR method

Equation (58) still has much room for improvement. Imagine that the update performed on a single iteration by Equation (58) changes the solution vector in the correct manner ("pushes it in the right direction"), but e.g. not enough. So if we could "push it a little further" the system would obviously (?) converge in fewer iterations. Such arguments are the foundation for many acceleration techniques. We first look at one called point SOR, where SOR means "successive over-relaxation".

The scheme is,

$$y_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right) \qquad (59)$$
$$x_i^{k+1} = x_i^k + \omega(y_i^{k+1} - x_i^k)$$

So first a temporary *y*-variable is found exactly as in (58), and then the *x*-variable is updated by "pushing" the update a little further than the Gauss-Seidel scheme, by a *relaxation parameter* ω. Note that if $\omega = 1$, point SOR is identical to Gauss-Seidel. Whether point SOR really is an improvement over Gauss-Seidel depends entirely on the relaxation parameter. No general rules for defining ω can be given, except that the optimal ω is between 1 and 2. With a lucky or good choice of ω the system can converge very quickly, while other values can actually give a performance degradation.
In conclusion, point SOR can be very efficient if a good relaxation parameter is found, but the method is problem dependent, and not very robust. (I.e. it doesn't solve all systems).

The point SOR method has seen many improvements, e.g. block SOR and different versions of line SOR. Mostly the methods are efficient, especially those which are tailored to a specific matrix structure, but the efficiency is always dependent on the relaxation parameter, which may be a disadvantage. We will not discuss the other SOR methods here, but go directly to the relaxation method used in Eclipse.

Conjugate Gradients – ORTHOMIN

Motivation
In the point SOR it was loosely referenced to "pushed in the right direction". If the problem is one, two, or three-dimensional we should have no difficulty in visualising the direction from the current solution vector to the final one. But the problems we encounter are *n*-dimensional, with *n* normally very large. So which direction to move the current vector in, and criteria for if we are moving in a sensible direction or not can be hard to formulate. These questions are directly addressed in the *conjugate gradient* method, which combined with ORTHOMIN (orthogonal minimisation) is a support pillar in the linear solver in Eclipse.

The orthomin procedure defines a sequence of "search vectors", $\mathbf{q}^0$, $\mathbf{q}^1$, $\mathbf{q}^2$,... Each $\mathbf{q}$ defines a "search direction" in the *n*-dimensional solution space, such that

- When a new $\mathbf{q}$-vector is constructed, it should be orthogonal to all previously used $\mathbf{q}$-vectors.
- Each $\mathbf{q}^k$ should optimally be the best search direction available at the present stage of the process

Two $\mathbf{q}$-vectors are orthogonal if $\mathbf{A}\mathbf{q}^k \cdot \mathbf{A}\mathbf{q}^j = 0$ when $k \neq j$.

As usual we start with an initial guess $\mathbf{x}^0$. Then at each iteration, $\mathbf{x}$ is updated by,

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \omega^k \mathbf{q}^k \tag{60}$$

(Note the similarity to point SOR, Equation (59).)

The relaxation parameter ω^k is tried optimized, but note that in contrast to the SOR methods, ω^k here is updated at each iteration.

The residual $\mathbf{r}^{k+1}$ is a measure for the current error, and is zero if $\mathbf{x}^{k+1}$ is an exact solution to the system:

$$\mathbf{r}^{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^k - \mathbf{A}\omega^k \mathbf{q}^k = \mathbf{r}^k - \omega^k \mathbf{A}\mathbf{q}^k \tag{61}$$

We now determine ω^k by finding $\min_{\omega} \left\| \mathbf{r}^{k+1} \right\|$:

$$\left\| \mathbf{r}^{k+1} \right\|^2 = \mathbf{r}^{k+1} \cdot \mathbf{r}^{k+1} = \mathbf{r}^k \cdot \mathbf{r}^k - 2\omega^k \mathbf{r}^k \cdot \mathbf{A}\mathbf{q}^k + (\omega^k)^2 \mathbf{A}\mathbf{q}^k \cdot \mathbf{A}\mathbf{q}^k \tag{62}$$

The minimum value is found where the derivative is zero:

$$\begin{aligned} &\min_{\omega} \left\| \mathbf{r}^{k+1} \right\|^2 : \frac{\partial}{\partial \omega^k} \left\| \mathbf{r}^{k+1} \right\|^2 = 0 \\ &\Rightarrow -2\mathbf{r}^k \cdot \mathbf{A}\mathbf{q}^k + 2\omega^k \mathbf{A}\mathbf{q}^k \cdot \mathbf{A}\mathbf{q}^k = 0 \\ &\Rightarrow \omega^k = \frac{\mathbf{r}^k \cdot \mathbf{A}\mathbf{q}^k}{\mathbf{A}\mathbf{q}^k \cdot \mathbf{A}\mathbf{q}^k} \end{aligned} \tag{63}$$

We see that the solution found in (63) is the only solution to the minimisation problem. Moreover, it must be a minimum since $\left\| \mathbf{r}^{k+1} \right\|^2 \to \infty$ when $\omega \to \pm\infty$.

Construction of the q-vectors

At iteration step *k*, the next $\mathbf{q}$, $\mathbf{q}^{k+1}$, is constructed from the most recent residual and a linear combination of all previous $\mathbf{q}$-vectors,

$$\mathbf{q}^{k+1} = \mathbf{r}^{k+1} + \sum_{j=0}^{k} \alpha_j^k \mathbf{q}^j \tag{64}$$

where α_j^k are parameters to be determined.

Since $\mathbf{A}\mathbf{q}^{k+1} \cdot \mathbf{A}\mathbf{q}^j = 0$ for all $j = 0,1,\ldots,k$, we get k equations for determining the unknown α_j^k:

$$0 = \mathbf{A}\mathbf{q}^{k+1} \cdot \mathbf{A}\mathbf{q}^i = \mathbf{A}\mathbf{r}^{k+1} \cdot \mathbf{A}\mathbf{q}^i + \sum_{j=0}^{k} \alpha_j^k \mathbf{A}\mathbf{q}^j \cdot \mathbf{A}\mathbf{q}^i \tag{65}$$

The term $\mathbf{A}\mathbf{q}^j \cdot \mathbf{A}\mathbf{q}^i = 0$ if $i \neq j$, so all terms in the sum vanish, except when $i = j$:

$$\begin{aligned} &0 = \mathbf{A}\mathbf{r}^{k+1} \cdot \mathbf{A}\mathbf{q}^i + \alpha_i^k \mathbf{A}\mathbf{q}^i \cdot \mathbf{A}\mathbf{q}^i \\ &\Rightarrow \alpha_i^k = -\frac{\mathbf{A}\mathbf{r}^{k+1} \cdot \mathbf{A}\mathbf{q}^i}{\mathbf{A}\mathbf{q}^i \cdot \mathbf{A}\mathbf{q}^i} \end{aligned} \tag{66}$$

The orthomin algorithm has then been established:

Set initial guess $\mathbf{x}^0$, and $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$, $\mathbf{q}^0 = \mathbf{r}$. Set iteration counter $k = 0$.

```
while( not converged) {
        Set ω^k by Equation (63)
        Update x by  x = x + ω^k q^k
        Update r by r = r - ω^k A q^k
        If not converged {
                Update α by Equation (66)
                Update q by Equation (64)
        }
        Increase k
}
```

The orthomin procedure can be shown to converge in maximum n iterations (where n was the matrix dimension). This is however a theoretical limit, and much too large to be of interest for typical problems. Experience has shown that the method converges in maximum a few tens of iterations. As a rule of thumb, if orthomin hasn't converged in about 50 iterations, it is recommended to reduce the time step. Also, for orthomin to function according to the theory it is necessary to store *all* previous **q**-vectors. Since each vector is n-dimensional (i.e. large), there will normally be a memory limitation to how many **q**-vectors that can be stored. If this maximum has been reached, the procedure will discard vectors from the beginning to create necessary storage. But then the search direction that was discarded will most likely be the optimal search direction available. With a (small) finite number of **q**-vectors stored there is hence a risk that the procedure will be running in loops.
But we shouldn't stress these problems too much – by experience the method generally has excellent performance.

Preconditioning

Another or additional method to accelerate solving of linear systems is the so-called *preconditioning*. For Eclipse (and other simulators), ORTHOMIN alone is not sufficient to solve the linear equations with needed efficiency – preconditioning is also needed, and for Eclipse is at least as important as the iteration method to acheive speed-up.

Basically, the argument is, can the original problem be replaced by an equivalent problem that is easier to solve? Problems where **A** is a diagonal matrix are trivially solved, and the objective of preconditioning is to change the system so that the coefficient matrix becomes "close to diagonal".

Our problem was $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. Multiplying this equation by the identity matrix $\mathbf{I} = \mathbf{B}^{-1}\mathbf{B}$ changes nothing:

$$\mathbf{AB}^{-1}\mathbf{Bx} = \mathbf{b} \Rightarrow (\mathbf{AB}^{-1})(\mathbf{Bx}) = \mathbf{b} \qquad (67)$$

Bx is a vector of similar complexity as **x**, so almost no extra work is required to compute **Bx**.
If $\mathbf{AB}^{-1} \approx \mathbf{I}$ the system (67) will be easy to solve (i.e. it converges very quickly if an iteration method is used). Obviously if this is going to work, the effort to define and invert **B** must be substantially smaller than solving the original problem, hence ideally,

- $\mathbf{B} \approx \mathbf{A}$
- **B** is easily inverted.

But if $\mathbf{B} \approx \mathbf{A}$, inverting **B** is as difficult as inverting **A**, and we are no better off. So the challenge is to construct an effective preconditioner **B** without too much extra work.

Before we describe how this is done in Eclipse we will look at how preconditioning affects orthomin.

Preconditioning and Orthomin

If **A** has been preconditioned, the equations in the ORTHOMIN description are changed by replacing **A** with $\mathbf{AB}^{-1}$ and **x** with **Bx**. Then Equations (60) – (66) become,

$$\mathbf{Bx}^{k+1} = \mathbf{Bx}^{k} + \omega^{k}\mathbf{q}^{k} \Rightarrow \mathbf{x}^{k+1} = \mathbf{x}^{k} + \omega^{k}\mathbf{B}^{-1}\mathbf{q}^{k} \tag{68}$$

$$\mathbf{r}^{k+1} = \mathbf{r}^{k} - \omega^{k}\mathbf{AB}^{-1}\mathbf{q}^{k} \tag{69}$$

$$\omega^{k} = \frac{\mathbf{r}^{k} \cdot \mathbf{AB}^{-1}\mathbf{q}^{k}}{(\mathbf{AB}^{-1}\mathbf{q}^{k}) \cdot (\mathbf{AB}^{-1}\mathbf{q}^{k})} \tag{70}$$

$$\alpha_{i}^{k} = -\frac{(\mathbf{AB}^{-1}\mathbf{r}^{k+1}) \cdot (\mathbf{AB}^{-1}\mathbf{q}^{i})}{(\mathbf{AB}^{-1}\mathbf{q}^{i}) \cdot (\mathbf{AB}^{-1}\mathbf{q}^{i})} \tag{71}$$

Note that **q** always appears as $\mathbf{B}^{-1}\mathbf{q}$. We can therefore redefine the **q**-vector to $\mathbf{B}^{-1}\mathbf{q}$, whereby the Equations (68) – (71) become almost identical to their origins.

The pseudo-algorithm for orthomin with preconditioning is then,

Set initial vector $\mathbf{x}^0$. Define $\mathbf{r}^0 = \mathbf{b} - \mathbf{Ax}^0$ and $\mathbf{q}^0 = \mathbf{B}^{-1}\mathbf{r}^0$.
Set iteration counter $k = 0$.
while(not converged or k > max permitted number of iterations) {

$$\omega^{k} = \frac{\mathbf{r}^{k} \cdot \mathbf{Aq}^{k}}{\mathbf{Aq}^{k} \cdot \mathbf{Aq}^{k}}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} + \omega^{k}\mathbf{q}^{k}$$

$$\mathbf{r}^{k+1} = \mathbf{r}^{k} - \omega^{k}\mathbf{Aq}^{k}$$

for(i = 0,...,k) {

$$\alpha_{i}^{k} = -\frac{(\mathbf{AB}^{-1}\mathbf{r}^{k+1}) \cdot (\mathbf{Aq}^{i})}{(\mathbf{Aq}^{i}) \cdot (\mathbf{Aq}^{i})}$$

}

$$\mathbf{q}^{k+1} = \mathbf{B}^{-1}\mathbf{r}^{k+1} + \sum_{i=0}^{k} \alpha_{i}^{k}\mathbf{q}^{i}$$

}

(We have omitted the non-essential required modification if only a limited number of **q**-vectors can be stored.)

The question of how to determine a good or optimal preconditioner **B** is still open. Eclipse does this by a technique called *Nested Factorisation*, which is the last element of the Eclipse linear solver procedure.

Determining a preconditioner – Nested Factorisation

Recall the structure for the 7-point stencil scheme, which was a 7-diagonal matrix. We rewrite the example of Figure 30 with a slightly different notation,

D	U1			U2								U3											
L1	D	U1			U2								U3										
	L1	D	U1			U2								U3									
		L1	D				U2								U3								
L2				D	U1			U2								U3							
	L2			L1	D	U1			U2								U3						
		L2			L1	D	U1			U2								U3					
			L2			L1	D				U2								U3				
				L2				D	U1											U3			
					L2			L1	D	U1											U3		
						L2			L1	D	U1											U3	
							L2			L1	D												U3
L3												D	U1			U2							
	L3											L1	D	U1			U2						
		L3											L1	D	U1			U2					
			L3											L1	D				U2				
				L3								L2				D	U1			U2			
					L3								L2			L1	D	U1			U2		
						L3								L2			L1	D	U1			U2	
							L3								L2			L1	D				U2
								L3								L2				D	U1		
									L3								L2			L1	D	U1	
										L3								L2			L1	D	U1
											L3								L2			L1	D

Figure 32. Labelling the seven diagonals in a 7-diagonal matrix

The nonzero entries in the matrix have been labelled D for the main diagonal, and L1, L2, L3 for the lower diagonals, U1, U2, U3 for the upper diagonals. The way the matrix was set up, the D, L1, and U1 correspond to communication between the centre cell *(i, j, k)* and the neighbours in the x-direction *(i±1, j, k).* This is a result of the natural ordering, and as noted earlier, the structure of the matrix is a result of the ordering used.

In the natural, or "book" ordering, by Eclipse terminology, the *x*-direction would be denoted as "*direction 1*", the *y*-direction as "*direction 2*", and the *z*-direction as "*direction 3*". I.e. "*direction 1*" is the one where the index changes fastest, "*direction 2*" is used for the second fastest, and "*direction 3*" for the slowest.

All permutations of "*direction 1*" – "*direction 2*" – "*direction 3*" will give the 7-diagonal matrix structure seen in Figure 32, but the communication direction will be different between the D, L1-3, and U1-3. E.g., if the *z*-direction is chosen as "*direction 1*", L1, D, and U1 will correspond to communication between *(i, j, k)* and *(i, j, k±1)*.
So irrespective of the ordering scheme, the coefficient matrix will be (using **D** for the matrix comprised of all the D-elements etc.),

A = D + L1 + U1 + L2 + U2 + L3 + U3

(Reminder: All the "elements" in Figure 32 are actually 2x2 matrices for two-phase flow, 3x3 for three-phase.)

Without proof we state that in general *the most efficient ordering scheme with respect to solving the system is such that "direction 1" is the direction of highest average transmissibility, "direction 2" next highest, and "direction 3" lowest transmissibility.*

This criterion for setting the ordering scheme is used as default by Eclipse, which means that in most simulations, "*direction 1*" will be the *z*-direction (*z*-permeability is normally lowest, but the average cell thicknesses are typically much smaller than *X,Y* lengths.)

Users can change the ordering by the SOLVDIRS keyword. (Generally not recommended.)

The technique called *nested factorisation* is a step-by-step procedure to build the preconditioning matrix **B**. The method may look complex and laborious, but keep in mind that it is an *iterative* construction, such that the right hand sides of the following equations are to be understood as "the values known from previous iteration".

B is built iteratively by a sequence of factorisations,

$$\mathbf{B} = (\mathbf{P} + \mathbf{L3})\mathbf{P}^{-1}(\mathbf{P} + \mathbf{U3}) \quad (72)$$
$$\mathbf{P} = (\mathbf{T} + \mathbf{L2})\mathbf{T}^{-1}(\mathbf{T} + \mathbf{U2}) \quad (73)$$
$$\mathbf{T} = (\mathbf{G} + \mathbf{L1})\mathbf{G}^{-1}(\mathbf{G} + \mathbf{U1}) \quad (74)$$

G is a diagonal matrix, defined by,

$$\mathbf{G} = \mathbf{D} - \mathbf{L1}\mathbf{G}^{-1}\mathbf{U1} - \text{colsum}(\mathbf{L2}\mathbf{T}^{-1}\mathbf{U2}) - \text{colsum}(\mathbf{L3}\mathbf{P}^{-1}\mathbf{U3}). \quad (75)$$

The procedure is initialised by constructing **G** cell by cell using (75), and then the updates (72) – (75) are performed iteratively until convergence (**B** changes less than tolerance between two iterations).

"colsum" means the column sum of the matrix.
The way **G** is defined implies that colsum(**A**) = colsum(**B**), which ensures no mass balance errors.

18. Convergence Control II – TUNING parameters

Summing up the discussion of the previous chapters,

At time step t^n, we have a solution $\mathbf{v} = (p_w, p_o, p_g, S_w, S_o, S_g)$.
To advance the solution to time t^{n+1}, with $\Delta t = t^{n+1} - t^n$, we must solve a nonlinear problem. The solution procedure was,

Define nonlinear problem $\mathbf{F}(\mathbf{v}, t^{n+1}) = \mathbf{0}$.

1. Solve nonlinear problem by Newton-Raphson, using $\mathbf{v}^0 = \mathbf{v}(t^n)$ as start value.
    a. Construct "tangent" – total differential of non-linear problem.
       $\Rightarrow$ linear system $\mathbf{Ax} = \mathbf{b}$.
    b. Solve linear system
        i. Precondition matrix
        ii. orthomin – iterations
        iii. End iterations when system has converged
2. Continue iterations until system has converged.

In this solution procedure, there are many parameters which have influence on how the iteration procedure performs.

1. Δt.
   All iteration processes perform best if the starting value is close to the sought solution. E.g. returning to the first non-linear problem we solved with Newton-Raphson (Figure 31), it is easy to imagine that if $f(x)$ has many zeros, and we start far from the sought one, we are going to get the wrong result. If Δt is small, the solution will probably not change that much over a time step, so that the iteration starting value $\mathbf{v}(t^n)$ is close to the sought solution $\mathbf{v}(t^{n+1})$. In general we want Δt as large as possible, but it is always an option to reduce the time step size if convergence problems do occur.
2. Iteration control on outer (Newton) iterations.
    a. The error tolerance, when has the process converged?
    b. The maximum number of iterations.
       Normally the process will converge after not too many iterations.
       But what if it doesn't? It may happen that we are facing a problem that will never converge. We can't allow the computer to continue iterating till the end of history, so at some time we must say "enough is enough".
    c. What to do if the problem for some reason didn't converge?

3. Iteration control on inner (linear) iterations
    a. Error tolerance
    b. Maximum number of linear iterations
    c. How many search directions (**q**-vectors) to keep in memory?
       In principle all **q**-vectors should be kept for good performance, but in practice available computer memory doesn't permit this.
    d. What to do if linear system does not converge.

Time step control was discussed in section 8.3, and some more follows below.

The other parameters can all be controlled by the user, and although Eclipse has a good automatic time step and convergence control, for large problems user action will often be required to achieve optimal performance.

In general it is *not* recommended to change the default values for error tolerances. These are defined in the second record of the TUNING keyword, which as a rule of thumb should never be altered. (No rule without exceptions of course.)

The third record of the TUNING keyword has the syntax (using Eclipse terminology),

NEWTMX NEWTMN LITMAX LITMIN MXWSIT ...

NEWTMX	Maximum number of Newton iterations in a time step (default 12)
NEWTMN	Minimum number of Newton iterations in a time step (default 1)
LITMAX	Maximum number of linear iterations in each Newton iteration (default 25)
LITMIN	Minimum number of linear iterations in each Newton iteration (default 1)
MXWSIT	Maximum number of iterations within well flow calculation (default 8)

The parameter controlling the number of **q**-vectors to store is not defined in TUNING. Since this parameter directly affects the amount of computer memory to reserve, it must logically be defined in the RUNSPEC section, with keyword NSTACK:

Example

```
NSTACK
   16   /
```

sets the number of search directions to keep to 16. (The reason for the name is that the memory area where the vectors are stored is organized as what in computer terminology is called a stack.)
The default value of NSTACK is 10.

Obviously, the way LITMAX and NSTACK are related, defining an NSTACK larger than LITMAX is going to be a waste of memory space, since the total number of **q**-vectors that are defined throughout the process can't be larger than the number of linear iterations.

I.e.: Always use NSTACK ≤ LITMAX.

The default values for time step and iteration control work fine in many simulations, especially for small or medium sized problems. Three phase simulations are more likely to create convergence problems than one- or two-phase problems. How to improve performance in simulations is often a result of trial and error, and is definitely assisted by experience with the simulator.

It should be mentioned that Eclipse will often try to advice on how to improve convergence when problems occur. But these advises are not based on some artificial intelligence analysis of the problem, so should be read with some scepticism. E.g. if LITMAX is set to 100, and Eclipse advises you to "increase LITMAX", the advice can safely be disregarded.

Relative roles of time step, LITMAX and NEWTMX

To move the simulation from time *t1* to *t2*, where we have experienced convergence problems, all three parameters Δt, LITMAX and NEWTMX can contribute to improved performance, so which buttons to turn? Eclipse will always attempt to use as large time steps as possible, but clearly

simulating from *t1* to *t2* in one step, with 100 Newton iterations, each using 50 linear iterations, will take longer time than dividing the time step in two, if the iterations at each time step are reduced to something like 70 Newton iterations with 30 linear iterations at each step. (First alternative uses a total of 100 * 50 = 5000 iterations, second alternative uses 2 * 70 * 30 = 4200 iterations).

Typical values for actually used number of iterations are in the order of 10-20 linear iterations in each Newton iteration, and rarely more than 10 Newton iterations at each time step. If the simulation persistently requires (substantially) more than this at every time step, it is probably wise to reduce the maximum permitted time step.

Eclipse will report any convergence irregularities in the PRT-file, and to the output window (or log-file). In addition many convergence related parameters can be output as summary data (chapter 11). Linear convergence failure will normally not be dramatic, while non-linear convergence failure is. This can easily be seen if we go back to the original Newton-Raphson problem, Figure 31. The objective is to find the intersection between *f(x)* and the *x*-axis. The linear iterations are used to solve each linear problem, corresponding to finding the intersection between a tangent and the *x*-axis. Failure in this task means that one or more of the x^i on the *x*-axis will not be in its correct position. But it shouldn't be hard to imagine that the procedure will continue to work, and a solution eventually found, even if some of the x^i may be wrong. Convergence failure of the non-linear equations, however, means that the intersection was never found – i.e. the problem had no solution, or at least we couldn't find it. This is of course serious, and we can hardly accept it.
Eclipse's way of handling the convergence irregularities reflect our view.

- A linear convergence failure will be reported as a warning (and a suggestion to increase LITMAX and/or NSTACK), but the procedure continues in the hope that it will be self-correcting (as we intuitively see the model problem in Figure 31 will be).
    - By the same kind of argument, it is not always necessary or important to force full convergence of the linear equations. If the linear solution is "good enough" to be used further in the procedure after a number of iterations, the procedure will not necessarily be accelerated by running still a few dozens of linear iterations to improve the linear solution marginally. These kinds of questions are very difficult to decide. In general, the best simulation performance can be expected when all linear equations are fully converged at each non-linear iteration.
- Non-linear convergence failure should not be taken lightly. Eclipse will then have to accept an erroneous solution, without knowing how bad it is (it could be so close to the sought solution that no harm is done, or it could be really way off – Eclipse only knows that it didn't come within the convergence tolerance). Eclipse (and the user) can only hope that the simulation will come "back on track" when the simulation is continued, but to increase the chances of success, the time step size is dramatically reduced for the following time steps.
    - Most simulations are run several times, with variation of the data. Often time intervals where convergence problems can be expected can be identified. In such cases it is often wise to reduce the time step before the simulator enters the interval.
    - How much the time step will be reduced (chopped) after the convergence failure is determined in the first record of TUNING
    - There is an absolute minimum for both time step size and chopable time step size. If convergence failure occurs and the time step cannot be reduced, the chances of finding the way back to the narrow path are slim. Such situations should be avoided at "any" cost!
- As a rule of thumb, rapid changes of reservoir state will require smaller time steps. The classic example is the start of water injection in the oil zone. Since water saturation at the start of the injection is at minimum value, water mobility is zero, so the injected water cannot flow before water saturation is increased, which it will not before the water flows. Solving this problem is definitely easier with a small time step, so one recommendation would be to reset the time step to some small value (e.g. one day) before each major well event.
    - No rule without exceptions. Although very rare, situations exist where the solution is so irregular that the convergence problems increase as the time step is reduced, and the most effective is to take one large time step crossing the troublesome interval in one go. (As said – this is exceptional, but should be mentioned...)

- A simulation that runs with alternating periods of time step increase and convergence failure / time step chopping / small time steps, will normally be much less efficient than a run with albeit shorter average time steps, but with no convergence problems. The optimal time step is the largest possible with no problems, but as exemplified above, this may vary (strongly) during the run, so it is absolutely not a trivial question.

TUNING keyword summarized

TUNING has three records
Record 1: Time step control
Record 2: Convergence tolerance parameters, recommended left at default values
Record 3: Iteration control.
Each record contains a number of parameters and is terminated by a slash.

The keyword syntax, with record 2 left default is,.

TUNING
TSINIT TSMAXZ TSMINZ TSMCHP TSFMAX TSFMIN TSFCNV TFDIFF ... /
/
NEWTMX NEWTMN LITMAX LITMIN MXWSIT .../

TSINIT	Max. length of *next* ministep (after this keyword) (default 1 day)
TSMAXZ	Max. length of any ministep (default 365 days)
TSMINZ	Min. length any ministep (default 0.1 day)
TSMCHP	Min. length any choppable ministep (default 0.15 day)
TSFMAX	Max. ministep increase factor (default 3)
TSFMIN	Min. ministep cutback factor (default 0.3)
TSFCNV	Cut factor after convergence failure (default 0.1)
TFDIFF	Max. increase factor after convergence failure (default 1.25)

NEWTMX	Maximum number of Newton iterations in a time step (default 12)
NEWTMN	Minimum number of Newton iterations in a time step (default 1)
LITMAX	Maximum number of linear iterations in each Newton iteration (default 25)
LITMIN	Minimum number of linear iterations in each Newton iteration (default 1)
MXWSIT	Maximum number of iterations within well flow calculation (default 8)

In addition, the number of search directions *stored* for use in the linear solver procedure is defined by the NSTACK keyword in the RUNSPEC section.

The TUNING keyword can be used anywhere in the SCHEDULE section, and as many times as desired. The parameters will be valid until the next TUNING keyword is encountered.

If the user only wants to change TSINIT, the size of the next time step, the keyword NEXTSTEP is an alternative to re-including the entire TUNING keyword.

19. Non-neighbour Connections and System Structure

We return to the structure of the system matrix as in Figures 29 and 30. In this section we will discuss how the structure shown in Figure 30 is affected by a fault, or more generally, by the presence of non-neighbour connections. The model segment we will study is shown in Figure 33.

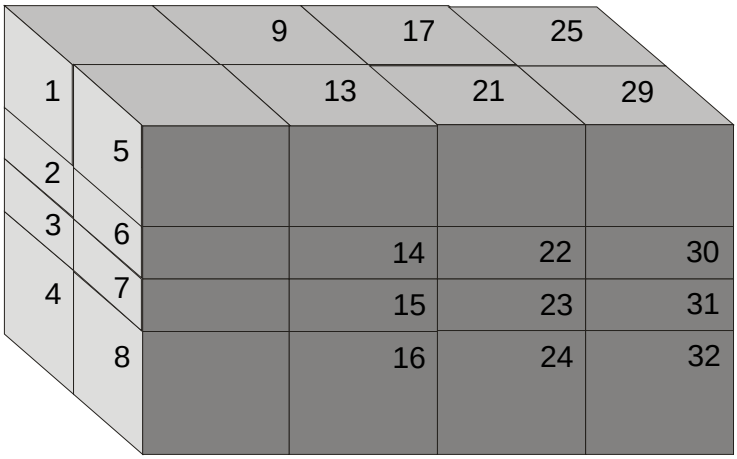


Figure 33. A grid with ZYX ordering

This time we use the ordering most often used by Eclipse, where "*direction 1*" is *Z*, "*direction 2*" is *Y*, and "*direction 3*" is *X*. The cell ordering is then seen in Figure 33.

This configuration will result in the system matrix shown in Figure 34.

	1				**5**					**10**					**15**					**20**					**25**					**30**		
1	x	x			x				x																								
	x	x	x			x				x																						
		x	x	x			x				x																					
			x	x				x				x																				
5	x				x	x							x																			
		x			x	x	x							x																		
			x			x	x	x							x																		
				x			x	x								x																	
	x								x	x			x				x															
10		x							x	x	x			x				x														
			x							x	x	x			x				x														
				x							x	x				x				x													
					x				x				x	x							x												
						x				x			x	x	x							x											
15							x				x			x	x	x							x									
								x				x			x	x								x									
									x								x	x			x				x								
										x							x	x	x			x				x							
											x	x						x	x				x										
20												x							x	x				x				x				
													x				x				x	x								x			
														x				x			x	x	x							x		
															x				x			x	x	x								x	
																x				x				x	x								x
25																	x								x	x			x			
																		x							x	x	x			x			
																			x							x	x	x			x		
																				x							x	x					x
																					x				x				x	x			
30																						x				x			x	x	x	
																							x				x			x	x	x	
																								x				x			x	x	

Figure 34. System structure for grid in Figure 33

Next we look at what happens if there is a fault between the rows $i = 2$ and $i = 3$ in the grid. The configuration is shown in Figure 35. (When working through this example we assume the sand to sand contact is as in the figure.)

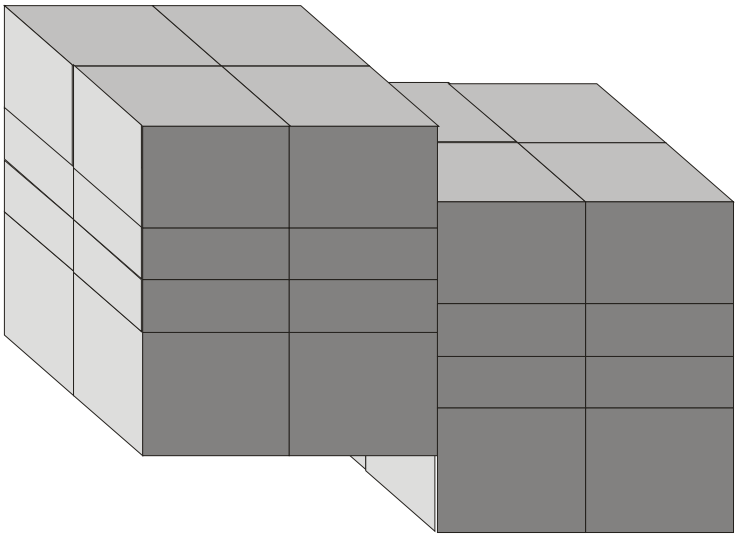


Figure 35. Grid as in Figure 33, but with fault

Note that some connections will disappear (cells are not in contact), while new contacts appear due to the fault. Connections which are present in Figure 33, but not in Figure 35 are marked with a "-" in the system, Figure 36, while new connections due to NNC-cells are marked with an "**n**".

	1				**5**					**10**					**15**					**20**					**25**					**30**		
1	x	x			x				x																								
	x	x	x			x				x																						
		x	x	x			x				x																					
			x	x				x				x																				
5	x				x	x							x																			
		x			x	x	x							x																		
			x			x	x	x							x																		
				x			x	x								x																	
	x								x	x			x				x															
10		x							x	x	x			x			**n**	-														
			x							x	x	x			x		**n**	**n**	-														
				x							x	x				x		**n**	**n**	x												
					x				x				x	x					**n**		x												
						x				x			x	x	x							**n**	-										
15							x				x			x	x	x						**n**	**n**	-									
								x				x			x	x							**n**	**n**	x								
									x	**n**	**n**						x	x			x				x								
										-	**n**	**n**					x	x	x			x				x							
											-	**n**						x	x	x			x					x					
20												x							x	x				x					x				
													x	**n**	**n**		x				x	x								x			
														-	**n**	**n**		x				x	x	x							x		
															-	**n**			x				x	x	x							x	
																x				x				x	x	x							x
25																	x								x	x			x				
																		x							x	x	x			x			
																			x							x	x	x			x		
																				x							x	x					x
																					x				x				x	x			
30																						x				x			x	x	x		
																							x				x			x	x	x	
																								x				x			x	x	

Figure 36. System structure for faulted grid

The most important consequence of the presence of the non-neighbour connections is that the 7-diagonal structure has been destroyed. Some elements have disappeared, while others have been added. In this particular example the difference isn't that large, but in general the NNC-entries can end up anywhere in the matrix, so techniques which utilise the 7-band structure can no longer be used. The zero-elements on the bands also pose problems.

Eclipse handles the NNC-entries as perturbations of the original structure, and project these entries onto the original bands. Effectively, the "alien" elements are moved to the right hand side of the

equation and lags one iteration behind in the procedure. This approach works satisfactory when the number of “irregular” entries is small, but if the number of NNCs become large (like more than 10% of ordinary connections) it should come as no surprise that convergence rate is severely reduced.

Obviously fault throws generate non-neighbour connections. In addition, the most common sources for NNCs are wells, aquifers, and local grid refinements.

A. GRF files in GRAF

The graphics post-processor GRAF is used for visualising summary data, and has been part of the Eclipse suite since the very beginning. Even though it has been "modernized" a bit, no one can argue that as an interactive tool it is at best cumbersome to use. However, GRAF has some very powerful batch processing capabilities, and utilizing these can make the program quite efficient in use.
The batch processing is enabled by the menu choice 8.2, execute GRF file. GRAF then reads batch commands from a so-called GRF file, and processes the commands to create and modify a set of pictures as requested by the user. The user will typically have one or several "standard" GRF files available, and by a quick editing of the appropriate file can generate the needed pictures quickly.
For a detail description of GRF commands, the user should consult the GRAF manual, but most of the more common commands are used in the examples below, which are conveniently commented.

A simple straightforward GRF file

```
-- Standard input characterisation
SET CONTINUATION '&'
--
SET INPUT UNFORMATTED
SET INPUT MULTIPLE
SET WSP UNFORMATTED
SET GRIDFILE EGRID
-- Load command. Loads the summary files EKS1.S000, EKS1.S001,...
-- Origin is used as a mnemonic, and is the name all create commands
-- later will use as reference (esp. relevant if several cases are loaded)
LOAD SUMMARY  EKS1  ORIGIN EKS1  NORESTART
--
-- PICTURE is an entire page
CREATE PICTURE 1
-- Every picture may contain several GRAPHS, which are separate plot
-- windows.
CREATE GRAPH 1
-- All curves are generated by "CREATE LINE" command, Y-axis VS X-axis.
-- X-axis is typically TIME or YEARS. The axes are ordered as they are
-- created, so X-axis is AXIS 1, first Y-axis is AXIS 2,...
-- Y-axis is the property to plot. All plot properties must be recognizable
-- Summary keywords (which have been loaded)
CREATE LINE FOPR VS YEARS EKS1
-- Another curve in the same plot. To use the same Y-axis as for FOPR,
-- specify OLDY 2, which means use the (old) AXIS 2 as Y-axis
CREATE LINE FWPR VS YEARS EKS1 OLDX 1 OLDY 2
-- FPR (Field pressure) has incompatible units with existing axis, so
-- specify new Y-axis. (Becomes AXIS 3)
CREATE LINE FPR VS YEARS EKS1 OLDX 1 NEWY
--
-- Create a new picture, containing two GRAPHS, i.e. two "windows"
-- First GRAPH contains four curves, second one.
CREATE PICTURE 2
CREATE GRAPH 1
CREATE LINE WOPR WP1 VS YEARS EKS1
CREATE LINE WOPR WP2 VS YEARS EKS1 OLDX 1 OLDY 2
CREATE LINE WOPT WP1 VS YEARS EKS1 OLDX 1 NEWY
CREATE LINE WOPT WP2 VS YEARS EKS1 OLDX 1 OLDY 3
--
CREATE GRAPH 2
CREATE LINE WWCT WP1 VS YEARS EKS1
END
```

Advanced GRF file

The following GRF file contains many “advanced” uses, and shows how a little work up front can save a lot of time later.

```
SET CONTINUATION '&'
--
SET INPUT UNFORMATTED
SET INPUT MULTIPLE
SET WSP UNFORMATTED
SET GRIDFILE EGRID
--
SET INPUT UNIFIED
--
-- In this example we define some variables at the start of the file.
-- If we later use the file for a different run, with another name, and
-- other well names, only these lines at the head of the GRF file need
-- to be changed.
-- Two different runs will be loaded, and for simplicity we use the same
-- ORIGIN name as the CASE name.
-- Variables in GRAF are identified by a trailing %.
ASSIGN CASE1% = EX3A
ASSIGN CASE2% = EX3B
ASSIGN WELLP1% = WP1A
ASSIGN WELLP2% = WP2AH
ASSIGN WELLI1% = WI1
ASSIGN WELLI2% = WI2B
--
LOAD SUMMARY  CASE1%  ORIGIN CASE1%  NORESTART
LOAD SUMMARY  CASE2%  ORIGIN CASE2%  NORESTART
--
CREATE PICTURE 1
CREATE GRAPH 1
CREATE LINE FOPR VS YEARS CASE1%
CREATE LINE FWPR VS YEARS CASE1% OLDX 1 OLDY 2
CREATE LINE FOPR VS YEARS CASE2% OLDX 1 OLDY 2
CREATE LINE FWPR VS YEARS CASE2% OLDX 1 OLDY 2
CREATE GRAPH 2
CREATE LINE FOPT VS YEARS CASE1%
CREATE LINE FWPT VS YEARS CASE1% OLDX 1 OLDY 2
CREATE LINE FOPT VS YEARS CASE2% OLDX 1 OLDY 2
CREATE LINE FWPT VS YEARS CASE2% OLDX 1 OLDY 2
--
--
CREATE PICTURE 2
CREATE GRAPH 1
CREATE LINE WOPR WELLP2% VS YEARS CASE1%
CREATE LINE WOPR WELLP2% VS YEARS CASE2%
CREATE GRAPH 2
CREATE LINE WOPR WELLP1% VS YEARS CASE1%
CREATE LINE WOPR WELLP1% VS YEARS CASE2%
CREATE GRAPH 3
CREATE LINE WOPT WELLP2% VS YEARS CASE1%
CREATE LINE WOPT WELLP2% VS YEARS CASE2%
CREATE GRAPH 4
CREATE LINE WOPT WELLP1% VS YEARS CASE1%
CREATE LINE WOPT WELLP1% VS YEARS CASE2%
--
.
.
.
-- Modification commands are used to change the default appearance of
-- the picture or graphs
```

```
MODIFY PICTURE 1
-- The BOUNDARY is the rectangle enclosing the entire picture
BOUNDARY POSITION 0.005 0.103 0.995 0.955
-- Redefine Picture title
TITLE IS Field Fluid Production rates and totals
-- Move title to desired position (can be done interactively first)
TITLE POSITION 0.307 0.688
MODIFY GRAPH 1
-- The VBOUNDARY is the View Boundary, i.e. the rectangle enclosing all
-- the GRAPHS
-- The GBOUNDARY is the GRAPH boundary, the rectangle enclosing each GRAPH
-- window.
-- Advice: To get the different BOUNDARY coordinates, first set the
-- rectangle interactively in GRAF for one picture. Then request writing
-- of GRF file, and use copy & paste to resize other pictures to the same
-- size.
VBOUNDARY POSITION 0.005 0.053 0.495 0.995
GBOUNDARY POSITION 0.096 0.069 0.988 0.843
-- Modify GRAPH title and position
GTITLE IS Field oil & water rates
GTITLE POSITION 0.163 0.776
-- Change character size in title
GTITLE HEIGHT 0.024
-- Move curve legends
CONTENTS POSITION 0.17 0.701
-- Change contents and character size for legend
MODIFY LINE 1
TITLE IS Oil rate CASE1%
TITLE HEIGHT 0.02
MODIFY LINE 2
TITLE IS Water rate CASE1%
TITLE HEIGHT 0.02
MODIFY LINE 3
TITLE IS Oil rate CASE2%
TITLE HEIGHT 0.02
MODIFY LINE 4
TITLE IS Water rate CASE2%
TITLE HEIGHT 0.02
-- Change axis 2 (the Y-axis)
MODIFY AXIS 2
-- Set min and max values on the axis
MIN 2000
MAX 4000
-- Set primary and secondary tickmarks
PTSPACE 1000
STSPACE  250
-- Set Tickmark label spacing
SCSPACE 1000
-- Use a field width of 6 characters on axis label
FORMAT 6
--
END
```

B. Some Considerations Regarding Grid Consistency

In a corner point grid, each cell is uniquely defined by its eight corners. A proper cell fulfils the following requirements, using the standard corner indexing (i.e. in a grid oriented parallel to the world axes the four top corners are indexed in the order nw, ne, sw, se, thereafter the four bottom corners are numbered in the same order).

1. The top and bottom surfaces should be oriented the same way. This can be checked by the *winding number*. Define a *winding vector* from the face centre to a corner, and the *winding angle* as the angle between the initial and current winding vector. Starting at corner 1 and passing through corners 2, 4, 3 and back to 1 (in that order) the winding vector should sweep through a complete circle such that the winding angle is increasing monotonically. When moving from corner *i* to *i+1* the winding number is incremented if the winding angle increases, decremented if it decreases. Hence for a proper face with four corners the winding number should be plus or minus four. For a proper cell the top face and bottom face winding numbers should be equal, normally +4.
2. All corners *i* should satisfy, (depth of corner $i+4$) $\geq$ (depth of corner *i*)
3. If two corners *c1* and *c2* are on the same coordinate line, *c1* belongs to a cell in layer *k1*, and *c2* belongs to a cell in layer *k2*, with $k2 > k1$, then (depth of *c2*) $\geq$ (depth of *c1*).
4. Optionally, if gaps are not permitted in the grid, then the coordinates of corners $(i, j, k, c+4)$ and $(i, j, k+1, c)$ should be equal.

These requirements ensure,

1. Coordinate lines do not intersect each other and maintain their internal ordering
2. No cells or part of cells should have negative thicknesses or negative volumes
3. Cells from different layers do not overlap
4. No gaps between layers allowed.

A grid where all cells fulfil the four requirements above is said to be *consistent*.

Eclipse will not permit active cells with negative volume, but apart from that just about anything is accepted. Inactive cells are not checked at all.

One consequence of Eclipse's lack of strictness has been the development of a "sloppy" attitude among grid builders. This does not need to be a serious drawback, as the grids may be quite useable for simulation anyway. But in some situations consistent grids are necessary. Occasionally, a grid is extended such that initially inactive parts of the grid become active. The extension may be very difficult to carry through if the inactive part is far from consistent.
Other simulators may be more restrictive than Eclipse, and may hence not accept a corner point grid that has been constructed and used by Eclipse. One particular case where consistency is mandatory is when the grid is planned used also to do rock mechanics simulations, which will be discussed below.

Hence there may be good reasons to plan and construct every grid such that they are consistent (also in the inactive parts) already from the outset.

Grids planned for use in rock mechanics simulations

A major difference between flow simulation and simulation of soil behaviour is the concept of active / inactive cells. A flow simulator characterizes any cell with vanishing pore volume as inactive, as such cells are not needed or used when solving for the flow state. When each cell represents a volume of material (soil), the concept of inactive obviously has no meaning, as a volume of soil will contribute to overall displacement and stress state, whether it is fluid-filled or not. By the same argument, void space as e.g. gaps cannot be permitted in a grid representing physical soil. It is thereby clear that grids for use in rock mechanics simulations must satisfy more strict requirements than grids used in flow simulation.

Embedding

Standard boundary conditions for a coupled rock mechanics – reservoir simulation run is to require a free top surface, else the edges are constrained to no displacement allowed. These boundary conditions are only realistic if applied to edges far from the reservoir, i.e. the model grid must be extended to include the entire overburden (such that the top free surface is the earth surface or sea bottom), and sufficient underburden and sideburdens that the boundaries do not influence displacement of the reservoir itself.

Hence, a reservoir grid is always extended to include a substantial volume of surrounding non-porous rock before the grid is ready for use in rock mechanics simulations. (The reservoir grid is embedded in the final grid).

When the reservoir grid has been finalised in advance, this embedding is normally done by adding grid blocks to all sides of the reservoir, typically extending grid block directions and with increasing block sizes towards the edges, a task that can be done with dedicated software. (For an Eclipse – Visage coupling the embedding can be done in the Visage suite program VisGen.) Although some constraints can be put on the grid extension process, the embedding is normally carried out without any attempt to honour actual rock topology in the extension area. A typical embedding of a simple reservoir grid is shown in Figure A1, where the original reservoir grid is shown as the thin blue slab in the middle of the final grid.

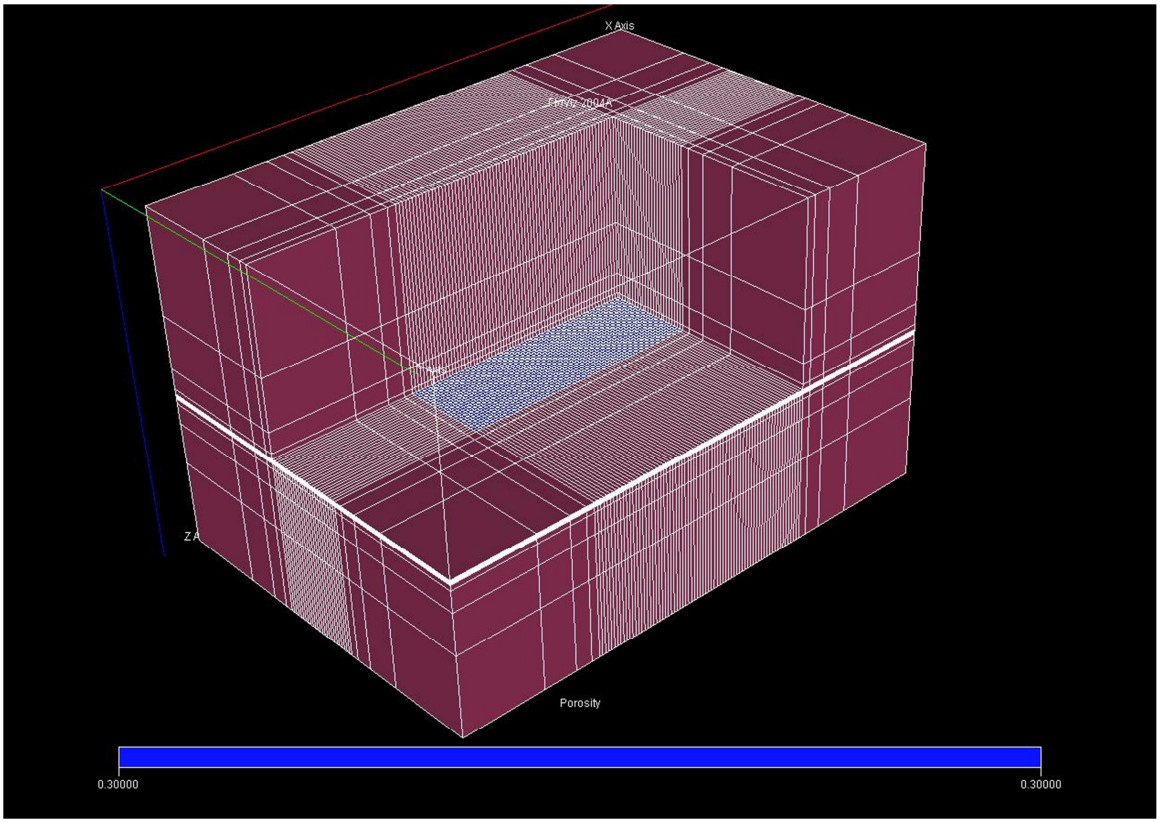


Figure A1. Embedding of Reservoir Grid

Since few if any reservoir engineers have such an embedding in mind when building the reservoir grid, the extension process can frequently run into problems. The commonly occurring problems are tied to non-consistent grids as discussed above, and the use of non-vertical coordinate lines, which will be discussed here.

Non-vertical coordinate lines

Coordinate lines can have any (non-horizontal) direction, and non-vertical coordinate lines are primarily used to honour actual fault geometries. Figure A2 shows a cross section of a fault block which bounding faults sloping in different directions.

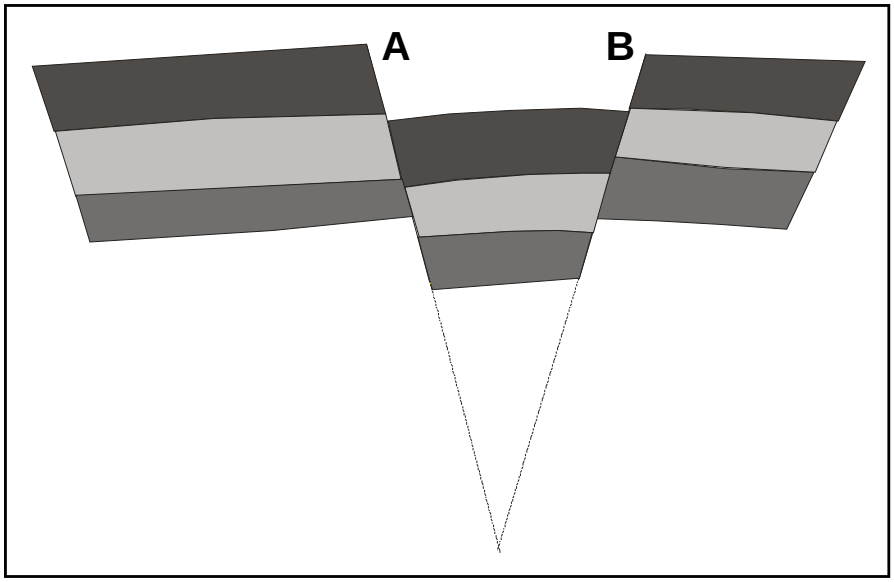


Figure A2. Coordinate lines and sloping faults (cross section view)

The coordinate lines A and B pose no problems in the reservoir grid (indicated by shades of grey). But if an underburden is added below the grid, lines A and B will intersect somewhere below the reservoir. Since the thickness of the over / underburden is normally several reservoir thicknesses, this problem is almost certain to emerge in the embedding process whenever not all coordinate lines are parallel. Fixing the problems at this stage is not a trivial task – especially if non-consistencies have to be fixed at the same time.

Hence the obvious recommendation is, for grids which are to be used in coupled simulations, to *build the entire grid including the embedding with the final purpose in mind from the outset*. This will often enforce compromises between desires / requirements for the reservoir grid and the embedded grid, but such conflicts are better solved when regarding all aspects of the problem simultaneously. (Certainly this advice is not easily applied to existing grids which have taken a considerable time to build, and where a request to redo the job is not welcomed…)

As an example, the situation depicted in Figure A2 cannot be resolved accurately. The intersecting coordinate lines cannot be accepted, so an adjustment of the fault slopes such that the coordinate lines A and B only just don't intersect in the underburden is perhaps to live with.

Honouring material properties of non-reservoir rock.

Aquifers

Aquifers should be handled differently in a coupled model than in flow simulation. Analytic or numeric aquifers are a convenient means of simulating aquifer pressure support to the reservoir, but cannot been used in such a setting when including rock mechanics effects. The aquifer is also a volume of porous soil, and hence it influences directly both the stress state and the reservoir pressure. Aquifers are a prime example of why we encourage to build the grid with the final embedded model as goal from the outset, not as a two-stage process.

Figure A3 is an example where the reservoir is connected to a large aquifer extending into the sideburden. This volume should obviously be modelled as fluid-bearing porous rock, not as soil inactive to flow as would result from a build-on embedding process. Note that a traditional numerical aquifer would not work here, since there is no way the node displacements in the aquifer volume could be accurately computed or accounted for by such an approach.

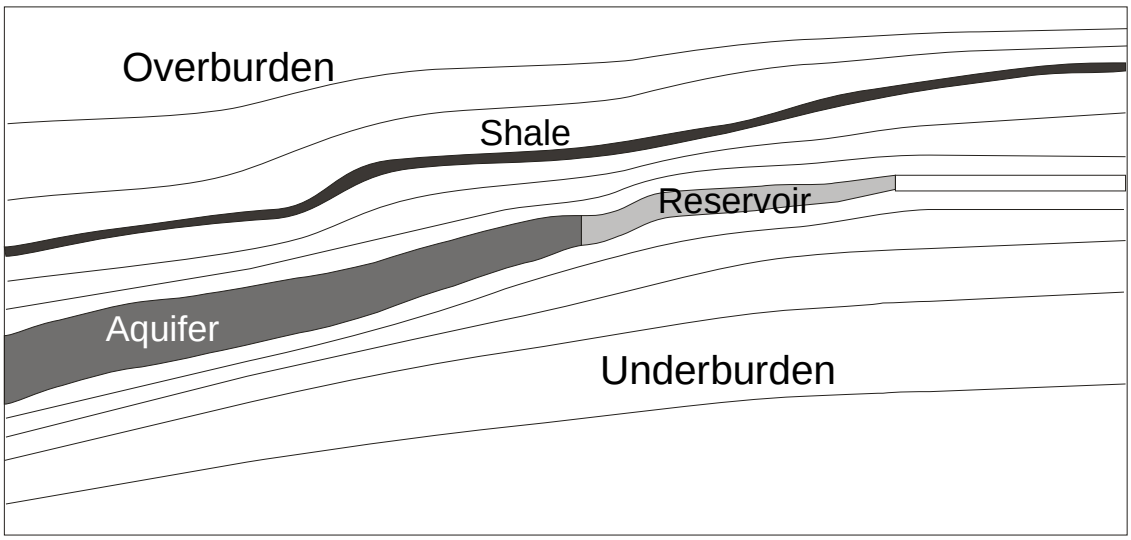


Figure A3. Honouring Geometry of non-reservoir soil (cross section view)

On the other hand, fluid flow in the aquifer is one-phase, so the aquifer can be modelled sufficiently accurate with grid block sizes of the same order that is normally used in the extension grid. The aquifer *could* be modelled as horizontal (using standard embedding procedure), but accounting for depth and thickness variation adds value – also honouring the non-horizontal non-porous to porous rock interface contributes to a more accurate description.

Other features

There may be other features in the over / under / sideburdens that could be advisable to capture in the grid. One example could be a (soft) shaly layer in the overburden, as indicated in Figure A3. The shale may have significantly different material properties from surrounding hard rock, and as such may have a non-neglectable influence on strain / displacement / failure. To capture this influence the layer should be modelled explicitly, even though this will increase the size of the finite element grid.

In Figure A3, suggested layering for the extension grid to over and underburden has been indicated by the thin solid lines.