Technical University of Denmark

Radon transport modelling: User's guide to RnMod3d

Andersen, Claus E.

Publication date:
2000

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Andersen, C. E. (2000). Radon transport modelling: User's guide to RnMod3d. (Denmark. Forskningscenter Risoe. Risoe-R; No. 1201(EN)).

Risø-R-1201(EN)

Radon Transport Modelling: User's Guide to RnMod3d

Claus E. Andersen

Risø National Laboratory, Roskilde, Denmark
August 2000

Risø-R-1201(EN)

Radon Transport Modelling: User's Guide to RnMod3d

Claus E. Andersen

Risø National Laboratory, Roskilde, Denmark
August 2000

Abstract `RnMod3d` is a numerical computer model of soil-gas and radon transport in porous media. It can be used, for example, to study radon entry from soil into houses in response to indoor-outdoor pressure differences or changes in atmospheric pressure. It can also be used for flux calculations of radon from the soil surface or to model radon exhalation from building materials such as concrete.

The finite-volume model is a technical research tool, and it cannot be used meaningfully without good understanding of the involved physical equations. Some understanding of numerical mathematics and the programming language Pascal is also required. Originally, the code was developed for internal use at Risø only. With this guide, however, it should be possible for others to use the model.

Three-dimensional steady-state or transient problems with Darcy flow of soil gas and combined generation, radioactive decay, diffusion and advection of radon can be solved. Moisture is included in the model, and partitioning of radon between air, water and soil grains (adsorption) is taken into account. Most parameters can change in time and space, and transport parameters (diffusivity and permeability) may be anisotropic.

This guide includes benchmark tests based on simple problems with known solutions. `RnMod3d` has also been part of an international model intercomparison exercise based on more complicated problems without known solutions. All tests show that `RnMod3d` gives results of good quality.

Version This guide concerns `RnMod3d`, version 0.8 (Sep. 15, 1997 – July 18, 2000).

Claus E. Andersen
Risø National Laboratory
Nuclear Safety Research Department
Building NUK-125
DK-4000 Roskilde, Denmark
Phone: +45 – 4677 4677 (main lab.)
Phone: +45 – 4677 4912 (direct)
Fax: +45 – 4677 4959
E-mail: `claus.andersen@risoe.dk`
Internet: `www.risoe.dk/nuk`

Printed August 21, 2000

ISBN 87-550-2734-2 (printed edition)

ISBN 87-550-2733-4 (internet edition)

ISSN 0106-2840

Information Service Department · Risø · 2000

Contents

1 Introduction

`RnMod3d` is a computer model of radon transport in porous media. It can been used for:

- flux calculations of radon from the soil surface into the atmosphere
- simulations of entry of soil gas and radon into houses in response to indoor-outdoor pressure differences or changes in atmospheric pressure
- calculation of radon exhalation from building materials
- error analysis of measurement procedures related to radon

The model is a technical research tool, and it cannot be used meaningfully without good understanding of the involved physical equations. Some understanding of numerical mathematics and the programming language Pascal is also required. Originally, the model was developed for internal use at Risø only. With this guide, however, it should be possible for others to use the model. The design has emphasized flexibility, robustness, programming transparency, and the ability to document and verify computations. Features such as speed, use of memory, and portability have been given a lower priority. The model can be run on personal computers with an Intel processor DX486 or above. *PC model*

1.1 What problems can be solved?

Three-dimensional steady-state or transient problems with Darcy flow of soil gas and combined generation, radioactive decay, diffusion and advection of radon can be solved. Moisture is included in the model, and partitioning of radon between air, water and soil grains (adsorption) is taken into account. Most parameters can change in time and space, and transport parameters (diffusivity and permeability) may be anisotropic.

The model can treat problems where *both* the soil-gas and the radon problem are time-dependent. For example, the model can calculate time-dependent combined diffusive and advective entry into a house when the flow of soil gas is created by changes in the atmospheric pressure.

1.2 What problems can not be solved?

Clearly it is difficult to list all the things `RnMod3d` can not do. Here are, however, the most important ones: The model cannot treat non-Darcy flow of soil gas or soil-gas flow in non-isothermal soil. In transient soil-gas simulations there are two restrictions: the air-filled porosity must be constant in time, and the pressure variations must be small (compared with the absolute pressure). The numerical procedures implemented in `RnMod3d` are relatively simple and the model is not particularly fast. Although the model in principle can treat time-dependent problems in full 3D, the computational time required to solve such problems can be too large to be of practical use. Finally, it is mentioned that `RnMod3d` is based on orthogonal grids. Hence, it is not possible to perform accurate calculations for complex geometries.

1.3 How to get a copy of `RnMod3d`

`RnMod3d` can be obtained from the author of this report.

1.4 Structure of this guide

The remaining part of this section tries to give an overview of what it takes to set up a problem. Section 2 tells how the model can be "installed" on a PC, and how the test case can be run. Section 3 presents the equations solved by `RnMod3d`. The numerical method is also described. After these "introductory" sections, Section 4 then describes all the so-called *control variables* used in `RnMod3d`. This is the reference section of the guide. Then from Section 5 and onwards specific issues are treated one by one. First it is shown how a grid is set up, then the idea of nodes and connectors are introduced etc. The final part of the guide gives examples of computations performed with `RnMod3d`.

1.5 How to use `RnMod3d`

Job file

To do calculations with `RnMod3d` it is necessary for the user to write, compile, and run a Pascal program. The program is here called a *job file*. An example is shown in the appendix starting page 94. The job file contains a link to the `RnMod3d` code plus all information about the problem in question: the computational grid, boundary and initial conditions, soil parameters, what output to calculate etc. To set up a job, the user needs to make proper assignments to what is here called *control variables*. Many of the control variables are Pascal pointers to user-defined functions or procedures. This design makes the model highly flexible.

Control variables

The most difficult part of setting up a job is probably to define the geometry of the problem. However, when the geometrical model has first been established (and verified) it is an easy task to change parameters, and to get the output of interest. The "geometrical model" can be saved and used (directly or in modified form) in other computations.

1.6 Making a job file

`run_model`

The structure of a job file is very simple: First, values are assigned to control variables. Then, `RnMod3d` is run by calling the predefined procedure `run_model`. Thereafter, it is possible to redefine one or more of the control variables, and additional simulations can be done with `run_model`. Each time `run_model` is called, `RnMod3d` performs a simulation corresponding to the settings of the control variables. The primary result of a simulation is pressures and flows of soil gas and/or concentrations and fluxes of radon. After the simulations, the procedure `close_model` is called (once) to close output files etc.

`close_model`

The structure of a prototype job file is shown in example 1. Only a few declarations and control-variable assignments are shown. Most code lines have been left out as indicated by the dots. The example includes two runs. In the first run, `RnMod3d` solves the problem on the basis of the coarse grid specified in the procedure `my_coarse_grid`. In the second run, a finer grid is used. Which grid is used is controlled by the pointer: `grid_def`. The difference in results from run 1 to 2 will show how sensitive the solution is to the selected grid.

Example 1 *Prototype structure of a job file.*

```
program F001prg;
{$I R3dirs03}
uses R3Defi03,R3Main03,R3Writ03; (* Links to RnMod3d *)

procedure my_coarse_grid;
begin
set_FixVal(xFix1,0.0); (* xFix1 = 0.0 m *)
set_FixVal(xFix2,3.3); (* xFix2 = 3.3 m *)
set_axis_single(xFix1,xFix2,5,FocusA,1.0) (* Allocate 5 nodes between xFix1 and xFix2 *)
```

```
...
end;

procedure my_fine_grid;
begin
set_FixVal(xFix1,0.0);
set_FixVal(xFix2,3.3);
set_axis_single(xFix1,xFix2,33,FocusA,1.0) (* Allocate 33 nodes between xFix1 and xFix2 *)
...
end;
...
begin (* main *)
runid:='001';
...
grid_def:=my_coarse_grid;
run_model; (* first run *)

grid_def:=my_fine_grid;
run_model;  (* second run *)

close_model;
end.
```

`grid_def` is just one single control variable (out of more than 60). The difference from run to run could have related to almost any other aspect of the computation: boundary conditions, material properties, requirement for convergence, numerical scheme, relaxation etc. `RnMod3d` is therefore particularly well suited for sensitivity analyses. As another example, imagine that entry into a house has to be calculated for a range of 10 permeabilities and 10 indoor-outdoor pressure differences contained in two arrays: `perm` and `press` defined by the user. Such a sensitivity analysis could be programmed as follows:

Sensitivity analyses

Example 2 *Prototype sensitivity analysis.*

```
program F002prg;
{$I R3dirs03}
uses R3Defi03,R3Main03,R3Writ03;

var ii,jj:1..10;
    perm,pres:array[1..10] of real;
...
begin (* main *)
runid:='002';
...
perm[1]:=1e-14; (* m2 *)
perm[2]:=2e-14;
perm[3]:=1e-13;
...
pres[1]:=-10; (* Pa *)
pres[2]:=-8;
pres[3]:=-3;
...
for ii:=1 to 10 do
  for jj:=1 to 10 do
    begin
      ... (* set permeability to perm[ii] and pressure to pres[jj] *)
      run_model;
    end;
close_model;
end.
```

The steps needed to set up a job file are described in the following.

Run identification

Before anything else, assign the job an identification tag with the `runid` control variable. If the job file is saved under the name: `F0997prg.dpr` then it would be convenient to set `runid := '0997'` because then standard output from `RnMod3d` runs will go to files such as `F0997LOG.dat` and `F0997FLW.dat`. This makes it easy to find out what files belong to what jobs.

Geometry

Then the geometry of the problem should be considered. All dimensions (for example, of building components) of importance for the problem should be identified and formally set up as so-called *fix points* called `xFix1`, `xFix2` etc. A link must then be established between the physical (x, y, z) world in meters and a three-dimensional grid of *control volumes* with index coordinates `(i,j,k)`. Initially (i.e. before the model set-up has been fully verified), it is best to use only a very coarse grid. Fortunately, the control-volume approach guarantees that even results obtained with coarse grids are physically meaningful (for example, the solution will not become unstable and the radon concentrations will not become negative for this reason). In the end, the grid must, however, have a sufficiently high resolution, otherwise the results will be too inaccurate. The use of fix points means that these points do not move as grids with more control volumes are used.

Fix points

Control-volume grid

Nodes and connectors

Then it must be defined how each control volume should "work". Most control volumes will be under the control of the transport equations for radon or soil gas, but some others may be fixed at certain external values (boundary conditions) or may not be part of the computations at all. In `RnMod3d`, each control volume has a property called *node type* which reflects these aspects. Likewise, some (adjacent) control volumes will be connected and some others will be disconnected. These aspects are specified in control-volume properties called *connectors*. Each control volume has six connectors (one for each neighbor). The user can set all *nodes* and *connectors* in accordance with the problem in question.

Materials

The next step is to define material properties. For example, in a simulation of radon transport, it is of course necessary to specify values for porosity and diffusivity etc. The assignment of material properties can be based on physical (x, y, z) coordinates (for example, it can be specified that the radon generation rate should change with soil depth in some user-defined way or that the top-soil moisture content should change in time because it rains). It is also possible to divide the computational plane into blocks of materials (called `mat1`, `mat2` etc.) and to set the material properties to be block-wise constant.

Output

`RnMod3d` solves the specified equations and returns field values at all nodes in the `(i,j,k)` computational field. This type of output is, however, seldom the end result from the user's point of view. Often the prime output will be field values or fluxes at a few selected locations (given in physical (x, y, z) coordinates). To get this type of output without troubles, `RnMod3d` is equipped with special "field measurement probes" (called `Obs1`, `Obs2` etc.) and "flux measurement probes"

Flux "probes" etc.

(called Flx1, Flx2 etc.). In radon simulations, this framework provides the user with the ability to monitor radon concentrations and fluxes of radon. In soil-gas simulations the probe values correspond to pressures and soil-gas flow rates. The "field measurement probes" are normally placed at given points defined by physical (x, y, z)-coordinates. In contrast, "flux measurement probes" are normally defined in relation to plane surfaces defined by reference to fix points.

2 Installation

To use RnMod3d, a Pascal compiler must be installed and the five source files (listed in Section 2.3) must be copied to a directory "visible" for the compiler together with the test job file: F0000prg.dpr.

2.1 Pascal compiler (Delphi)

Delphi

It is best to run RnMod3d from the editor/compiler environment of Delphi (only Delphi 3 has been tested, but other versions are probably all right). Observe, that RnMod3d is a pure console application. No use is made of the Windows user-interface in Delphi. Here is what to do to make an old-fashioned hello-world program. A job file for RnMod3d can be made in the same way.

1. Open Delphi.
2. Create a new application using *File* | *New Application.*
3. Go to the *Project Manager* (*View* | *Project Manager*).
4. Remove the default form from the project (highlight the unit and hit *delete*). Do not save changes.
5. Go to *Project Source* (*View* | *Project Source*).
6. Edit the project source file:
   - Remove code inside `begin end`.
   - Replace the `Forms` unit in the `uses` section with `SysUtils`.
   - Remove `{$R *.RES}`.
   - Place `{$apptype console}` in a line by itself right after the `program` statement.
7. Add whatever statements needed in the body of the program. The program can look like this:

```
program test;
{$apptype console}
begin
writeln('Hi there');
readln;
end.
```

8. Compile the program with (*Project* | *Compile*).
9. Run the program with *Run* | *Run.*
10. When the program is saved, it is best to use the default extension for Delphi projects: `.dpr`. Any units that are created should be saved with extension: `.pas`.

2.2 Pascal compiler (Borland Pascal 7)

RnMod3d can also be run from Borland Pascal 7. The only change is that the following two lines must be removed from the code file R3Dirs03.pas:

```
{$DEFINE Delphi}
{$apptype console}
```

There are three reasons why it is best to run RnMod3d from Delphi:

- RnMod3d runs much slower under Borland Pascal v. 7 compared with Delphi (a factor of 2 or such).
- The memory model is better in Delphi.
- The max_time control variable does not work under Borland Pascal (see Section 4.58)

2.3 Source files

RnMod3d consists of more than 5500 code lines. The code is placed in the following five files:

R3Dirs03.pas	Compiler directives
R3Defi03.pas	Global declarations
R3Main03.pas	The main program
R3Writ03.pas	Additional procedures (mainly output routines)
R3Delp03.pas	Special code for Delphi and Borland Pascal 7

Code directory These files should be placed in the working directory or better in a separate code directory. In the latter case, Delphi needs to know about this directory. This is done by adding the path (e.g. d:\data\pascal\rnmod3d\code) in the *Libary Path* (*Environmantal Options | Library*). It is advisable to make the code files *read only*.

2.4 Test case: F0000prg.dpr

F0000prg.dpr is a test job file where everything is defined by default. The (hidden) problem that is solved is a simple heat-conduction problem of no interest here. It just serves as a simple test. When the model is run, the following two (output) files are created: f0000LOG.dat and f0000RES.dat. The first is a log file with all sorts of output. The second is a general purpose result file. In the default case, no output goes to the RES file. If everything works, running F0000prg.dpr should give: Flx1 : J = -1.0000000E-0006 (this is a flux) and Obs1 : c = 1.5000000E+0000 (this is a concentration).

Example 3 *Test case:* F0000prg.dpr

```
program F0000prg;
{$I R3dirs03}
uses R3Defi03,R3Main03,R3Writ03;
begin
default_problem;
run_model;
close_model;
end.
```

3 Method

The purpose of this section, is to present the basic transport equations solved by `RnMod3d`. The framework is consistent with that used in [An92] and [An99^c]. An outline of the finite-volume approach is also given.

3.1 Basic definitions

Consider a reference element δV of soil. This volume may be split into three parts: δV_{g} for the volume of grains, δV_{w} for the volume of water, and δV_{a} for the volume of air:

$$\delta V = \delta V_{\mathrm{g}} + \delta V_{\mathrm{w}} + \delta V_{\mathrm{a}} \tag{1}$$

Hence the (total) porosity ϵ, the water porosity ϵ_{w}, and the air porosity ϵ_{a} can be expressed as:

$$\epsilon = \frac{\delta V_{\mathrm{w}} + \delta V_{\mathrm{a}}}{\delta V} \tag{2}$$

$$\epsilon_{\mathrm{w}} = \frac{\delta V_{\mathrm{w}}}{\delta V} \tag{3}$$

$$\epsilon_{\mathrm{a}} = \frac{\delta V_{\mathrm{a}}}{\delta V} \tag{4}$$

We define the fraction of water saturation of the pore volume (i.e. the volumetric water content) as:

$$\theta_{\mathrm{v}} = \frac{\delta V_{\mathrm{w}}}{\delta V_{\mathrm{a}} + \delta V_{\mathrm{w}}} = \frac{\epsilon_{\mathrm{w}}}{\epsilon} \tag{5}$$

Hence $\theta_{\mathrm{v}} = 1$ means that the pores are completely filled with water whereas $\theta_{\mathrm{v}} = 0$ means that the soil is dry. The total mass of the reference element is:

$$\delta M = \delta M_{\mathrm{g}} + \delta M_{\mathrm{w}} \tag{6}$$

where δM_{g} is the mass of grain material and δM_{w} is the mass of water. The mass of air is neglected. The density of the grain material is:

$$\rho_{\mathrm{g}} = \frac{\delta M_{\mathrm{g}}}{\delta V_{\mathrm{g}}} \tag{7}$$

For a wide range of soils ρ_{g} is in the (narrow) range from 2.65 to $2.75 \cdot 10^3$ kg m^{-3}. The density for water:

$$\rho_{\mathrm{w}} = \frac{\delta M_{\mathrm{w}}}{\delta V_{\mathrm{w}}} \tag{8}$$

is about $1.0 \cdot 10^3$ kg m^{-3}. The wet-soil density for given porosity and water content can be calculated as:

$$\rho_{\mathrm{ws}} = \frac{\delta M}{\delta V} = (1-\epsilon)\rho_{\mathrm{g}} + \theta_{\mathrm{v}}\rho_{\mathrm{w}} \tag{9}$$

The dry-soil density is:

$$\rho_{\mathrm{ds}} = \frac{\delta M_{\mathrm{g}}}{\delta V} = (1-\epsilon)\rho_{\mathrm{g}} \tag{10}$$

We define the amount of water per dry mass of soil (i.e. the gravimetric water content) as:

$$\theta_{\mathrm{g}} = \frac{\delta M_{\mathrm{w}}}{\delta M_{\mathrm{g}}} = \frac{\rho_{\mathrm{w}}\delta V_{\mathrm{w}}}{\rho_{\mathrm{g}}\delta V_{\mathrm{g}}} = \frac{\epsilon_{\mathrm{w}}}{1-\epsilon}\,\frac{\rho_{\mathrm{w}}}{\rho_{\mathrm{ds}}} = \frac{\epsilon}{1-\epsilon}\,\frac{\rho_{\mathrm{w}}}{\rho_{\mathrm{ds}}}\,\theta_{\mathrm{v}} \tag{11}$$

Hence if the porosity of the soil is $\epsilon = 0.3$, then full water saturation ($\theta_{\mathrm{v}} = 100$ %) means that the amount of water per dry mass is normally about $\theta_{\mathrm{g}} = 16$ %.

3.2 Radon transport equation

The total activity δA of radon-222 (simply referred to as "radon" in all of the following) in the reference element δV may be split into three parts:

$$\delta A = \delta A_{\mathrm{g}} + \delta A_{\mathrm{w}} + \delta A_{\mathrm{a}} \tag{12}$$

where the indices have the same meaning as in equation 1. We now define the concentration of radon in the air-filled parts of the pores as:

$$c_{\mathrm{a}} = \frac{\delta A_{\mathrm{a}}}{\delta V_{\mathrm{a}}} \tag{13}$$

and the radon concentration in the water-filled parts of the pores as:

$$c_{\mathrm{w}} = \frac{\delta A_{\mathrm{w}}}{\delta V_{\mathrm{w}}} \tag{14}$$

Part of the grain activity δA_{g} is available for transport in the pore system. This is the radon adsorbed to soil-grain surfaces: $\delta A_{\mathrm{g,s}}$. The immobile part $(\delta A_{\mathrm{g}} - \delta A_{\mathrm{g,s}})$ is radon produced by the "non-emanating" part of the grain radium. In line with the framework presented by Rogers and Nielson [Rog91A], we introduce the sorbed radon concentration per kg dry mass (Bq kg^{-1}) as:

$$c_{\mathrm{s}} = \frac{\delta A_{\mathrm{g,s}}}{\delta M_{\mathrm{g}}} \tag{15}$$

where δM_{g} is the grain mass within δV.

We assume rapid sorption kinetics [Wo92] such that the partitioning of radon between air, water and soil grains is permanently in equilibrium at any point of the soil:

$$\begin{aligned} c_{\mathrm{w}} &= L c_{\mathrm{a}} && (16) \\ c_{\mathrm{s}} &= K c_{\mathrm{a}} && (17) \end{aligned}$$

where L is the Ostwald partitioning coefficient given in Table 1, and K is the radon surface sorption coefficient [Rog91A, Na92]. The equilibrium assumption simplify the problem considerably as we can then express the total mobile radon activity by reference to the concentration in just one phase. Normally, the radon concentration in the air phase c_{a} is selected as "reference concentration". This approach is also taken in `RnMod3d`. The mobile activity in δV is hence given as:

$$\delta A_{\mathrm{g,s}} + \delta A_{\mathrm{w}} + \delta A_{\mathrm{a}} = \beta c_{\mathrm{a}} \delta V \tag{18}$$

where

$$\beta = \epsilon_{\mathrm{a}} + L\epsilon_{\mathrm{w}} + K\rho_{\mathrm{ds}} \tag{19}$$

is sometimes called the partition-corrected porosity. If the medium is dry and without grain sorption, we have: $\beta = \epsilon$. The equilibrium assumption is widely used in models of pollutant transport, but is not universally correct [Th97]. Support for the assumption can be found in [Na88, Na92]

If radium is present only in soil grains, we define the radon generation rate per pore volume (Bq s^{-1} per m^3-pore) as:

$$G = \frac{\lambda \rho_{\mathrm{ds}} E}{\epsilon} = \lambda E \frac{1-\epsilon}{\epsilon} \rho_{\mathrm{g}} \tag{20}$$

where λ is the decay constant of radon ($2.09838 \cdot 10^{-6}$ s^{-1}), and E is the emanation rate of radon to the soil pores (i.e. the number of atoms that emanates into water and air per second per kg dry mass). We can write the emanation rate as $E = fA_{\mathrm{Ra}}$, where f is the fraction of emanation and A_{Ra} is the activity concentration (Bq kg^{-1}) of radium-226 per dry mass.

Table 1. Radon solubility L in water as function of temperature (from [Cl79], p. 228).

Temperature	L
K	-
273.15	0.5249
278.15	0.4286
283.15	0.3565
288.15	0.3016
293.15	0.2593
298.15	0.2263
303.15	0.2003
308.15	0.1797

A mass-conservation equation for the mobile radon activity in δV is:

$$\frac{\partial \beta c_{\mathrm{a}}}{\partial t} = \epsilon G - \lambda \beta c_{\mathrm{a}} - \nabla \cdot \vec{j} \tag{21}$$

where $\vec{j}$ is the bulk flux density (in units of $\mathrm{Bq\,s^{-1}}$ per $\mathrm{m^2}$) at time t. The term 'bulk' means that the density is measured per total cross-sectional area perpendicular to $\vec{j}$. Hence, a flux J ($\mathrm{Bq\,s^{-1}}$) across some plane with geometric area A (e.g. a 120 $\mathrm{m^2}$ crawl-space floor) and uniform bulk flux density $\vec{j}$ gives: $J = \vec{j} \cdot A\hat{a}$, where $\hat{a}$ is a unit vector perpendicular to the plane.

The bulk flux density is divided into two:

$$\vec{j} = \vec{j}_a + \vec{j}_d \tag{22}$$

Ignoring water movement, the advective flux density is given by:

$$\vec{j}_a = c_{\mathrm{a}} \vec{q} \tag{23}$$

where $\vec{q}$ is the bulk flux density of soil gas (in units of $\mathrm{m^3\,s^{-1}}$ per $\mathrm{m^2}$) discused later. We assume, that the diffusive flux can be written as:

$$\vec{j}_d = -D \nabla c_{\mathrm{a}} \tag{24}$$

such that the bulk diffusivity D accounts for radon diffusion through air and water in the pores. D is a function of temperature and pressure [Wa94] and may therefore (if not for other reasons) change in time and space. We assume, that the soil-gas flow is so low that mechanical dispersion can be ignored (i.e. D is independent of $\vec{q}$) [Do92].

3.3 Soil-gas transport equation

It is assumed that the flow is of the Darcy type, that the soil has a uniform temperature (natural convection in the soil is ignored), and that ϵ_{a} is constant in time. Also, it is assumed that pressure variations are small in comparison with the absolute pressure. The equation can be derived as given next.

The equation of continuity for soil gas transport is [Bi60]:

$$\frac{\partial \epsilon_{\mathrm{a}} \rho_{\mathrm{a}}}{\partial t} = -\nabla \cdot (\rho_{\mathrm{a}} \vec{q}) \tag{25}$$

where ρ_{a} is the density of the gas (in $\mathrm{kg\,m^{-3}}$). For an ideal gas under isothermal conditions, ρ_{a} is proportional to the absolute pressure $P(x, y, z, t)$ (in Pa). Hence, we have:

$$\frac{\partial \epsilon_{\mathrm{a}} P}{\partial t} = -\nabla \cdot (P\, \vec{q}) \tag{26}$$

We can split the absolute pressure into three parts:

$$P(x,y,z,t) = P_0 - \rho_{\mathrm{a},0}\, g\, z + p(x,y,z,t) \tag{27}$$

where P_0 is the mean pressure at the atmospheric surface, and where p is the disturbance pressure field. The middle term consists of: the average air density at the given temperature $\rho_{\mathrm{a},0}$ (about 1.3 $\mathrm{kg\,m^{-3}}$), the acceleration due to gravity g (about 9.8 $\mathrm{m\,s^{-2}}$), and the depth $-z$ below the atmospheric surface (located at $z = 0$). The z-axis points upwards. The "aerostatic" pressure:

$$P_H(z) = P_0 - \rho_{\mathrm{a},0}\, g\, z \tag{28}$$

increases about 13 Pa per m depth.

The left-hand side of equation 26 can be evaluated as follows:

$$\begin{aligned} \frac{\partial \epsilon_{\mathrm{a}} P}{\partial t} &= \frac{\partial \epsilon_{\mathrm{a}} (P_H(z) + p(x,y,z,t))}{\partial t} && (29) \\ &= P_H(z) \frac{\partial \epsilon_{\mathrm{a}}}{\partial t} + \frac{\partial \epsilon_{\mathrm{a}} p}{\partial t} && (30) \end{aligned}$$

We limit the treatment to the situation when ϵ_{a} is constant in time, and we therefore have:

$$\frac{\partial \epsilon_{\mathrm{a}} P}{\partial t} = \frac{\partial \epsilon_{\mathrm{a}} p}{\partial t} \tag{31}$$

On the right-hand side of equation 26, we assume that the disturbance pressure is small in comparison with $P_H(z)$ such that:

$$\begin{aligned} P\, \vec{q} &= (P_H(z) + p)\, \vec{q} && (32) \\ &\approx P_0\, \vec{q} && (33) \end{aligned}$$

From this, we can approximate equation 26 as:

$$\frac{\partial \epsilon_{\mathrm{a}} p}{\partial t} = -\nabla \cdot (P_0 \vec{q}) \tag{34}$$

or

$$\frac{\epsilon_{\mathrm{a}}}{P_0} \frac{\partial p}{\partial t} = -\nabla \cdot \vec{q} \tag{35}$$

where $\vec{q}$ is given by Darcy's law:

$$\vec{q} = -\frac{k}{\mu} \nabla p \tag{36}$$

In the special case of homogeneous soil, we can reduce equation 35 and 36 to:

$$\frac{\partial p}{\partial t} = D_p \nabla^2 p \tag{37}$$

which is a usual diffusion equation, where

$$D_p = \frac{k P_0}{\mu \epsilon_{\mathrm{a}}} \tag{38}$$

is the diffusivity. Observe, that without the important simplification in equation 33, we would had obtained a transport equation with the term: $\nabla^2 p^2$. Instead, only $\nabla^2 p$ is part of the final equation[1]. Hence, equation 33 has lead to a linearization of the problem.

[1] Observe, that in one dimension,

$$\frac{\partial}{\partial x}\left(p \frac{\partial p}{\partial x}\right) = \frac{1}{2} \frac{\partial^2}{\partial x^2} p^2 \tag{39}$$

`RnMod3d` solves the following equation for radon transport:

$$\frac{\partial \beta c_\mathrm{a}}{\partial t} = \epsilon G - \lambda \beta c_\mathrm{a} - \nabla \cdot \vec{j} \tag{40}$$

where

$$\vec{j} = c_\mathrm{a}\vec{q} - D\nabla c_\mathrm{a} \tag{41}$$

is the bulk flux density of radon (in Bq s^{-1} per m^2) and where

c_a is the radon concentration in the air-filled parts of the pores (Bq m^{-3})

t is the time (s)

$\beta = \epsilon_\mathrm{a} + L\epsilon_\mathrm{w} + K\rho_\mathrm{ds}$ is the partition-corrected porosity (dimensionless)

ϵ is the porosity (dimensionless)

G is the radon generation rate per pore volume (Bq s^{-1} per m^3)

λ is the decay constant for radon ($2.09838 \cdot 10^{-6}$ s^{-1} for radon-222)

D is the bulk diffusivity (m^2 s^{-1})

$\vec{q}$ is a known bulk flux density of soil gas (m^3 s^{-1} per m^2)

Box 1: Radon transport equations.

`RnMod3d` solves the linearized equation for soil-gas transport:

$$\frac{\epsilon_\mathrm{a}}{P_0}\frac{\partial p}{\partial t} = -\nabla \cdot \vec{q} \tag{42}$$

where:

$$\vec{q} = -\frac{k}{\mu}\nabla p \tag{43}$$

is the bulk flux density of soil gas (in m^3 s^{-1} per m^2), and where

p is the disturbance pressure (Pa)

t is the time (s)

ϵ_a is the air porosity (dimensionless)

P_0 is the mean absolute pressure (about 10^5 Pa)

k is the gas permeability (m^2)

μ is the dynamic viscosity (about $17.5 \cdot 10^{-6}$ Pa s at 10 °C)

Box 2: Soil-gas transport equations.

3.4 `RnMod3d` treatment of radon and soil gas

`RnMod3d` is programmed to solve equations 40 and 41 in Box 1 and the formalism used to define problems in `RnMod3d` closely follows that used in these equations. For example, the control variable relating to the radon-decay constant (λ) is called `lambda_def` in `RnMod3d`. Hence, equation 40 and 41 relate to `RnMod3d` in a straightforward manner. For the soil-gas problem, the situation is a bit more complicated. First, we observe, that equations 42 and 43 in Box 2 are also of the *form* given in equations 40 and 41. We just have to substitute c_a with p, D with $\frac{k}{\mu}$ and β with ϵ_a/P_0. The rest of the "radon-equation coefficients" must be set to zero: $\lambda = 0$, $G = 0$, $\epsilon = 0$ and $\vec{q} = 0$. If we do that, `RnMod3d` solves the soil-gas problem as defined by equations 42 and 43.

Translation table

Table 2 should ease the translation of quantities used in `RnMod3d` and the two

Table 2. Quantities etc. used by `RnMod3d` *in radon and soil-gas problems.*

`RnMod3d`	Radon problem		Soil-gas problem	
	Equation 40 + 41		Equation 42 + 43	
Basic field value	c_a	$[\mathrm{Bq\,m^{-3}}]$	p	$[\mathrm{Pa}]$
`D_def`	D	$[\mathrm{m^2\,s^{-1}}]$	$\frac{k}{\mu}$	$[\mathrm{m^2\,Pa^{-1}\,s^{-1}}]$
`e_def`	ϵ	[-]	0	
`beta_def`	β	[-]	$\frac{\epsilon_\mathrm{a}}{P_0}$	$[\mathrm{Pa^{-1}}]$
`G_def`	G	$[\mathrm{Bq\,s^{-1}\,m^{-3}}]$	0	
`lambda_def`	λ	$[\mathrm{s^{-1}}]$	0	
`flowfield`	`import`		`export`	
`J`	$\int_\Omega \vec{j}\cdot\mathrm{d}\vec{a}$	$[\mathrm{Bq\,s^{-1}}]$	$\int_\Omega \vec{q}\cdot\mathrm{d}\vec{a}$	$[\mathrm{m^3\,s^{-1}}]$
`Q`	$\int_\Omega \vec{q}\cdot\mathrm{d}\vec{a}$	$[\mathrm{m^3\,s^{-1}}]$	0	

sets of transport equations. The first line of the table concerns the "field values" used by `RnMod3d`. For radon problems, these field values represent the radon concentration in the air-filled pore parts (c_a). For soil-gas problems, they correspond to the disturbance pressure (p). Therefore these quantities should be used when specifying fixed-value boundary conditions. Furthermore, it should be observed that model output of field values are based on these quantities.

Dual meaning of material properties

The next lines concern control variables `D_def` to `Lambda_def`. These control variables are Pascal pointers to user-defined functions as described in Section 7. Here we just state that their meaning relates directly to the coefficients of equations 40 and 41. Hence, for radon problems, `D_def` should point to the user-defined function where the bulk diffusivity D of the material is defined and `e_def` should point to the user-defined function of (total) porosity. For soil-gas problems, the situation is different as already stated: `D_def` should point to the function where the gas permeability divided by the dynamic viscosity is defined, `e_def` should point to a function which always return zero etc.

Flow field

The next line of the table concerns the soil-gas flow field $\vec{q}$. It links the soil-gas problem and the radon problem. It must be observed, that in the soil-gas equation, $\vec{q}$ *results* from the calculation. Its relation to the pressure field is given in equation 43. In advective radon problems, $\vec{q}$ is a *known* flow field of soil gas. `RnMod3d` has a control variable called `flowfield`. In a soil-gas simulation, we may set this to `export` meaning that the calculated flow field $\vec{q}$ should be output (exported) to a file. Later, in a radon simulation, we may want to import this soil-gas flow field. We can do that by setting `flowfield` to `import`. Other settings are also possible.

Output "probes"

The two final lines of the table concern `RnMod3d` "probes" for flux measurements. More details can be found in Section 8. Here we just mention, that in radon problems `J` and `Q` give the flux of radon and soil gas, respectively. In problems with pure diffusion, the soil-gas flow will be zero. In soil-gas problems, `J` is the calculated flow of soil gas whereas `Q` is without meaning (the model returns `Q` = 0). Flux measurements are done over some surface Ω as indicated in the table.

3.5 Finite-volume method

`RnMod3d` is based on a finite-volume (also called control-volume) method. This method is closely related to the finite-difference method. Information about these

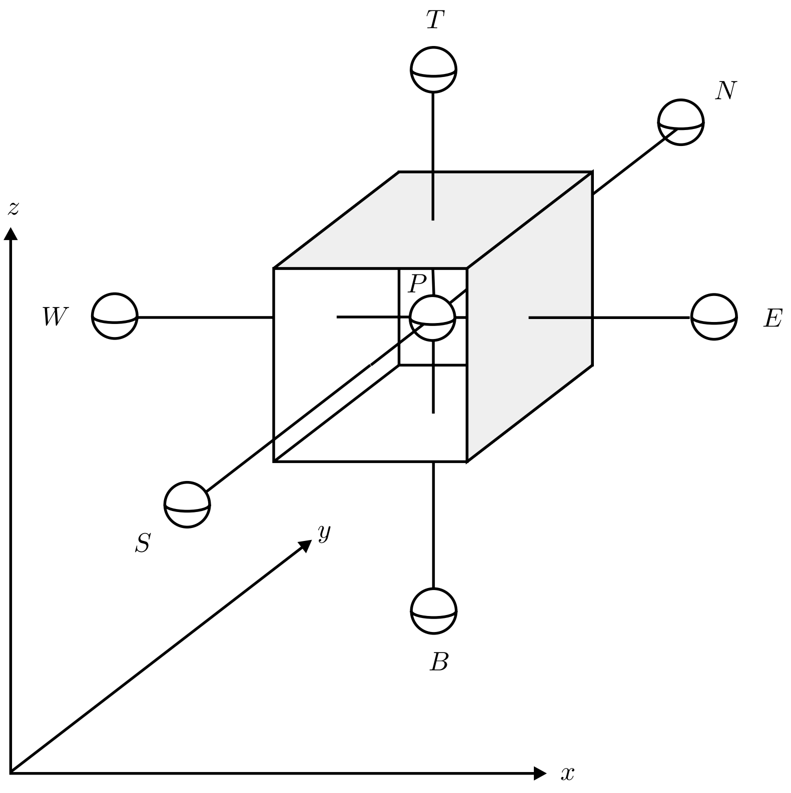


Figure 1. Sketch of the control volume located around node P. The six adjacent nodes are called W for west, E for east, S for south, N for north, B for bottom, and T for top.

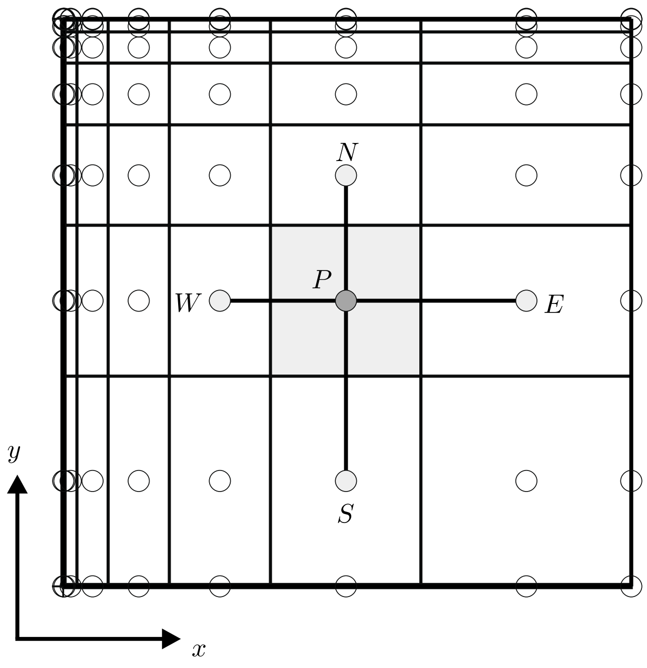


Figure 2. Two-dimensional projection of a grid of control volumes. Observe, that control volumes need not have the same size, and that nodes are always located midway between control-volume interfaces.

numerical techniques can be found in [Pa80, Ve95, He96, Fe99]. The finite-volume approach has been used also in other models of radon transport [Lo87, An92, Sp98].

The computational grid is divided into control volumes as sketched in Figure 1 and 2. Each control volume is a box with one (center) node and six faces. The prime variable is the value of the field at the nodes. Soil-gas problems are based on the disturbance pressure field $p(x, y, z)$ whereas radon problems are based on the radon concentration field in the air-filled parts of the pores $c_a(x, y, z)$. Transport from one control volume to another is approximated by linear flux expressions. These expressions involve field values at pairs of adjacent nodes (e.g. P and E in Figure 1). Fluxes are calculated for each of the six control-volume faces. Considering sources and sinks and that soil-gas and radon may accumulate in the control volume, we then require strict conservation of mass. This gives one algebraic equation for each control volume P:

$$a_P c_P = a_E c_E + a_W c_W + a_N c_N + a_S c_S + a_T c_T + a_B c_B + b \tag{44}$$

where the a's and the b are coefficients, the c's are the unknown field values, and where indices E, W, N, S, T, and B refer to the adjacent control volumes on the east, west, north, south, top, and bottom sides of P. The coefficients are calculated from material properties and control volume sizes.

We do the same thing for all control volumes in the grid and therefore obtain a traditional matrix equation with N equations and N unknown field values. N is typically 10 000 or more, so it is virtually impossible to solve the equation by ordinary matrix inversion (the main matrix would be of size: N by N). `RnMod3d` therefore uses iterative methods for finding the solution.

In summary, `RnMod3d` is based on field values at control-volume *nodes* and fluxes at *interfaces* between pairs of adjacent nodes. Section 5.9 contains a few more details about these matters.

4 Control variables

`RnMod3d` is controlled by more than 60 *control variables*. Some control variables can be assigned simple types of data: strings, floating-point numbers, integers, or booleans (i.e. `true` or `false`). Other variables are enumerated types of data. These are used to help clarify the meaning of variable assignments and to restrict assignments to what is actually meaningful. For example, the control variable `geometry` is declared as an enumerated type, and it can only be assigned the values: `cartesian3D`, `cartesian2D` or `cylindrical2D`. Otherwise an error will occur.

Enumerated types

Some of the control variables are pointers to pre-defined or user-defined procedures and functions. These variables have names ending with `_def`. For example, `grid_def` is a pointer to the user-defined procedure where the grid is defined. If the user has defined a procedure called `mygrid`, then we can tell `RnMod3d` about it with the assignment: `grid_def := mygrid`. Sometimes it makes sense to set such variables to `nil`. For example, during debugging it may be of interest to avoid calling the solver. This can be done with `solver_def:=nil`.

Pointer variables end with `_def`

In the following, all global control variables in `RnMod3d` are presented one by one. The order of presentation follows the likely order of assignment during creation of a job file.

Default values

When `RnMod3d` starts, all control variables are set to their default values. If more runs are conducted within the same job file, it is sometimes desirable to reset all control variables to these values. This can be done with the procedure:

```
  set_control_variables_to_defaults;
```

4.1 runid

Type: String with four characters **Default:** `'0000'` **Description:** Each model calculation is assigned an identification tag called `runid`. If we set `runid := '0997'` and run the model, then standard output goes to the files `f0997LOG.dat` and `f0997RES.dat`. It is a good idea to name that job file `f0997PRG.dpr`, because then all files relating to the run can be found as `f0997*.*`. **Additional information:** See Section 13.1

4.2 runtitle

Type: String **Default:** `'Default control variables'` **Description:** This variable is used to assign a descriptive line of text to the model calculation. For example, `runtitle := 'My first job'`. This line of text is output to the screen and to the `LOG`-file.

4.3 solution

Type: Enumerated variable with two possible assignments: `steady` or `unsteady` **Default:** `steady` **Description:** The assignment: `solution := steady` implies that the model calculation corresponds to steady-state conditions. The assignment: `solution := unsteady` is used for time-dependent problems. **Additional information:** See Section 11.

4.4 geometry

Type: Enumerated variable with three possible assignments: `cartesian3d`, `cartesian2d`, or `cylindrical2d` **Default:** `cartesian3d` **Description:** Selection of coordinate system. **Additional information:** See Section 5.

4.5 Ly

Type: Floating-point number larger than 0 **Default:** `1.0` **Description:** `Ly` gives the thickness of the grid (in m) in the y-direction. The value of `Ly` is only of importance when `geometry := cartesian2d`.

4.6 grid_def

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined procedure with the grid. **Additional information:** See Section 5.

4.7 force_new_grid_in_every_run

Type: Boolean **Default:** `false` **Description:** The assignment `force_new_grid_in_every_run := true` forces a re-calculation of the grid pointed to by `grid_def` every time `run_model` is issued. If the variable is `false`, the grid will only be recalculated if the procedure pointed to by `grid_def` changes from one run to another. For example, it will do so in:

```
...
grid_def:=mygrid;
run_model; (* Run 1 *)
grid_def:=myothergrid;
run_model; (* Run 2 *)
...
```

regardless of the setting of `force_new_grid_in_every_run`. In the following example, it is, however, important that `force_new_grid_in_every_run` is set to `true`:

```
...
procedure mygrid;
begin
set_FixVal(xFix1,0.0);
set_FixVal(xFix2,1.0);
set_axis_single(xFix1,xFix2,NN,FocusA,1.0);
...
end;
...
grid_def:=mygrid;
for NN:=10 to 100 do
  run_model;
...
```

otherwise all model runs will be performed for the grid size corresponding to `NN:=10`.

4.8 `boundary_conditions_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined procedure where the boundary conditions are defined. Other information about nodes and connectors located within the grid boundary can also be specified here. **Additional information:** See Section 6.

4.9 `flux_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined procedure where the "flux measurement probes" (`Flx1`, `Flx2` etc.) are defined. **Additional information:** See Section 8.

4.10 `probe_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined procedure where pressure or radon-concentration probes (`Obs1`, `Obs3` etc.) are defined. **Additional information:** See Section 9.

4.11 `materials_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined procedure where the materials `mat1`, `mat2` etc. are defined. **Additional information:** See Section 7.1.

4.12 `e_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined function where the (total) porosity (ϵ) is defined. **Additional information:** See Section 7.2.

4.13 `beta_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined function where the partition-corrected porosity (β) is defined. **Additional information:** See Section 7.3.

4.14 `G_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined function where the radon generation rate per pore volume (G) is defined. **Additional information:** See Section 7.4.

4.15 `lambda_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined function where the decay constant of radon (λ) is defined. **Additional information:** See Section 7.5.

4.16 `D_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined function where the bulk diffusivity (D) is defined. **Additional information:** See Section 7.6.

4.17 `initialfield_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to a user-defined function. In time-dependent problems, it is possible to specify that the initial field should be given by some function (e.g. equal to a non-zero constant). The variable `initialfield_def` can be used to tell `RnMod3d` about such a function. A nil-assignment: `initialfield_def := nil` is required if the initial field is specified by other means (see `import_initialfield`). **Warning:** Only nodes of the type `free` (see Section 6.1 page 39) will be initialized by `initialfield_def`. For example, nodes that are fixed to certain values (such as boundary conditions) are unaffected by the initial field read through `initialfield_def`.

4.18 `import_initialfield`

Type: Boolean **Default:** `false` **Description:** In time-dependent problems, it is possible to specify that the initial field should be read from a file with: `import_initialfield := true`. The name of the file is given by `import_file_name`. Observe, `import_initialfield` should be set to false after the first time step has been taken.

4.19 `import_finalfield_guess`

Type: Boolean **Default:** `false` **Description:** The solver in `RnMod3d` solves the field equations iteratively until some requirements of convergence are met. If `import_finalfield_guess := false`, then the first "guess" is the field already in the main data structure `GP`. In the very first run in a job file, the field in GP is always 0 everywhere. If `import_finalfield_guess := true`, then the solver imports a field from the file given by `import_file_name` and uses that as an initial guess of the final solution. It is important to distinguish between the initial field for a time-dependent problem and the initial guess for the iterative solution procedure.

4.20 `export_field`

Type: Boolean **Default:** `false` **Description:** If `export_field := true` then the final field is exported to the file given by `export_file_name`. If the file already exists, then it will be overwritten without warning. It is particularly useful to export a "field" if the computations have not converged. The field can then be imported with `import_finalfield_guess := true` and used as a good starting guess for more iterations. If `export_field := false`, then no such field is exported.

4.21 `use_fieldbuffer`

Type: Enumerated variable with three possible assignments: `cBUF1`, `cBUF2` or `no_cBUF` **Default:** `no_cBUF` **Description:** If `use_fieldbuffer` has been set to `cBUF1` then the state of `RnMod3d` in the next `run_model` call will be reset to that stored in the buffer called `cBUF1`. Likewise, the results of the new computations will be stored in the same buffer. Buffers are needed in problems when both the soil gas and the radon problems are time-dependent. If `use_fieldbuffer` has been set to `cBUF2` then the state of `RnMod3d` is encapsulated in the buffer called `cBUF2`. If `use_fieldbuffer` has been set to `no_cBUF`, then no such buffer is used. **Additional information:** Section 11.7.

4.22 `flowfield`

Type: Enumerated variable with three possible assignments: `none`, `export`, `import`, `export_to_qBUF`, or `import_from_qBUF` **Default:** `none` **Description:** The term `flowfield` refers to the flow of soil gas ($\vec{q}$ in Box 1 and 2, page 11). In a calculation of soil-gas transport, the assignment `flowfield := export`, forces the flow field of soil gas between control volumes to be exported to the file given by `flowfield_name`. In a later calculation with advective radon transport, it is possible to import this flow field with `flowfield := import`. The flow field is read from the file given by `flow_field_name`. In a calculation of radon transport with pure diffusion, we set `flowfield := none`. With `flowfield` set to `export_to_qBUF`, the flow field is stored in the flow field buffer called `qBUF`. This is a dynamic variable created for the purpose only when needed. It can be used only within a single job file. With `flowfield` set to `import_from_qBUF`, a flow field already stored in `qBUF` can be restored. An example of the use of `qBUF` can be found in Section 11.7. **Warning:** Observe, that it is meaningful to use flow fields in other calculations only if these calculations are performed with grids identical to that used in the original flow calculation. For example, if the flow of soil gas is calculated with a very fine grid of 20 000 nodes, then this grid cannot be used in a later radon calculation based on a coarser grid with only 5 000 nodes. `RnMod3d` tests if the number of nodes are identical in the two situations. If they are not, an error message will appear. The model does, however, not test if the two grids are truly identical.

4.23 `flowfactor`

Type: Floating-point number **Default:** `1.0` **Description:** During import of a flow field of soil gas, all flows between control volumes are multiplied by the `flowfactor`. For example, imagine a problem with diffusive and advective radon entry into a house depressurized 1 Pa relative to the outdoors. If the soil-gas flow field has been calculated (in a previous model run) with a steady depressurization of 1 Pa, then we use `flowfactor := 1.0` in the radon calculation (no scaling). However, if we want to know the radon entry in the situation of a 5 Pa depressurization, we set `flowfactor := 5.0`. Thereby all flows between control volumes are multiplied by a factor of 5. If the radon entry during house pressurization is needed, we set `flowfactor` to be negative (all flows are reversed). The flow field can be scaled meaningfully in this fashion because of the linearity of the transport equation for the soil-gas flow. Observe, that the procedure is not directly applicable if the house has entry points with different depressurizations etc. `flowfactor` affects both flow fields imported from files and from the buffer called `qBUF`, see Section 4.22.

4.24 import_field_name

Type: String with a valid file name or `''` **Default:** `''` **Description:** This variable is used to specify the name of a file from which a field may be imported (see `initialfield_def` and `import_finalfield_guess`). If `import_field_name = ''` then the import file takes the standard name `fxxxx_00.dat` where `xxxx` is given by the `runid`. For example, if `runid := '0997'` then import will be done from the file `f0997_00.dat`. If the field should be imported from a file called `myfile.dat` then use the assignment: `import_field_name := 'myfile.dat'`.

4.25 export_field_name

Type: String with a valid file name or `''` **Default:** `''` **Description:** This variable is used to specify the name of a file to which the final field should be exported (see `export_field`). If `export_field_name := ''` then the export file takes the standard name `fxxxx_00.dat` where `xxxx` is given by the `runid` (see `import_field_name`).

4.26 flowfield_name

Type: String with a valid file name or `''` **Default:** `''` **Description:** This variable is used to specify the name of a file to which or from which the flow field of soil gas should be imported or exported (see `flowfield`). If `flowfield_name =: ''` then the flowfield file takes the standard name `fxxxxflw.dat` where `xxxx` is given by the `runid` (see `import_field_name`).

4.27 plotfiles_def

Type: Pointer **Default:** `nil` **Description:** This is a pointer to the user-defined function that makes data files for plotting purposes. **Additional information:** See Section 13.3.

4.28 user_procedure_each_iteration_def

Type: Pointer **Default:** `nil` **Description:** This is a pointer to a user-defined procedure that is called once every iteration by the solver. This procedure is called from within the `find_field`-procedure. See Section 14.7. Normally, this variable is set to `nil`. The procedure can be used to monitor the convergence of the iterative solution procedure:

```
procedure monitor_convergence;
begin
writeln('Iteration = ',iter,'c = ',GP[5]^[2]^[5].c);
end;
```

where

```
user_procedure_each_iter_def := monitor_convergence;
```

In principle it is also possible to let the procedure change the boundary conditions. For example, the `cBC[fixed1]` can be changed. The methods described in Section 12 are, however, more suited for that purpose.

4.29 `wr_details`

Type: Boolean **Default:** `false` **Description:** `wr_details := true` forces `RnMod3d` to output detailed information about the progress of the computations. The output goes to the screen during run time. This can be used to debug problematic job files.

4.30 `wr_main_procedure_id`

Type: Boolean **Default:** `false` **Description:** `wr_main_procedure_id := true` forces `RnMod3d` to output identification headers every time the main procedures are called. This can be used to debug problematic job files.

4.31 `wr_all_procedure_id`

Type: Boolean **Default:** `false` **Description:** `wr_all_procedure_id := true` forces `RnMod3d` to output identification headers whenever procedures are called. This can be used to debug problematic job files.

4.32 `wr_iteration_line_log`

Type: Boolean **Default:** `false` **Description:** `wr_iteration_line_log := true` makes `RnMod3d` output information to the `LOG`-file about the number of iteration conducted:

```
Iteration = 501 (1000) Time = 13.02 min (60.00) Residual = 3.40E+0002
```

The meaning is a follows: The line was written after completion of iteration number 501. The number in parentheses shows that a maximum of 1000 iterations is allowed. 13.02 minutes have passed since the `run_model` was issued. The maximum time allowed for the computations is 60.00 minutes. The sum of residuals amounts to $3.4 \cdot 10^2$. The concept of residuals is described page 60. The "iteration line" is output whenever the iteration number divided by `conv_evaluation_period` gives an integer.

4.33 `wr_iteration_line_screen`

Type: Boolean **Default:** `true` **Description:** This variable has the same meaning as `wr_all_procedure_id`, except that the line of information goes to the screen.

4.34 `wr_residual_during_calc_log`

Type: Boolean **Default:** `false` **Description:** `wr_residual_during_calc_log := true` forces `RnMod3d` to write information in the `LOG`-file about residuals. The output comes during the computations, and it is useful for monitoring if the solution converges. The output comes whenever the iteration number divided by `conv_evaluation_period` gives an integer. An example is shown here:

```
* Abs. sum of bs          =  0.00000E+0000
* Abs. sum of residuals   =  2.63340E+0002 (change = -5.31864E-0003)
* Max residual           =  1.02542E+0001 (change =  7.43469E-0003)
* Max residual at (i,j,k) = (   2,   1,   3)
* Max residual at (x,y,z) = ( 7.500E-0001, 0.000E+0000, 2.250E+0000)
```

The quantities involved are defined in Section 10.4. The output includes the sum of absolute values of residuals, the max residual, and the location of the max residual (both as control-volume coordinates (`i,j,k`) and physical coordinates (x, y, z)). The "relative change per iteration" is also given. In the example, the sum of absolute residuals decreases about 0.53 % per iteration. Observe: To obtain information about the number of iterations reached, `wr_iteration_line_log` should be set to `true`.

4.35 wr_residual_during_calc_screen

Type: Boolean **Default:** `false` **Description:** This variable has the same meaning as `wr_residual_during_calc_log`, except that the output goes to the screen.

4.36 wr_flux_during_calc_log

Type: Boolean **Default:** `false` **Description:** If this variable is set to `true`, `RnMod3d` will output results of "flux measurements" with the probes `Flx1, Flx2` etc. The output comes during the computations and can therefore be used to monitor the convergence of the run. The output does not come for every single iteration, unless `conv_evaluation_period := 1`. An example of output is shown here:

```
Flx1  : J =  0.0000000E+0000 ( change =  0.0000000E+0000 ) Q =  0.0000000E+0000
Flx2  : J = -3.3263329E+0000 ( change = -1.3561772E-0002 ) Q =  1.2035000E-0007
Flx3  : J =  0.0000000E+0000 ( change =  0.0000000E+0000 ) Q =  0.0000000E+0000
Flx4  : J =  0.0000000E+0000 ( change =  0.0000000E+0000 ) Q =  0.0000000E+0000
Flx5  : J =  0.0000000E+0000 ( change =  0.0000000E+0000 ) Q =  0.0000000E+0000
```

If this output concerns a radon problem, then the meaning of the numbers is as follows: Flux probe no. 2 (`Flx2`) reports a flux (`J`) of about -3.326 Bq s^{-1}. The relative change per iteration is about -1.35 %. The flow of soil gas (`Q`) is about $1.2 \cdot 10^{-7}$ $m^3 s^{-1}$. Observe: To obtain information about the number of iterations reached, `wr_iteration_line_log` should be set to `true`.

4.37 wr_flux_during_calc_screen

Type: Boolean **Default:** `false` **Description:** This variable has the same meaning as `wr_flux_during_calc_log`, except that the output goes to the screen.

4.38 wr_probes_during_calc_log

Type: Boolean **Default:** `false` **Description:** If this variable is set to `true`, `RnMod3d` will output results of "concentration measurements" with the probes `Obs1, Obs2` etc. The output comes during the computations and can therefore be used to monitor the convergence of the run. The output does not come for every single iteration, unless `conv_evaluation_period := 1`. An example of output is shown here:

```
Obs1  : c =  2.7050633E-0001 ( change =  1.2874671E-0001 )
Obs2  : c =  0.0000000E+0000 ( change =  0.0000000E+0000 )
Obs3  : c =  0.0000000E+0000 ( change =  0.0000000E+0000 )
Obs4  : c =  0.0000000E+0000 ( change =  0.0000000E+0000 )
Obs5  : c =  0.0000000E+0000 ( change =  0.0000000E+0000 )
```

If this output concerns a radon problem, then the meaning of the numbers is as follows: Concentration probe no. 1 (Obs1) reports a field value (c) of about 0.27 Bq m^{-3}. The relative change per iteration is about -12.9 %. Observe: To obtain information about the number of iterations reached, wr_iteration_line_log should be set to true.

4.39 wr_probes_during_calc_screen

Type: Boolean **Default:** false **Description:** This variable has the same meaning as wr_probes_during_calc_log, except that the output goes to the screen.

4.40 wr_final_results_log

Type: Boolean **Default:** true **Description:** If this variable is set to true, output will be written to the LOG-file after the solver has ended its computations. The output includes: (1) the title of the run as specified by runtitle, (2) a statement of why the computations stopped (e.g. because the computations converged), (3) a line showing the number of iterations and time used for the computations (see wr_iteration_line_log), (4) results about residuals (see wr_residual_during_calc_log), (5) results of "flux measurements" (see wr_flux_during_calc_log), and (6) results of "concentration measurements" (see wr_probes_during_calc_log).

4.41 wr_final_results_screen

Type: Boolean **Default:** true **Description:** This variable has the same meaning as wr_final_results_log, except that the output goes to the screen.

4.42 wr_axes

Type: Boolean **Default:** true **Description:** If this variable is set to true, the information about the grid is output to the LOG-file. **Additional information:** See Section 5.10.

4.43 wr_nodes

Type: Boolean **Default:** false **Description:** If this variable is set to true, information about each individual control volume is written to the LOG-file. An example is shown here:

```
wr_nodedata
---------------------------------------------------------------------------
  i   j   k            c nodetyp west   east   south  north  bottom top
---------------------------------------------------------------------------
  1   1   1        0.000 fixed1  nill   noflow nill   noflow nill   noflow
---------------------------------------------------------------------------
```

The output includes: (1) The index coordinates of the control volume (i,j,k), (2) the field value (c) in Bq m^{-3} if a radon problem is solved, (3) the type of node, and (4) the connectors for each of the six faces of the control volume.

4.44 wr_nodes_numbers

Type: Boolean **Default:** true **Description:** If this variable is set to true, output will be written to the LOG-file about the number of each type of control volumes in the grid. For example:

```
wr_count_nodes
* Type and number of nodes incl. boundary conditions :
*        NOP       328
*        free      640
*        fixed1     16 value =  1.00000000000E+0000
*        fixed2     16 value =  0.00000000000E+0000
*        fixed3      0 value =  0.00000000000E+0000
*     unchanged      0
*          Total  1000
```

tells that there are 328 "no operation" control volumes, 640 control volumes that are "free floating" (i.e. controlled by the transport equation), 16 control volumes of the type `fixed1`, and 16 control volumes of type `fixed2`. In total there are 1000 control volumes in the grid. When the computations ended the "fixed values" at `fixed1` and `fixed2` were 1.0 and 0.0, respectively. The item "unchanged" has no meaning here.

4.45 wr_node_sizes

Type: Boolean **Default:** `false` **Description:** If this variable is set to `true`, output will be written to the `LOG`-file about the sizes of each individual control volume in the grid. For example:

```
wr_cvsize
2   5   8 ArW= 3.000E+0000 ArE= 3.000E+0000
          ArS= 3.000E+0000 ArN= 3.000E+0000
          ArB= 2.250E+0000 ArT= 2.250E+0000
          dV= 4.500E+0000
```

tells that the area of the west face (`ArW`) of control volume `(i,j,k) = (2,5,8)` amounts to 3 m^2. The areas of the east, south, north, bottom, and top faces are also given. The volume `dV` of the control volume is 4.5 m^3.

4.46 wr_coefficients

Type: Boolean **Default:** `false` **Description:** If this variable is set to `true`, output will be written to the `LOG`-file about the coefficients of each individual control volume in the grid. For example:

```
wr_all_coefficients
  1   1   1 mat1  ap= 1.000E+0000 b= 1.000E+0000
                  aw= 0.000E+0000 ae= 0.000E+0000
                  as= 0.000E+0000 an= 0.000E+0000
                  ab= 0.000E+0000 at= 0.000E+0000
```

tells that the material of control volume `(i,j,k) = (1,1,1)` is `mat1`, and that the `ap` coefficient amounts to 1.0 etc.

4.47 wr_material_volumes

Type: Boolean **Default:** `true` **Description:** If this variable is set to `true`, output relating to the materials `mat1, mat2` etc. (see Section 7.1) will be written to the `LOG`-file. This information can be particularly useful for testing if the problem has been set up correctly. For example, in 3D simulations with building components of different materials, it is useful to test if the volumes of these components are as intended. For example,

```
wr_material_volumes_etc (volume-averaged field values)
  mat           Avg(conc)           Activity             Volume          N  N_invalid
 mat1   1.197706031E+0004  9.581648252E+0002  4.000000000E-0001         36         89
 mat2   2.638484718E+0004  6.860060266E+0003  1.300000000E+0000         36         89
 mat3   2.221443068E+0004  5.775751977E+0003  1.300000000E+0000         63        162
  mat          Min(conc)   i   j   k           x           y            z
 mat1   2.999521243E+0003   2   4   2  1.852E-0002  9.815E-0001 -2.950E+0000
 mat2   2.434101159E+0004   2   4  10  1.852E-0002  9.815E-0001 -1.500E+0000
 mat3   2.156342193E+0004   2   4  18  1.852E-0002  9.815E-0001 -5.000E-0002
  mat          Max(conc)   i   j   k           x           y            z
 mat1   2.095874291E+0004   2   2   5  1.852E-0002  3.519E-0001 -2.650E+0000
 mat2   2.851119095E+0004   4   2   8  6.481E-0001  3.519E-0001 -2.300E+0000
 mat3   2.300846592E+0004   4   2  11  6.481E-0001  3.519E-0001 -1.100E+0000
Total geometric volume      =  3.00000000000E+0000
Total activity              =  1.35939770688E+0004
Overall mean concentration  =  2.26566284481E+0004
```

gives the following information for the control volumes set to `mat1`: (1) the average air concentration (c_{a}) is 11.97 kBq m^{-3}, (2) the total activity considering all phases ($\sum \beta c_{\mathrm{a}} \delta V$) is 958 Bq, (3) the total geometric volume taken up by `mat1` ($\sum \delta V$) is 0.4 m^3, (4) there are 36 control volumes with valid field values and 89 with invalid field values (invalid field values could be from control volumes of zero volume or `NOP`'s–see Section 6.1 page 39), (5) the minimum concentration is 3.0 kBq m^{-3} and this value occurs at control volume `(i,j,k) = (2,4,2)` with physical coordinates (x,y,z) = (0.019 m, 0.98 m, $-$2.95 m), and (6) the maximum concentration is 21 kBq m^{-3}. Similar information is given for the two other materials: `mat2` and `mat3`.

The item "Total geometric volume" gives the total geometric volume included in the grid without consideration for porosity. The item "Total activity" is the total (mobile) activity covered by the grid regardless of phase ($\sum \beta c_{\mathrm{a}} \delta V$). The "Overall mean concentration" corresponds to the average air concentrations (c_{a}) listed for the individual materials. The only difference is that this value includes results from all materials. Observe: all average air concentrations use the volume of the included control volumes as weight.

4.48 `warning_priority_log`

Type: Enumerated variable with the following possible assignments: `war_interpolation`, `war_other`, `war_fileimport`, `war_convergence`, `war_residual`, or `war_none` **Default:** `war_other` **Description:** It is used to control what type of warnings that should be written to the `LOG`-file: only warnings of sufficient importance will be output. If `warning_priority_log := war_none`, then no warnings whatsoever will be output. If `warning_priority_log := war_other`, then warnings of this type plus those lower on the list (convergence warnings, file import warnings, and residual warnings) will be output. Warnings from the (less important) field interpolation functions will not be output. Normally, `warning_priority_log` will be set to `war_other` because in some circumstances the output can be flooded with warnings from the field interpolation procedure.

4.49 `warning_priority_screen`

Type: Enumerated variable with the following possible assignments: `war_interpolation`, `war_other`, `war_fileimport`, `war_convergence`, `war_residual`, or `war_none` **Default:** `war_other` **Description:** This variable has the same meaning as `warning_priority_log`, except that the output goes to the screen.

4.50 solver_def

Type: Pointer to nil, find_better_field_Thomas, or find_better_field_Gauss_Seidel **Default:** nil **Description:** This is a pointer to the RnMod3d procedure that should be used as solver. (1) nil means that no solver is defined. This is useful during debugging. For example, it can be tested if the grid is set up correctly or if there are memory problems etc. without really solving the problem. (2) find_better_field_Thomas means that the Thomas algorithm is used as solver. This procedure sweeps the grid line by line in alternating directions. (3) find_better_field_Gauss_Seidel means that the Gauss-Seidel procedure is used as solver. This is a point-iterative procedure. The procedure pointed to by solver_def is called once in each iteration. **Additional information:** See Section 10.3.

4.51 scheme

Type: Enumerated variable with the following possible assignments: powerlaw, central, upwind, hybrid, or exact. **Default:** exact **Description:** This variable is used to control how the coefficients are calculated. **Additional information:** See the function Apower in the file R3Main03.pas and [Pa80].

4.52 relax_factor

Type: Positive floating-point number **Default:** 1.0 **Description:** This variable controls the relaxation of the iterative solution procedure:

```
c_new:=c_old+relax_factor*(c_now-c_old)
```

where c_old is the field value reached in the previous iteration, c_now is the result reached in this iteration, and c_new is the new result after relaxation. With relax_factor := 1.0 there is no relaxation. Relaxation factors above 1 can be used to obtain quicker convergence (over-relaxation). Relaxation factors below 1 (under-relaxation) can be used to "tame" unstable problems. Relaxation factors above 2.0 are unstable. Optimal relaxation factors for soil-gas problems are often around 1.98.

4.53 flux_convset

Type: Set of the enumerated values: Flx1, Flx2 etc. **Default:** [ ] (i.e. an empty set) **Description:** The variable specifies which "flux measurement probes" that should be used by the solver for for convergence tests. For example, if flux_convset := [Flx3], then only the convergence of Flx3 is used by the solver. Examples of other assignments are:

Calculation with Pascal sets!

```
flux_convset := [Flx1,Flx2];         (* Flx1 and Flx2 *)
flux_convset := [Flx1,Flx3..Flx5];   (* Flx1, Flx3, Flx4, and Flx5 *)
flux_convset := [Flx1..Flx5]-[Flx2]; (* Flx1, Flx3, Flx4, and Flx5 *)
flux_convset := [];                  (* Empty set, no flux probes  *)
```

Warning: Flux probes with end results close to zero should not be included in flux_convset. **Additional information:** See Section 8.

4.54 probe_convset

Type: Set of the enumerated values: Obs1, Obs2 etc. **Default:** [ ] (i.e. an empty set) **Description:** This variable has the same meaning as flux_convset (see the previous section) except that this one concerns "the probes for concentration measurements": `Obs1`, `Obs2` etc. **Warning:** Probes with end results close to zero should not be included in `probe_convset`. **Additional information:** See Section 9.

4.55 conv_evaluation_period

Type: Integer number **Default:** `50` **Description:** This variable is used to control when convergence tests should be conducted. This is of interest because the solver works iteratively, and because it costs computational time to evaluate if convergence has been reached. In particular, calculation of fluxes `Flx1`, `Flx2` etc. can be somewhat time consuming. If `conv_evaluation_period := 1` then a convergence test is performed after every single iteration. If `conv_evaluation_period := 100` then a convergence test is performed only after every 100 iterations. There is one side effect to the setting of `conv_evaluation_period`: The procedures associated with the variables:

```
wr_iteration_line_log
wr_iteration_line_screen
wr_residual_during_calc_log
wr_residual_during_calc_screen
wr_flux_during_calc_log
wr_flux_during_calc_screen
wr_probes_during_calc_log
wr_probes_during_calc_screen
```

will generate output only for those of the iterations with convergence tests.

4.56 min_iterations

Type: Integer number **Default:** `5` **Description:** This variable sets the minimum number of iterations that the solver should use. For example: `min_iterations:=50` will force the solver to do at least 50 iterations regardless of all other settings.

4.57 max_iterations

Type: Integer number **Default:** `500` **Description:** This variable sets the maximum number of iterations that the solver can use. For example: `max_iterations:=1000` will force the solver to stop after a maximum of 1000 iterations. The iterations may stop before that (e.g. if convergence is reached).

4.58 max_time

Type: Floating-point number **Default:** `180` **Description:** This variable sets the maximum computational time (wall-clock time in seconds) that the solver can use. For example: `max_time:=12*3600` will force the solver to stop after 12 hours of computations. The iterations may stop before that (e.g. if convergence is reached). **Warning:** This variable has no meaning if `RnMod3d` is complied with Borland Pascal v. 7.

4.59 max_change

Type: Floating-point number **Default:** `1e-6` **Description:** This variable sets part of the criteria for "convergence". When all of the probes included in the sets `flux_convset` and `probe_convset` changes less (per iteration) than the value given by `max_change`, we consider this part of the requirement for convergence to have been met (but there are others). Hence `max_change := 1e-4` means that convergence is not reached before the monitored values change by less than 0.01 % per iteration.

4.60 `max_residual_sum`

Type: Floating-point number **Default:** `1e-4` **Description:** In addition to "flux measurements" and "concentration measurements" (see `max_change`), `RnMod3d` also uses the sum of residuals as a criteria for convergence. If the sum of residuals is larger than the value assigned to `max_residual_sum`, the model does not consider the solution to have converged. **Additional information:** See Section 10.4.

4.61 `dtim`

Type: Non-negative floating-point number **Default:** `0.0` **Description:** `dtim` is the time step given in seconds. Hence, in a time-dependent problem (i.e. when `solution` has been set to `unsteady`) the call `run_model` will advance the field of pressures or radon concentrations by `dtim`. For example, if each time step should be 1 hour, then we set `dtim := 3600` **Additional information:** See Section 11.2.

4.62 `BC_running`

Type: Boolean **Default:** `false` **Description:** If this variable is set to `false` then no adjustment of boundary conditions are carried out. Hence this value must be set to `true` when "running boundary conditions" are needed. **Additional information:** See Section 12.

4.63 `BC_running_update_of_cBCs_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to a user-defined procedure that controls how the boundary conditions (e.g. `cBC[fixed1]`) are changed. To prevent unstable solutions the process is normally under-relaxed. **Additional information:** See Section 12.

4.64 `BC_running_min_iterations`

Type: Interger number **Default:** `100` **Description:** This variable is of type integer. It sets the minimum number of iterations that `RnMod3d` needs to carry out before it attempts to change the boundary conditions. If the value is set too low, the solution procedure can become unstable. **Additional information:** See Section 12.

4.65 `BC_running_max_residual_sum_before_new_BC`

Type: Floating-point number **Default:** `1e-9` **Description:** This variable gives the maximum sum-of-residuals before `RnMod3d` attempts to change the boundary conditions. If the value is set too high, the solution procedure can become unstable. **Additional information:** See Section 12.

4.66 `BC_running_convergence_def`

Type: Pointer **Default:** `nil` **Description:** This is a pointer to a user-defined function that returns the value `true` if some user-defined criteria for convergence has been met. Otherwise it should return the value `false`. For example, in a simulation of exhalation from concrete into a chamber it can be tested if there is consistency between the assumed fixed-concentration and the calculated flux. **Additional information:** See Section 12.

4.67 `wr_BC_running_messages_log`

Type: Boolean **Default:** `false` **Description:** This variable that controls if `RnMod3d` should output information about the *running boundary conditions* to the `LOG`-file. **Additional information:** See Section 12.

4.68 `wr_BC_running_messages_screen`

Type: Boolean **Default:** `false` **Description:** This variable controls if `RnMod3d` should output information about *running boundary conditions* to the screen. **Additional information:** See Section 12.

4.69 `press_enter_wanted`

Type: Boolean **Default:** `true` **Description:** If the variable is set to `true`, then the console (window) where `RnMod3d` runs does not close before the user has pressed enter. If the variable is set to `false` `RnMod3d` can be run in batch mode.

5 Geometry

`RnMod3d` uses a grid of control volumes as the basis for all computations. This section tells how the grid spacing is controlled. Essentially, the user needs to write a procedure that maps the computational `(i,j,k)` space onto the physical (x, y, z) world in meters. Only orthogonal grids are possible in `RnMod3d`. This means that control volumes with the same `i`-coordinate have identical x coordinates (regardless of `j` and `k`). The same is true for the other dimensions. Hence the mapping can be done independently for each axis:

$$\begin{aligned} \texttt{i} &\leftrightarrow x \\ \texttt{j} &\leftrightarrow y \\ \texttt{k} &\leftrightarrow z \end{aligned}$$

The mapping involves three steps:

- **Selection of coordinate system** The type of coordinate system is selected with the control variable `geometry`. Three possibilities are available: `cartesian2d`, `cylindrical2d` and `cartesian3d`.
- **Declaration of fix points** All physical dimensions of importance for the problem should be associated formally with *fix points* called `xFix1`, `xFix2` etc. For example, in a house simulation there may be a crack in the floor at $x = 3$ m. We can associate this location with `xFix2` with the call: `set_FixVal(xFix2,3.0)`. In other parts of the job file, reference should be made to this physical location through `xFix2`. Reference directly to $x = 3.0$ m should be avoided.
- **Node spacing** To achieve a sufficiently accurate numerical solution it is important that grid points are closely spaced in regions with large field gradients. In `RnMod3d`, the grid is generated "by hand". Essentially, each axis can

be subdivided as specified by the user. If each of the three axes are divided into 50 pieces, then the grid will consist of $50^3 = 125\,000$ control volumes.

When the grid has been set up in this way, each control volume has a certain location and size. Section 5.9 provides more details about this. However, geometrical information for individual control volumes is normally not needed because of the use of fix points.

An example of a user-defined grid is shown next. The grid is from a three-dimensional problem. The full meaning of the statements is explained in the following.

Example 4 *Three-dimensional grid, where* `geometry:=cartesian3d` .

```
procedure mygrid;
begin
set_FixVal(xFix1,0.0); (* x-axis *)
set_FixVal(xFix2,1.0);
set_axis_single(xFix1,xFix2,10,FocusA,2.0);

set_FixVal(yFix1,0.0); (* y-axis *)
set_FixVal(yFix2,1.0);
set_axis_single(yFix1,yFix2,10,FocusB,2.0);

set_FixVal(zFix1,-3.0); (* z-axis *)
set_FixVal(zFix2,-2.5);
set_FixVal(zFix3,-0.5);
set_FixVal(zFix4, 0.0);
set_axis_single(zfix1,zfix2,1,FocusA,1.0);
set_axis_single(zfix2,zfix3,5,focusA,1.0);
set_axis_single(zfix3,zfix4,1,FocusA,1.0);
end;
```

To force `RnMod3d` to use the grid defined in the example, we need set the control variable called `grid_def` as follows:

```
grid_def := mygrid;
```

5.1 Grid size (memory issues)

The only limitation for the size of the grid is the available computer memory. Grids with as many as 250 000 nodes have been used on a pc with 128 Mb of ram. By default `RnMod3d`, however, is limited to the grid size given by the compiler directives in the file `R3DIRS03.pas`. Typical settings are:

```
{$DEFINE imax100}
{$DEFINE jmax100}
{$DEFINE kmax200}
```

This means that a maximum of 100 nodes can be located on the x-axis (index `i`) and the y-axis (index `j`), whereas 200 nodes are allowed on the z-axis (index `j`). The following `DEFINE`-directives are possible for the x-axis:

```
imax3
imax10
imax50
imax100
imax150
imax200
imax250
imax300
```

```
imax350
imax400
imax450
imax500
```

Similar directives can be used for the other axes. The maximum number of nodes that can be allocated on the x-, y-, and z-axes are given by `imaxTot`, `jmaxTot`, and `kmaxTot`. The values of these variables are output to the LOG-file and the screen when a job is run. In case too many nodes are specified in a job file, the computations will end with an error message.

5.2 `geometry`

The coordinate system used by `RnMod3d` is set by the control variable `geometry`. As shown in Figure 3 there are three possibilities.

The assignment `geometry := cartesian3d` implies that an ordinary cartesian (x, y, z)-coordinate system is used for the computations. With `geometry := cartesian2d`, only the cartesian (x, z)-coordinates are used. The thickness of the grid in the y-direction can be set with the control variable `Ly` (see Section 4.5). The default thickness is 1 meter. With `geometry := cylindrical2d`, the (x, z)-coordinates refer to the (r, z) cylindrical coordinates. In this case, the y coordinate has no meaning. One-dimensional problems are solved with either of the three coordinate systems. To minimize the use of memory, only one node should be devoted to each of the "passive" dimensions.

Grids for 1D-problems

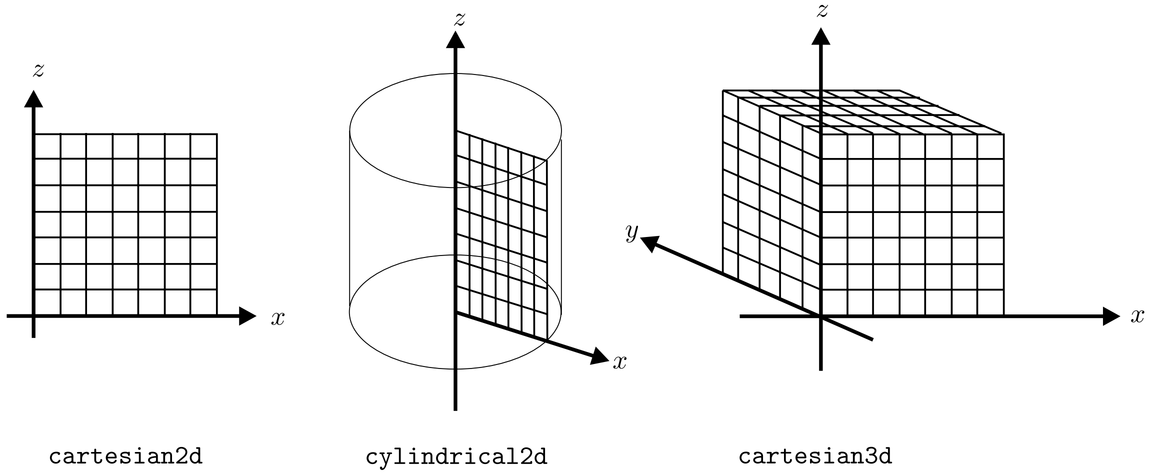


Figure 3. Types of grid geometries that can be selected with the control variable `geometry`. *The idea for this figure comes from [Ho94, p. 30].*

5.3 `set_FixVal`

As already stated, physical dimensions of importance for the problem should be associated with fix points called `xFix1`, `xFix2` etc. for the x-axis, `yFix1`, `yFix2` etc. for the y-axis, and `zFix1`, `zFix2` etc. for the z-axis. Fix points are associated with physical dimensions by calls such as:

```
set_FixVal(xFix2,3.0)
```

that links `xFix2` to the physical dimension $x = 3.0$ m. Fix points should be set up in accordance with the following rules:

- Fix points should be defined starting from `xFix1` for the x-axis, `yFix1` for the y-axis, and `zFix1` for the z-axis.
- The physical dimensions associated with fix points should be given in ascending order: "`xFix1`" < "`xFix2`" < "`xFix3`" etc.
- There can be "no gaps" in the series of fix points used. For example, it is not possible to define `xFix2` and `xFix4` without also defining `xFix3`.

5.4 `wFixVal`

`RnMod3d` stores all information about fix points in the array `wFixVal`. For each fix point, the array contains a record of two items called `defined` and `w`. The first element is a boolean that tells if the fix point has been defined or not. The second element is the physical (meter) coordinate of the fix point. This type of information is useful in some special situations. For example, imagine that a basement slab is set to span vertically from `zFix2` to `zFix3`. To ascertain that this slab thickness has been correctly implemented in the job file, we could make the following call in the job file:

Example 5 *Reference to fix points.*

```
if (wFixVal[zFix2].defined) and (wFixVal[zFix3].defined) then
  writeln('The slab thickness is = ',wFixVal[zFix3].w-wFixVal[zFix2].w,' m');
```

5.5 Node spacing

The grid of control volumes is created by subdividing each of the three axes. In `RnMod3d`, this subdivision is always done *between* single pairs of fix points. Fix points never move (regardless of the grid spacing). This means, for example, that control volumes will never "cross" a fix point. This has the following important implication: If, for example, a concrete slab in a house simulation is defined by reference to fix points then the "concrete" is always filled up completely by control volumes. There will be no control volumes which are partly in the soil and partly in the concrete. To say it differently: All "cuts" are made along fix points. Subdividing an axis between pairs of fix points can be done with the three procedures:

```
  set_axis_single(wFixA,wFixB,hAB,f,pow);
  set_axis_double(wFixA,wFixB,hAM,hMB,fA,fB,powA,powB,wdiv);
  set_axis_triple(wFixA,wFixB,hAM,hMM,hMB,fA,fM,fB,powA,powM,powB,wdivA,wdivB);
```

The fix points "FixA" and "FixB" must be adjacent. Hence, we cannot make a call such as `set_axis_single(xFix2,xFix5...)`. To help read the `set_axis`-procedure headers it is probably useful to know that `w` is used for x, y and z. Likewise `h` is a generic reference to the index variables: `i`, `j` and `k`.

5.6 `set_axis_single`

```
  set_axis_single(wFixA,wFixB,hAB,f,pow);
```

where

`wFixA` and `wFixB` are fix points such as `xFix1` and `xFix2`.

`hAB` is the wanted number of subdivisions between "A" and "B".

`f` is `focusA` or `focusB`.

`pow` is a real number (e.g. 1.0).

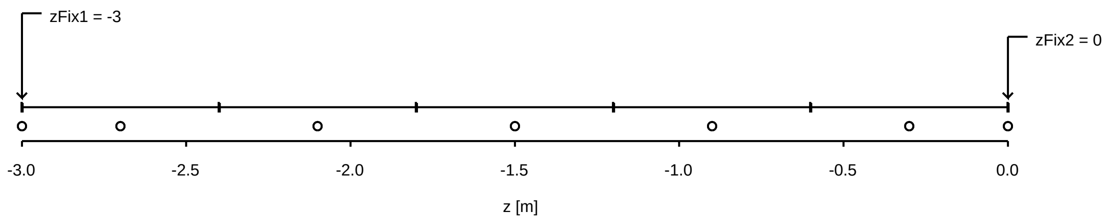


Figure 4. Z-axis generated with: `set_axis_single(zFix1,zFix2,5,FocusA,1.0);` *Five nodes are placed at uniform distances between the fix points at -3 and 0.0.*

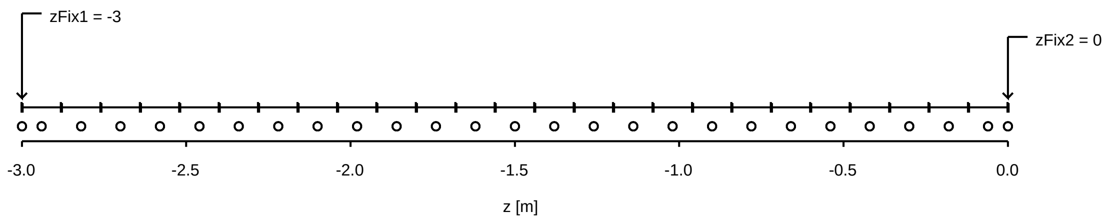


Figure 5. Z-axis generated with: `set_axis_single(zFix1,zFix2,25,FocusA,1.0);` *Now 25 nodes are located between the two fix points.*

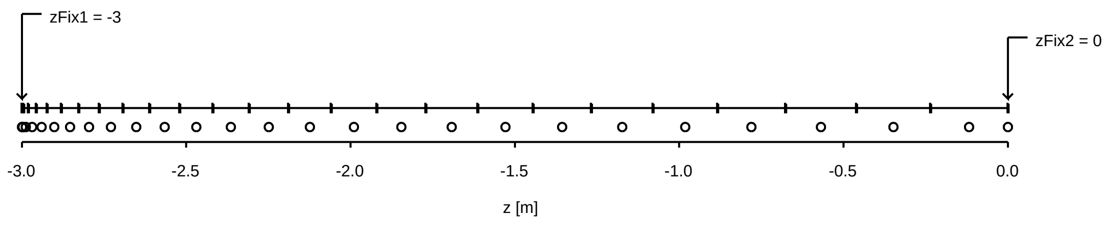


Figure 6. Z-axis generated with: `set_axis_single(zFix1,zFix2,25,FocusA,2.0);` *Now the 25 nodes are not distributed uniformly. The density is highest in the left part of the axis (*`FocusA`*).*

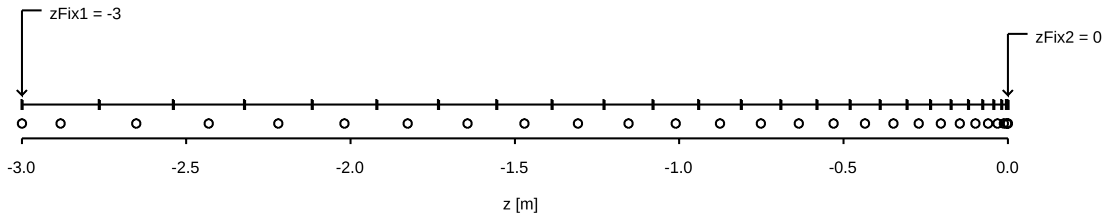


Figure 7. Z-axis generated: `set_axis_single(zFix1,zFix2,25,FocusB,2.0);` *Now the focus is to the right side (*`FocusB`*).*

This procedure divides the axis between the fix point pair: `wFixA` and `wFixB` into `hAB` subdivisions. Power functions of the type w^{pow} are used for the purpose. `pow` equal to 1 makes the node-spacing uniform. If a non-uniform distribution is wanted, we set `pow` to values different from 1. For example, `pow` set to 2, will give a spacing that increases as a square-function. The parameter `f` is used to control where the density of divisions should be highest. If the density should be highest close to fix point A, then set `f=focusA`. If the density should be highest in the other end, use `f=focusB`. The term 'single' in the name of the procedure refers to the fact that the grid point density changes monotonically (i.e. in one single interval) as specified by the distribution function.

Sample calls with `set_axis_single` are given in Figure 4 to 7. The two fix points `zFix1` and `zFix2` are set as follows:

```
set_FixVal(zFix1,-3.0)
set_FixVal(zFix2, 0.0)
```

5.7 set_axis_double

```
set_axis_double(wFixA,wFixB,hAM,hMB,fA,fB,powA,powB,wdiv);
```

where

`wFixA` and `wFixB` are fix points such as `xFix1` and `xFix2`.

`hAM` and `hMB` are the wanted numbers of subdivisions between "A" to "M" and "M" to "B", respectively.

`fA` and `fB` are assigned the values `focusA` or `focusB`.

`powA` and `powB` are real numbers.

`wdiw` is a real number (e.g. 0.5).

In this procedure, the axis between the two fix points (A and B) is split into two. The physical location of the middle point (M) is given by `wdiv`. `wdiv` gives the location of M as a fraction of the total physical distance between A and B. `wdiv=0.5` means that M is half way between A and B. `wdiv=0.01` means that M is located very close to A: The distance A–M is 1 % of the total distance from A to B. `wdiv=0.99` locates M very close to B. The number of subdivisions allocated to cover the interval A to M is specified by `hAM`. Likewise, the number of subdivisions between M and B is `hMB`. Within the two intervals: AM and MB, subdivisions are distributed as described for the `set_axis_single` procedure. Each interval has its own `pow`-parameter. So for example, it is possible to have uniform spacing between A and M and highly non-uniform spacing from M to B.

Sample calls with `set_axis_double` are given in Figure 8 and 9. The two fix points `zFix1` and `zFix2` are set to -3 and 0.0, respectively.

5.8 set_axis_triple

```
set_axis_triple(wFixA,wFixB,hAM,hMM,hMB,fA,fM,fB,powA,powM,powB,wdivA,wdivB);
```

where

`wFixA` and `wFixB` are fix points such as `xFix1` and `xFix2`.

`hAM`, `hMM`, and `hMB` are the wanted numbers of subdivisions between "A" to "M", "M" to "M", and "M" to "B", respectively.

`fA`, `fM`, and `fB` are assigned the values `focusA` or `focusB`.

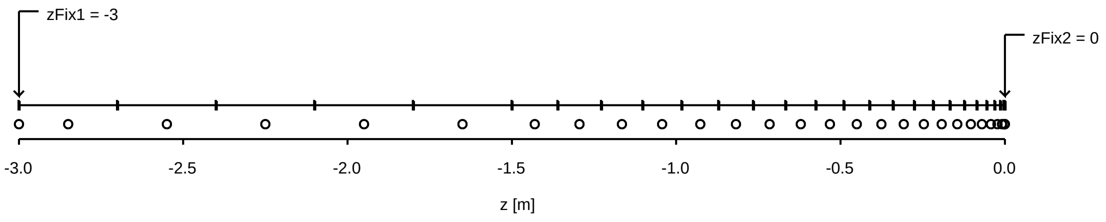


Figure 8. Z-axis generated with: `set_axis_double(zFix1,zFix2,5,20,FocusA, FocusB,1.0,2.0,0.5);` *5 Nodes are devoted to the left interval and 20 to the right. The middle point (that separates left from right) cuts the z-interval into two parts of equal parts (f=0.5). The node distribution is uniform in the left interval (from* -3 *to* -1.5 *m and non-uniform (the exponent equals 2.0) to the right (from* -1.5 *to* 0 *m).*

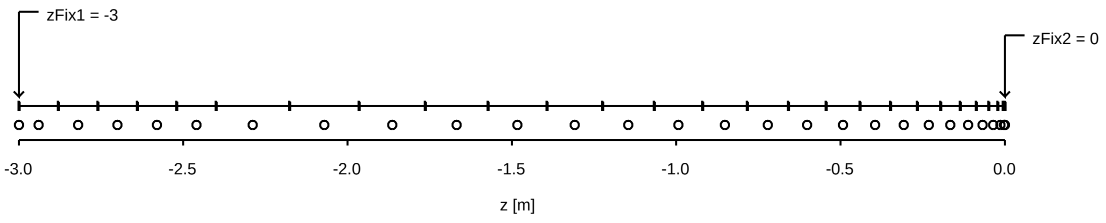


Figure 9. Z-axis generated with: `set_axis_double(zFix1,zFix2,5,20,FocusA, FocusB,1.0,2.0,0.2);` *Now the two intervals are not of equal size. The left interval covers 20 % of the distance between the two fix points. The right interval covers the remaining 80 %.*

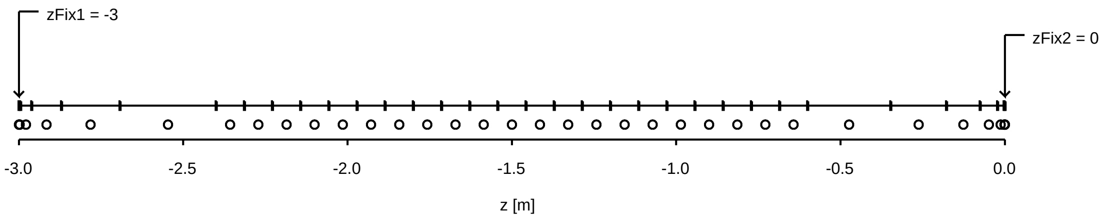


Figure 10. Z-axis generated with: `set_axis_triple(zFix1,zFix2,5,20,5, FocusA,FocusA,FocusB,3.0,1.0,3.0,0.2,0.8);` *The distance between the two fix points is now divided into three. Cuts are made after 20 % and 80 %.*

`powA`, `powM`, and `powB` are real numbers.

`wdiwA` and `wdiwB` are real numbers (e.g. 0.5).

This procedure is a natural extension of `set_axis_double`. The only difference is that now the interval between node A and B is split into three subintervals. A sample call with `set_axis_triple` is given in Figure 10. The two fix points `zFix1` and `zFix2` are set to -3 and 0.0, respectively.

5.9 Location and size of specific control volumes

Figure 11 shows a generic control volume. It is located at `(i,j,k)`. The control *volume* is represented by the gray region. Any material property assigned to the control volume (porosity, diffusivity etc.) applies to that region. Hence, material

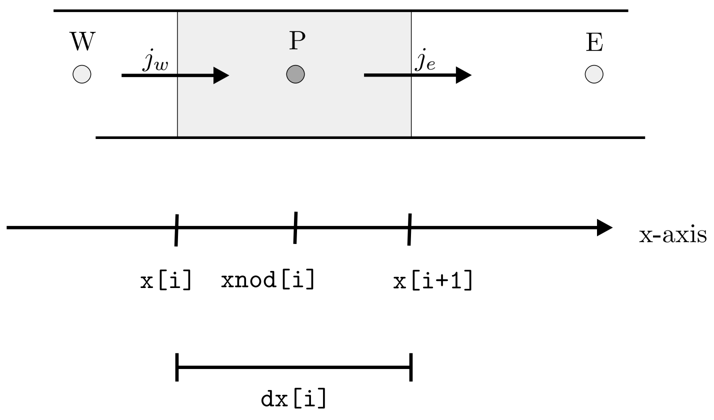


Figure 11. Geometry and x-coordinates for control volume `(i,j,k)`.

properties are assumed to be constant within each control volume. `RnMod3d` finds the field value (e.g. the radon concentration) exactly at the node P in the center of the control volume, and fluxes are calculated exactly at the interfaces between adjacent control volumes (j_e and j_w in the figure).

The x-coordinates for the control volume `(i,j,k)` are shown in Figure 11. The west-side interface is located at `x[i]` and the east-side interface is located at `x[i+1]`. The node P is located at `xnod[i]` midway between the interfaces. Hence `xnod[i]` $=$ `x[i]` $+0.5 \cdot \mathtt{dx}[\mathtt{i}]$. The length of the control volume is `dx[i]` $= \mathtt{x}[\mathtt{i}+1] - \mathtt{x}[\mathtt{i}]$.

In `RnMod3d` the location of control-volume interfaces are stored in the arrays `x[i]`, `y[j]`, and `z[k]`. Likewise the "height, width and depth" are stored in the arrays `dx[i]`, `dy[j]`, and `dz[k]`. Hence such information can be accessed directly by the user as given in the following example. Alternatively, a standard list can be written to the `LOG`-file as shown in Section 5.10.

Example 6 *How to print out the coordinates of a specific control volume.*

```
...
run_model;
writeln('Control volume (5,3,6) has its west   interface at: x = ',x[5]);
writeln('Control volume (5,3,6) has its south  interface at: y = ',y[3]);
writeln('Control volume (5,3,6) has its bottom interface at: z = ',k[6]);
...
```

`xnod(i)`, `ynod(j)` and `znod(k)`

Often it is necessary only to get the (x, y, z) coordinate of the node (not the interfaces). This information is most easily obtained with the functions `xnod(i)`, `ynod(j)` and `znod(k)`. For example, the physical x-coordinate of the center-node of the `i`'th control-volume is `xnod(i)`. An illustrative example of the use of these functions is given page 47.

Areas and volume of a given control volume

As shown in Figure 1 page 13 each control volume is a "box" with six sides. The area of each side are output to the `LOG`-file if the control variable `wr_node_sizes` is set to `true` (see Section 4.45). It is also possible to get the information for just one single control volume. This can be done with the procedure `set_cvsize`. The following example shows what to do.

Example 7 *Application of* `set_cvsize` .

```
procedure wr_interface_areas(i:itype; j:jtype; k:ktype);
var ArW,ArE,ArS,ArN,ArB,ArT,dV:datatype;
begin
set_cvsize(i,j,k,ArW,ArE,ArS,ArN,ArB,ArT,dV);
writeln('The area of the west-side interface of control volume: ');
writeln(i:4,j:4,k:4,' is: ',ArW,' m2');

writeln('The volume of control volume : ');
writeln(i:4,j:4,k:4,' is: ',dV,' m3');
end;
```

5.10 Grid inspection: `wr_axes`

The location of control volumes can be inspected if the control variable `wr_axes` is set to `true`. A table like the one given next will appear in the `LOG`-file (see page 74).

Example 8 *Grid output created with* `mygrid` *in the example page 29.*

```
axis   i       x[i]      x[i+1]       dx[i]        dcdx   dcdxnorm  Fixpts
   x   1    0.00000     0.00000   0.0000000  8.883E-0005 0.00005922   xFix1
   x   2    0.00000     0.01000   0.0100000  4.254E-0004 0.00028357       -
   x   3    0.01000     0.04000   0.0300000  6.003E-0004 0.00040023       -
   x   4    0.04000     0.09000   0.0500000  6.801E-0004 0.00045337       -
   x   5    0.09000     0.16000   0.0700000  7.312E-0004 0.00048749       -
   x   6    0.16000     0.25000   0.0900000  7.477E-0004 0.00049845       -
   x   7    0.25000     0.36000   0.1100000  7.191E-0004 0.00047942       -
   x   8    0.36000     0.49000   0.1300000  6.325E-0004 0.00042165       -
   x   9    0.49000     0.64000   0.1500000  4.695E-0004 0.00031299       -
   x  10    0.64000     0.81000   0.1700000  1.878E-0004 0.00012520       -
   x  11    0.81000     1.00000   0.1900000  2.168E-0019 0.00000000       -
   x  12    1.00000     1.00000   0.0000000  0.000E+0000 0.00000000   xFix2

axis   j       y[j]      y[j+1]       dy[j]        dcdy   dcdynorm  Fixpts
   y   1    0.00000     0.00000   0.0000000  8.181E-0005 0.00005454   yFix1
   y   2    0.00000     0.19000   0.1900000  8.392E-0005 0.00005594       -
   y   3    0.19000     0.36000   0.1700000  3.420E-0004 0.00022799       -
   y   4    0.36000     0.51000   0.1500000  5.414E-0004 0.00036095       -
   y   5    0.51000     0.64000   0.1300000  6.500E-0004 0.00043333       -
   y   6    0.64000     0.75000   0.1100000  6.927E-0004 0.00046177       -
   y   7    0.75000     0.84000   0.0900000  6.845E-0004 0.00045634       -
   y   8    0.84000     0.91000   0.0700000  6.361E-0004 0.00042405       -
   y   9    0.91000     0.96000   0.0500000  5.471E-0004 0.00036470       -
   y  10    0.96000     0.99000   0.0300000  3.096E-0004 0.00020641       -
   y  11    0.99000     1.00000   0.0100000  2.168E-0019 0.00000000       -
   y  12    1.00000     1.00000   0.0000000  0.000E+0000 0.00000000   yFix2

axis   k       z[k]      z[k+1]       dz[k]        dcdz   dcdznorm  Fixpts
   z   1   -3.00000    -3.00000   0.0000000  0.000E+0000 0.00000000   zFix1
   z   2   -3.00000    -2.50000   0.5000000  1.500E+0000 1.00000000       -
   z   3   -2.50000    -2.50000   0.0000000  0.000E+0000 0.00000000   zFix2
   z   4   -2.50000    -2.10000   0.4000000  1.127E-0002 0.00751807       -
   z   5   -2.10000    -1.70000   0.4000000  1.021E-0002 0.00681031       -
   z   6   -1.70000    -1.30000   0.4000000  8.553E-0003 0.00570229       -
   z   7   -1.30000    -0.90000   0.4000000  6.397E-0003 0.00426476       -
   z   8   -0.90000    -0.50000   0.4000000  1.824E-0003 0.00121574       -
   z   9   -0.50000    -0.50000   0.0000000  2.280E-0003 0.00151967   zFix3
   z  10   -0.50000     0.00000   0.5000000  4.337E-0019 0.00000000       -
   z  11    0.00000     0.00000   0.0000000  0.000E+0000 0.00000000   zFix4
```

The example corresponds to the mygrid procedure given page 29. First information about the x-axis is given. Nodes on this axis are indexed by the variable i. There are 12 nodes on the axis, so i goes from 1 to 12. x[i] gives the physical coordinate for the left interface of control volume i. Likewise x[i+1] is the coordinate of the right interface. dx[i] is the thickness of the control volume i (see Figure 11, page 35). The columns dcdx and dcdxnorm will be described in the following section. They contain information about maximum gradients of the field (c) in the direction of the x-axis, so this information can be used to identify where more grid points should be located. The final column Fixpts marks fix-point locations. The same type of information is given for the other two axis. Here, j and k are used as index variables for the y and z-axeses, respectively. Observe that for each fix point, there will be a "ghost" node of zero thickness.

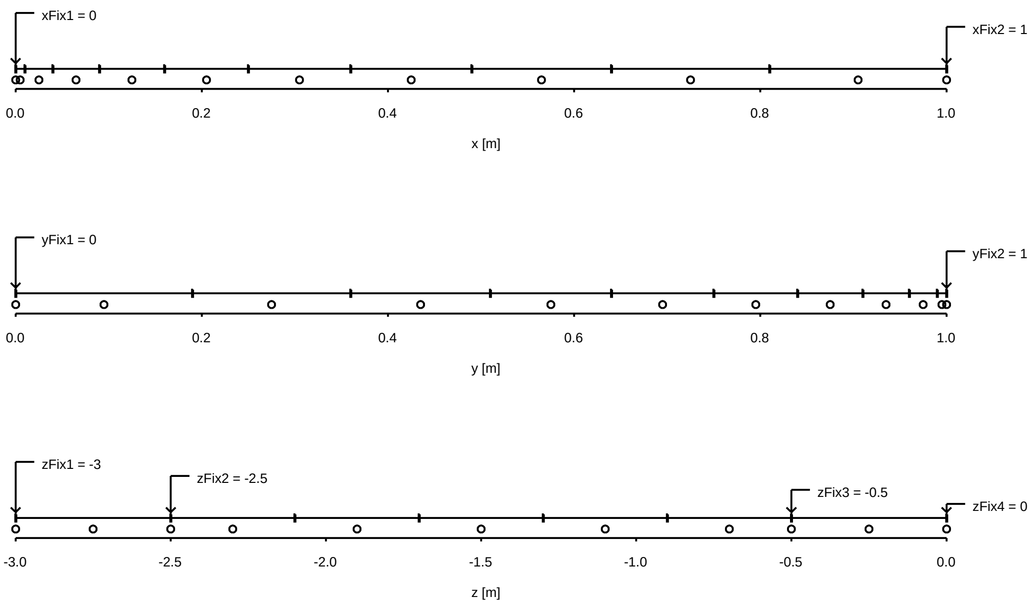


Figure 12. Plot of axes generated with mygrid *defined page 29. The circles represent nodes. Fixpoints are named* xFix1 *etc. Interfaces of control volumes are indicated on the line above the nodes.*

5.11 Grid evaluation: dcdx and dcdxnorm

The output from wr_axes (see the previous section) also indicates where to add more grid points in the grid: For all nodes at i, RnMod3d calculates the field differences ("Δc") to the adjacent nodes at i+1. The largest field difference is called dcdx. In a soil-gas simulation, dcdx is measured in Pa. In a radon simulation, dcdx is measured in Bq m^{-3}. What does this quantity tell? If dcdx is found to be 1 Pa at i=10 whereas dcdx is much smaller than 1 Pa, for all other i's in the grid, then it would probably be good to add some more grid points between i=10 and i=11. Observe, dcdx is the field "gradient" per node in the x-direction.

Similar calculations are carried out for the two other axes, and the results are stored in dcdy and dcdz. To find out which axis that may be in most need of more grid points, the maximum of dcdx, dcdy, and dcdz is calculated. From this global maximum, we normalize all the dcdx-values etc. The results are called dcdxnorm, dcdynorm, and dcdznorm.

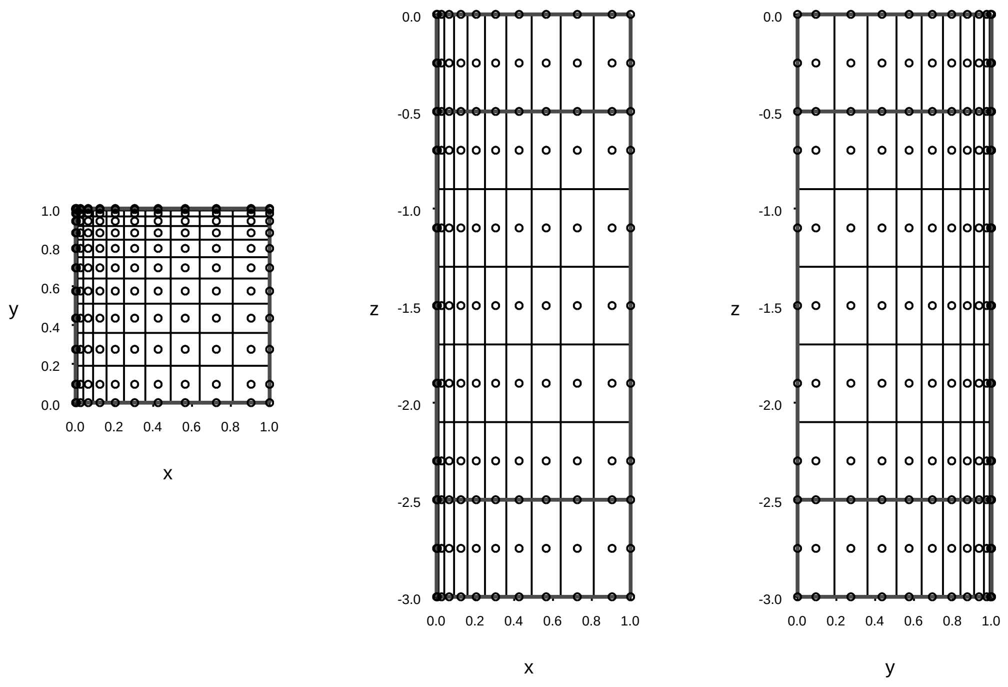


Figure 13. Two-dimensional projection plots of the grid generated with `mygrid` *defined page 29. The circles represent nodes. Fix points are drawn with thick lines. Interfaces of control volumes are drawn with thin lines.*

As shown in the previous section, `dcdx` and `dcdxnorm` etc. are output to the `LOG`-file by setting `wr_axes` to true.

6 Nodes and connectors

Having created a `(i,j,k)`-grid of control volumes and linked them to the physical (x, y, z)-world in meters (see the previous section), it is now time to define how each control volume should "behave" and how each control volume should be connected with its nearest neighbors. This section tells how to do that. Like the geometry of the grid is contained in the procedure pointed to by the control variable `grid_def`, the control variable `boundary_conditions_def`[2] points to the procedure where the "grid behavior" is defined. In other words, this section tells how the `boundary_conditions_def` procedure should be programmed.

The section is divided in two. First, the different types of nodes and connectors are presented, and it is shown how these may be set with the procedures `set_node` and `change_node`. Then we describe the so-called `in`-functions: `in_cube`, `in_plane`, `in_region`, and `in_interval`, which are used to pin point "collections" of control volumes (for example an entire boundary) by reference to fix points (`xFix1`, `xFix2` etc.) defined in the grid procedure. As described in subsequent sections, the `in`-functions are used also for setting up material properties and flux measurements.

[2]The name of this control variable is a little bit misleading. The procedure pointed to by `boundary_conditions_def` controls not only boundary conditions, but all nodes and connectors in the grid.

6.1 Node types

Three types of nodes are used in `RnMod3d`:

the standard node `free`,

the no-operation node `nop`, and

the fixed-value nodes `fixed1`, `fixed2` etc.

Control volumes with a node type set to `free` are controlled by the transport equation for radon or soil gas given in Section 3. Of course, this is normally the majority of control volumes. The no-operation node type, `nop`, is used for control volumes which are not part of the problem (they just happen to be in the grid because the grid is always a regular box). For example, imagine a 3D-simulation of soil-gas entry into a basement house: That part of the grid that is "in the basement" is not controlled by Darcy's law and control volumes in this region should be set to `nop`. The fixed-value node types: `fixed1`, `fixed2` etc. are used for control volumes where the field is held fixed at certain constant values (regardless of transport equations). The fixed values are set up in the array `cBC`. For example, in the previous basement example, the pressure at the interface between concrete and basement (or soil and basement) may be set to some fixed value (e.g. -3.0 Pa). If we set these control volumes to be of node type `fixed1`, then we set `cBC[fixed1] := -3.0`. Likewise, the collection of control volumes located at the interface between soil and atmosphere may be assigned the node type `fixed2`. If this boundary is maintained at zero Pa, then we set `cBC[fixed2] := 0.0`. The control volumes in the soil (or concrete) are of the type `free`.

6.2 Connector types

Each control volume is like a box (see Figure 1, page 13): it represents a certain volume and has six faces. For control volumes (deep) inside the grid, each control volume interfaces with six other control volumes. Radon (or soil gas) therefore normally can flow from one control volume to six others (nearest neighbors). The ability of having transport between neighboring control volumes is handled by *connectors*. All control volumes have six connectors. These are called: `Econ`, `Wcon`, `Ncon`, `Scon`, `Tcon`, and `Bcon` for the east, west, north, south, top, and bottom of the control volume faces, respectively. There are three types of connectors in `RnMod3d`:

the standard connector `std`,

the no-flow connector `noFlow`, and

the connector to "nowhere" called `nill`.

The standard connector type, `std`, is used when radon or soil gas can flow freely (as given by the governing transport equations) between the two control volumes linked by the connector. The no-flow connector called `noFlow` is used when such transport is explicitly set to be zero. This represents a no-flow boundary condition. Clearly, connectors are only meaningful *between* control volumes. Control volumes located at the boundary of the grid will have one or more faces pointing to nowhere. These connectors are set to be of the type called `nill`.

6.3 Default nodes and connectors

When `run_model` is called the first time and the grid geometry has been set in accordance with the procedure pointed to by `grid_def`, `RnMod3d` assigns the following default values to grid nodes and connectors:

All nodes are set to be of type `free`.

Connectors *between* control volumes are set to be of type `std`.

Connectors at the boundary of the grid (pointing to nowhere) are set to `nill`.

This means that the default computational grid behaves as a closed box. For example, if we add some radon activity, it cannot leave the box: All boundaries are closed off. However, radon can move around inside the box and decay as specified by the radon transport equation.

If we make a list of nodes in the grid (e.g. by setting the control variable `wr_nodes` to `true`) it will be found that the above description is not entirely true. A glance at such a table will reveal that (all) grids contain a number of `nop` nodes and `noFlow` connectors. These occur for control volumes that have all three coordinates at fix points (for example, the x-coordinate could be at `xFix3`, the y-coordinate could be at `yFix2`, and the z-coordinate could be at `zFix5`). Such control volumes have zero volume, and we refer to them as being "dead". The "dead" control volumes plays absolutely no role for the user when a problem is set up. They can be ignored completely[3]. Only, they may cause a little confusion in the situation (already mentioned) where the node types in the grid are inspected: The user may think that something is wrong with the grid because it contains a bunch of `nop`'s that he or she had not explicitly defined.

The type of nodes and connectors should be changed from the default with the procedures `set_node` and `change_node`. These procedures are "clever" in the sense that if for example the connector at the top face of control volume `(i,j,k)` is set to `noFlow`, then the program automatically updates the connector of the bottom face of the control volume `(i,j,k+1)`. Also, these procedures take care of the "dead" control volumes.

6.4 Inspection of nodes and connectors

Often it is useful to be able to verify that the correct nodes and connectors are set up. An easy way to inspect the grid is to set the control variable `wr_nodes` to `true`. Then a complete listing of alle nodes and connectors are output to the `LOG`-file (see Section 4.43). Another way is to interact with the main data structure `GP` directly (see Section 14). A list of specific nodes or connectors can be made as shown in the following examples.

Example 9 *Print list of control volumes with specific nodes. The procedure can be called as* `wr_nodelist(NOP)`*.*

```
procedure wr_nodelist(what:nodetyptype);
var i:itype;
    j:jtype;
    k:ktype;
begin
for i:=1 to imax do
  for j:=1 to jmax do
    for k:=1 to kmax do
      if GP[i]^[j]^[k].nodetyp=what then
        writeln(i,' ',j,' ',k,' Found one node = ',nodetyp_string(what))
end;
```

Example 10 *Print list of control columes with specific connectors. The procedure can be called as:* `wr_connectorlist(noFlow)`*.*

[3]Well, they can almost be ignored completely. If the user makes direct access to the field values of the grid make sure to test if the grid values are valid. See the examples in Section 9 for how to do that.

```
procedure wr_connectorlist(what:nodecontype);
var i:itype;
    j:jtype;
    k:ktype;
begin
for i:=1 to imax do
  for j:=1 to jmax do
    for k:=1 to kmax do
      if GP[i]^[j]^[k].Wcon=what then
        writeln(i,' ',j,' ',k,' Found one connector = ',nodecon_string(what))
end;
```

6.5 set_node

The procedure `set_node` is used to set the node type of a single control volume. For example, if the control volume `(i,j,k)` should be set to type `fixed2`, then we simply make the call:

```
set_node(i,j,k,fixed2)
```

The connectors of the control volume remain unaffected by `set_node` unless the node type is set to `nop`. In that case, all connectors to other control volumes are set to `noFlow`.

6.6 change_node

The procedure `change_node` can be used to change any feature of nodes and connectors. The procedure is called as follows:

```
change_node(i,j,k,nodetypNew,
                  wconNew,econNew,
                  sconNew,nconNew,
                  bconNew,tconNew)
```

where `(i,j,k)` are the index coordinates of the control volume to be affected, `nodetypNew` is the new node type that should be assigned to the control volume, and `wconNew` is the new connector type that should be assigned to the west face of the control volume. Similarly, the other parameters concern connectors at the east, south, north, bottom, and top faces of the control volume. Hence, the following call sets the node type of control volume `(i,j,k)` to `free` and all connectors (except the one at the top) to `std`. The top connector is set to `noFlow`:

```
change_node(i,j,k, free, std,std, std,std, std,noFlow)
```

`nodX` *and* `conX`

Often, we want to change only the node type (or a single connector) and leave everything else unchanged. To do that, we use `nodX` for "unchanged node type" and `conX` for "unchanged connector". Hence, if we want to set the top connector of control volume `(i,j,k)` to `noFlow` and leave everything else unchanged we use the call:

```
change_node(i,j,k, nodX, conX,conX, conX,conX, conX,noFlow);
```

6.7 boundary_conditions_def

All node types and connectors (deep inside the grid or on the true boundary) can be set by the procedure pointed to by the control variable: `boundary_conditions_def`. In the simple situation where we want to use the default settings (i.e. to model transport in a closed box as explained previously) and nothing more, we "programme" a procedure with no changes of nodes or connectors:

Example 11 *Default equations and boundary conditions.*

```
procedure my_closed_box(i:itype;j:jtype;k:ktype);
begin
end;
```

With the assignment:

```
  boundary_conditions_def := my_closed_box;
```

`RnMod3d` is told that this is the procedure with all our changes.

Normally, changes of nodes and connectors are not really made for individual control volumes. In the typical situation we make changes for "collections" of control volumes. One example of such a "collection" for a house simulation is the collection of nodes that are located at the atmospheric soil-air interface. In a soil-gas calculation we may want to set the pressure to zero with a `fixed` node type. To pinpoint such "collections" of control volumes by reference to fix points (`xFix1`, `xFix2` etc.), special `in`-functions have been developed. This means that the user need not (explicitly) know the index coordinates `(i,j,k)` of the control volumes in the "collection". The `in`-functions are: `in_cube`, `in_plane`, `in_region`, and `in_interval`. These functions are look-up tables: It can be tested if any control volume `(i,j,k)` is within, outside or at the "edge" of a certain region defined by fix points (`xFix1 xFix2` etc.). The functions return the value `true`, if the control volume belongs to the region, and `false` if it does not. Before describing how the `in`-functions are used, we will present a simple example.

Imagine, a 30 m high column of sand with cross-sectional area of 2 x 2 m. The sand is placed in some container with walls impermeable to gas flow. The top of the container is maintained at -3 Pa relative to the bottom. To treat this problem we define the following two procedures:

Example 12 *Sand column example.*

```
procedure grid_column;
begin
set_FixVal(xFix1,0.0);  (* x-axis *)
set_FixVal(xFix2,2.0);
set_axis_single(xFix1,xFix2,5,Focus,1.0);
set_FixVal(yFix1,0.0);  (* y-axis *)
set_FixVal(yFix2,2.0);
set_axis_single(yFix1,yFix2,5,Focus,1.0);
set_FixVal(zFix1, 0.0); (* z-axis *)
set_FixVal(zFix2,30.0);
set_axis_single(zFix1,zFix2,10,Focus,1.0);
end;

procedure BC_column(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=0;
cBC[fixed2]:=-3.0;
if in_plane([eqAB,inside],i,xFix1,xFix2,yFix1,yFix2,zFix1,zFix1) then
  set_node(i,j,k,fixed1); (* Boundary at zFix1 *)
if in_plane([eqAB,inside],i,xFix1,xFix2,yFix1,yFix2,zFix2,zFix2) then
  set_node(i,j,k,fixed2); (* Boundary at zFix2 *)
end;
```

and we make the assignments:

```
  geometry                := cartesian3D;
  grid_def                := grid_column;
  boundary_conditions_def := BC_column;
```

Observe the following: (1) control-volume sizes etc. are in `grid_column` and control volume "behaviors" are as defined by default with the changes given in `BC_column`. (2) All geometrical features are assigned to fix points (column height and cross-sectional dimensions). The procedure `BC_column` contains references only to fix points. Hence, `BC_column` remains valid even if we (later) change the column dimensions or if we add more control volumes to the grid (i.e. if we make a finer grid).

6.8 in_cube

The boolean function `in_cube` is called as:

```
in_cube(reg, i,xFixA,xFixB, j,yFixA,yFixB, k,zFixA,zFixB)
```

where

`reg` is a Pascal set of `inside`, `outside`, and `eqAB`.

`i`, `j` and `k` are index coordinates of control volumes.

`xFixA` and `xFixB` are adjacent fix points on the x-axis.

`yFixA` and `yFixB` are adjacent fix points on the y-axis.

`zFixA` and `zFixB` are adjacent fix points on the z-axis.

The function concerns the location of the control volume with index coordinates `(i,j,k)` in relation to the "cubic"[4] region defined by the six planes defined symbolically by `x = xFixA`, `x = xFixB`, `y = yFixA` etc. where `x = xFixA` is the plane of all control volumes with physical x-coordinates equal to the fix point `xFixA` etc. If the control volume `(i,j,k)` belongs to the region, then `in_cube` returns the value `true`. Otherwise it returns the value `false`.

The parameter `reg` is a Pascal set of the elements: `inside`, `outside`, and `eqAB`[5]. The meaning will become clear after the example given next. An example call is:

Example 13 *Test procedure for the function* `in_cube`.

```
procedure test;
var i:itype;
    j:jtype;
    k:ktype;
begin
for i:=1 to imax do
  for j:=1 to jmax do
    for k:=1 to kmax do
      if in_cube([inside],
                 i,xFix1,xFix4,
                 j,yFix3,yFix5,
                 k,zFix1,zFix2) then
                   writeln('Inside the cube : ',i:4,j:4,k:4);
end;
```

This example prints a list of all control volumes that have physical coordinates within the cube given by `(xFix1 < x < xFix4) and (yFix3 < y < yFix5) and (zFix1 < z < zFix2)`. To refer to the control volumes that are not inside the cube, `[inside]` should be substituted with `[outside]`. To refer to those control volumes exactly on the faces of the cube, `[inside]` should be substituted with `[eqAB]`. To refer to those control volumes, that are inside or on the face of the

[4]The name "cube" is misleading in the sense that the sides of the region need not be of equal size. It had probably been better to call the function: `in_box`.

[5]For the other `in`-functions the values `eqA` and `eqB` can also be used.

cube, write `[inside,eqAB]`. To refer to those control volumes, that are inside or outside (but not on the face of the cube), write `[inside,outside]`. To refer to all control volumes, write `[inside,outside,eqAB]`. To specify a an empty region use `[ ]`,

6.9 in_plane

The boolean function `in_plane` is called as:

```
in_plane(reg, i,xFixA,xFixB, j,yFixA,yFixB, k,zFixA,zFixB)
```

where

`reg` is a Pascal set of `inside`, `outside` and `eqAB`.

`i`, `j` and `k` are index coordinates of control volumes.

`xFixA` and `xFixB` are adjacent fix points on the x-axis.

`yFixA` and `yFixB` are adjacent fix points on the y-axis.

`zFixA` and `zFixB` are adjacent fix points on the z-axis.

The function concerns the location of the control volume with index coordinates `(i,j,k)` in relation to the plane defined by whichever (single) pair of fix points that are identical. If `xFixA=xFixB`, then the function concerns the plane of control volumes with physical x-coordinates equal to the fix point `xFixA`. Planes in the other directions can be specified by `yFixA=yFixB` or `zFixA=zFixB`. Only one pair of fix points can be identical.

If `xFixA=xFixB`, then the fix points for the y- and z-axis are used to limit the region to be some rectangular part of the plane. An example will be given below. The parameter `reg` is a Pascal set of the elements: `inside`, `outside`, and `eqAB`. The meaning is identical to that described for the function `in_cube`.

The following example shows three sample calls of `in_plane`. The first case concerns the region defined symbolically as: `(x = xFix1) and (yFix3 < y < yFix5) and (zFix1 < z < zFix2)`. So this is a rectangle in the yz-plane through `x = xFix1`. The function returns `true` if `(i,j,k)` is inside the region.

Example 14 *Sample calls of* `in_plane`.

```
if in_plane([inside]     ,i,xFix1,xFix1, j,yFix3,yFix5, k,zFix1,zFix2) then
  writeln('A ',i:4,j:4,k:4); (* case A *)
if in_plane([eqAB]       ,i,xFix1,xFix2, j,yFix2,yFix2, k,zFix1,zFix2) then
  writeln('B ',i:4,j:4,k:4); (* case B *)
if in_plane([inside,eqAB],i,xFix1,xFix1, j,yFix3,yFix5, k,zFix3,zFix3) then
  writeln('C ',i:4,j:4,k:4); (* case C *)
```

6.10 in_region

The boolean function `in_region` is called as:

```
in_region(i,xFixA,xFixB,xreg, j,yFixA,yFixB,yreg, k,zFixA,zFixB,zreg)
```

where

`xreg`, `yreg`, and `zreg` are Pascal sets of `inside`, `outside`, `eqA`, `eqB` and `eqAB`.

`i`, `j` and `k` are index coordinates of control volumes.

`xFixA` and `xFixB` are adjacent fix points on the x-axis.

`yFixA` and `yFixB` are adjacent fix points on the y-axis.

zFixA and zFixB are adjacent fix points on the z-axis.

This is a more general function than in_cube and in_plane. Six planes are defined: two yz-planes at x = xFixA and x = xFixB, two xz-planes at y = yFixA and y = yFixB, and two xy-planes at z = zFixA and z = zFixB. For *each* pair of planes, it can be specified if it is the region inside, outside etc. that is of interest. The function will return true if the control volume (i,j,k) is within the "x-region" and the "y-region" and the "z-region". Otherwise, it will return the value false. The parameters xreg, yreg, and zreg are sets of the elements: inside, outside, eqA, eqB, eqAB. The meaning of the elements inside, outside, and eqAB is identical to that described for the function in_cube. The values eqA and eqB can be used to include only fix point A or B. This is demonstrated by the following sample call. Here, the in_region is true for all control volumes that fulfill: (xFix2 $\leq$ x < xFix3) and (yFix2 < y < yFix3) and (zFix2 $\leq$ z $\leq$ zFix3).

Example 15 *Sample call of* in_region.

```
if in_region(i,xFix2,xFix3,[inside,eqA],
             j,yFix2,yFix3,[inside],
             k,zFix2,zFix3,[inside,eqA,eqB]) then writeln(i:4,j:4,k:4);
```

6.11 in_interval

The boolean function in_interval is called as:

in_interval(h,wFixA,wFixB,wreg):boolean;

h is an index coordinate (i, j or k) of a control volume.

wFixA and wFixB is a pair of adjacent fix points on the x-, y- or the z-axis.

wreg, yreg, and zreg is a Pascal set of inside, outside, eqA, eqB and eqAB.

This function concerns only one coordinate. h is a generic index variable i, j, and k. Likewise, w is a generic physical coordinate x, y, and z. The parameter wreg is a Pascal set of the elements inside, outside, eqA, eqB, or eqAB. The meaning of these elements is identical to that described for the function in_region. The sample call shown next, concerns all control volumes that have x-coordinates in the interval: xFix2 $\leq$ x < xFix3.

Example 16 *Sample call of* in_interval.

```
if in_interval(i,xFix2,xFix3,[inside,eqA]) then writeln(i:4,j:4,k:4);
```

7 Materials

RnMod3d solves transport equations of the *form* given in Box 1, page 11. These equations involve five material properties: β, ϵ, G, D and λ. The physical interpretation of the coefficients is clear in the case of radon transport. When problems of soil-gas transport are considered, the same "coefficients" are used, however, with a different physical interpretation. This is discussed in Section 3.4.

This section outlines how RnMod3d is linked to user-defined functions of material properties through the control variables: beta_def, e_def, G_def, D_def and lambda_def. The technique is flexible as it allows material properties to change in space and time.

To ease the assignment of material properties, it is useful to divide the computational grid into different types of materials. RnMod3d has a tool for that. This is described next.

7.1 `materials_def` (`mat1`, `mat2` etc.)

Most computations involve materials of different types. For example, calculations of entry into houses almost always involve concrete, fill and undisturbed soil. In `RnMod3d`, each control volume can be set to a given material. These materials are named: `mat1`, `mat2` etc. The assignment of control volumes to materials takes place through the user-defined procedure pointed to by `materials_def`. The simplest example is if all control volumes are set to be of the same material:

Example 17 *Homogeneous problem.*

```
function mymaterials(i:itype;j:jtype;k:ktype):mattype;
begin
mymaterials:=mat2;
end;
```

where the control variable `materials_def` must be set to `mymaterials`. An example involving four materials is given below. The materials `mat1`, `mat2`, `mat3` and `mat4` could be layers of soil in a laboratory column experiment. The `in`-procedure described in Section 6 are useful for the task.

Example 18 *Inhomogeneous problem.*

```
function mymaterials(i:itype;j:jtype;k:ktype):mattype;
var mat:mattype;
begin
mat:=mat1;
if in_cube([inside,eqAB],i,xFix1,xFix2,j,yFix1,yFix2,k,zFix1,zFix2) then
  mat:=mat2;
if in_cube([inside,eqAB],i,xFix1,xFix2,j,yFix1,yFix2,k,zFix2,zFix3) then
  mat:=mat3;
if in_cube([inside,eqAB],i,xFix1,xFix2,j,yFix1,yFix2,k,zFix3,zFix4) then
  mat:=mat4;
mymaterials:=mat;
end;
```

A simple way to verify that the geometrical extension of the involved materials has been defined correctly, is to set the control variable `wr_material_volume` to `true`. This will make `RnMod3d` output a list of the total volume occupied by each of the defined materials (see page 23). Other material-specific information is also output.

It should be observed that the use of materials `mat1`, `mat2` etc. is just a "bookkeeping tool". As will be described in the following, this tool is useful when material properties are defined, however, it is perfectly all right not to use the tool. This can be done by setting all control volumes to be of the same type (e.g. `mat1`).

7.2 Porosity, `e_def`

The control variable `e_def` has to point to the user-defined procedure where the porosity is defined. The following example shows how to set all control volumes to have a porosity equal to 0.5:

Example 19 *Homogeneous porosity.*

```
function e_test(i:itype;j:jtype;k:ktype):datatype;
begin
e_test:=0.5;
end;
```

where `e_def` must be set to `e_test`. In principle we can assign an individual porosity for each control volume `(i,j,k)`:

Example 20 *Porosity specified by index variables.*

```
function e_test(i:itype;j:jtype;k:ktype):datatype;
var ee:datatype;
begin
ee:=0.5;
if i=10 then ee:=0.3;
if j=5 the ee:=0.4;
if znod(k)>5.33 then ee:=0.2;
e_test:=ee;
end;
```

If the grid has been split into four materials called `mat1`, `mat2`, `mat3` and `mat4`, with porosities 0.5, 0.4, 0.3 and 0.3, respectively, we would define `e_test` as follows:

Example 21 *Blockwise (in)homogeneous porosity.*

```
function e_test(i:itype;j:jtype;k:ktype):datatype;
var ee:datatype;
begin
case materials_def(i,j,k) of
  mat1: ee:=0.5;
  mat2: ee:=0.4;
  mat3: ee:=0.3;
  mat4: ee:=0.3
else
  error_std('e_test','Unknown material');
end; (* case *)
e_test:=ee;
end;
```

Observe, the following: (1) the function pointed to by the control variable `materials_def` is used to look up the type of material assigned to each individual control volume. (2) It was said that only material `mat1`, `mat2`, `mat3` and `mat4` were used in the application. Hence, porosities are defined only for these materials. If `materials_def` returns some other material, an error has occurred. We therefore stop the computations by calling `error_std`. The use of error procedures is described in Section 13.6. (3) Imagine that `mat2` represents the soil layer from the atmospheric surface down to 0.3 m. If we at some point discover, that in fact this soil layer goes down to a depth of only 0.2 m, then we make changes only in the "materials" function. The porosity of `mat2` is unchanged.

In the examples above, each individual material were assumed to be homogeneous. Material properties can, however, easily be non-constant. A simple example, is if the porosity of `mat3` changes with depth as described in the example page **??**:

Example 22 *Depth dependent porosity.*

```
function e_test(i:itype;j:jtype;k:ktype):datatype;
var ee:datatype;
begin
case materials_def(i,j,k) of
  mat1: ee:=0.5;
  mat2: ee:=0.4;
  mat3: ee:=0.50+0.125*znod(k);
  mat4: ee:=0.3
else
  error_std('e_test','Unknown material');
end; (* case *)
e_test:=ee;
end;
```

The function `znod(k)` is used to get the physical z-coordinate (in meters) of control volume `(i,j,k)` (see Section 5.9).

In some cases, the porosity has been measured in the field for a number of depths (e.g. from 0 to 3 m at 10 cm intervals). Such results can easily be used by `RnMod3d` in the following way: First, the data are gathered in a file. Then a Pascal function is written that can read the file and perform (e.g. linear) interpolation between measurement points. Here we imagine a function called `look_up_etot(depth)`. It simply returns the estimated porosity at any given depth within the measurement interval. Finally, the function is used by the porosity procedure in `RnMod3d`:

Example 23 *Depth dependent porosity read from a file.*

```
function e_test(i:itype;j:jtype;k:ktype):datatype;
var depth:datatype;
begin
depth:=-znod(k);
e_test:=look_up_etot(depth);
end;
```

Parameters can also change in time. See Section 11.5.

7.3 Partition-corrected porosity, `beta_def`

The control variable `beta_def` links `RnMod3d` to the user-defined function where the partition-corrected porosity β is defined. For example, consider a homogeneous medium with air porosity (ϵ_{a}) equal to 0.2, water porosity (ϵ_{w}) equal to 0.2, and a partition coefficient L equal to 0.36, we would write:

Example 24 *Homogeneous β.*

```
function my_beta(i:itype;j:jtype;k:ktype):datatype;
var ea,ew,L:datatype;
begin
ea:=0.2;
ew:=0.2;
L:=0.36;
my_beta:=ea+L*ew;
end;
```

and set `beta_def` to `my_beta`.

The physical meaning of the `beta_def`-procedure is different in radon problems and in problems of soil-gas transport. This is discussed Section 3.4.

7.4 Generation rate, `G_def`

The gereration rate of radon per pore volume is defined by the function pointed to by `G_def`. For example, consider a homogeneous medium with generation rate equal to 0.209838 $\mathrm{Bq\,s^{-1}}$ per $\mathrm{m^3}$. In dry soil this gives a deep-soil radon concentration equal to $G/\lambda \approx 100$ $\mathrm{kBq\,m^{-3}}$. In that case we would write:

Example 25 *Homogeneous G.*

```
function my_G(i:itype;j:jtype;k:ktype):datatype;
begin
my_G:=0.209838;
end;
```

and set `G_def` to `my_G`.

The physical meaning of the `G_def`-procedure is different in radon problems and in problems of soil-gas transport. This is discussed in Section 3.4.

7.5 Decay constant, `lambda_def`

The decay constant of radon is defined by the function pointed to by `lambda_def`. Normally, the decay constant is set to the same value in all parts of the computational plane:

Example 26 *Decay constant λ.*

```
function my_lambda(i:itype;j:jtype;k:ktype):datatype;
begin
my_lambda:=2.09838e-6;
end;
```

and set `lambda_def` to `my_lambda`. If the soil-gas pollutant in question is not radon, but some trace chemical being removed from the soil by a first-order process, "λ" could indeed change from place to place. Ventilation can also be lumped into "λ".

The physical meaning of the `lambda_def`-procedure is different in radon problems and in problems of soil-gas transport. This is discussed in Section 3.4.

7.6 Diffusivity, `D_def`

The bulk diffusivity of radon is defined by the function pointed to by `D_def`. The technique is identical to that described for β, ϵ, G, and λ. Only one thing is different: Diffusitivity may be anisotropic. The header of the `D_def`-function therefore includes a directional parameter. We return to this shortly. In the situation with homogeneous isotropic soil and a bulk diffusivity equal to 10^{-6} m^2 s^{-1}, we define:

Example 27 *Homogeneous isotropic D.*

```
function my_D(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
begin
my_D:=1e-6;
end;
```

with `D_def` set to `my_D`. The `dir` parameter in the header of the diffusion function can take the values: `xdir`, `ydir` and `zdir`. If the diffusivity is homogeneous but anisotropic with $D = 10^{-6}$ m^2 s^{-1} in the x and y directions (i.e. horizontally) and $0.2 \cdot 10^{-6}$ m^2 s^{-1} in the z direction (i.e. vertically), we define:

Example 28 *Homogeneous anisotropic D.*

```
function my_D(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
var dd:datatype;
begin
case dir of
  xdir,ydir:  dd:=1e-6;
  zdir:       dd:=0.2e-6;
else
  error_std('my_D','Unknown direction');
end;
my_D:=dd;
end;
```

Example 29 shows how the diffusion constant found by Rogers and Nielson can be implemented in `RnMod3d`.

The physical meaning of the `D_def`-procedure is different in radon problems and in problems of soil-gas transport. This is discussed in Section 3.4.

7.7 Moisture

`RnMod3d` uses only the material properties defined by the functions pointed to by `beta_def`, `e_def`, `G_def` and `lambda_def`. When these parameters are derived

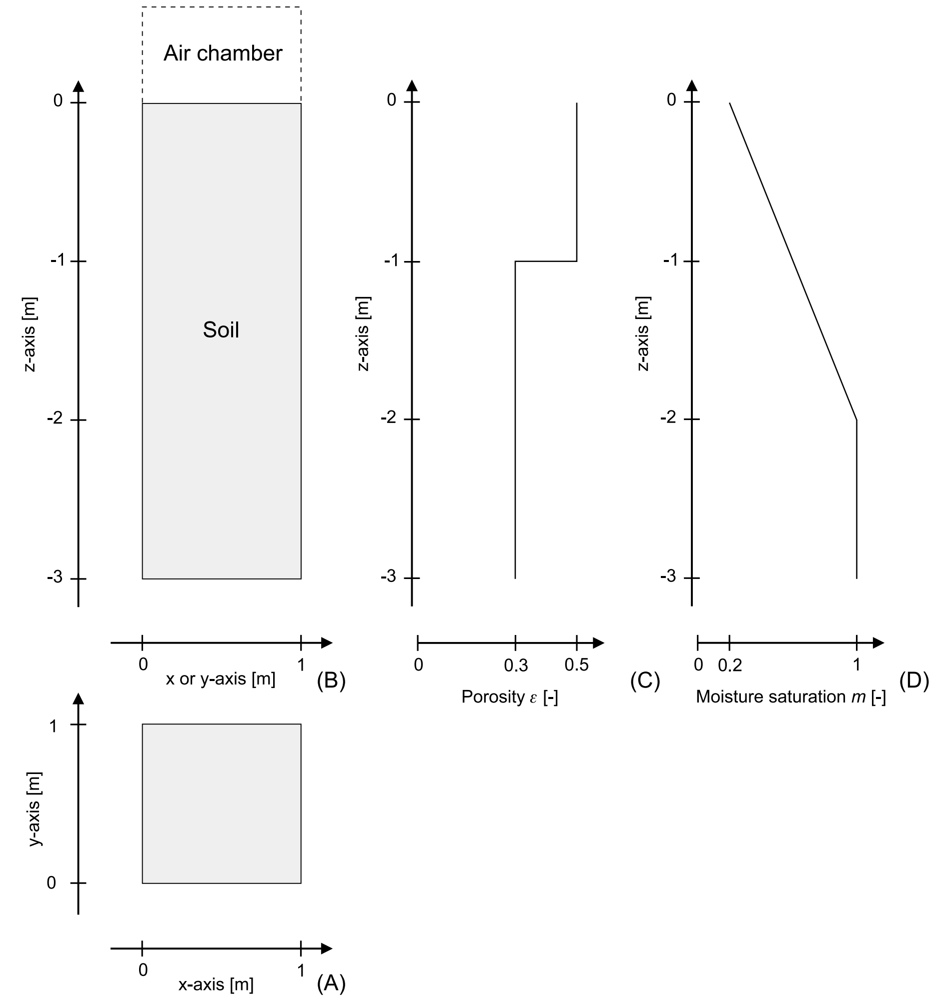


Figure 14. Sketch of the geometry used in case 1 of the ERRICCA model intercomparison exercise [An99[a]]. (A) is the soil column viewed from the top, (B) is a side view of the column. (C) and (D) are plots of porosity (ϵ) and moisture saturation ($m = \theta_\mathrm{v}$), respectively.

from (or related to) some common quantities such as soil moisture content or soil temperature, it is often helpful to introduce such quantities explicitly in the job file. The example below shows how case 1 in the ERRICCA model intercomparison exercise was modelled with `RnMod3d` (see [An99[a]]). The problem is sketched in Figure 14. z goes from 0 at the atmospheric surface to -3.0 m. The function `m` describes the moisture profile (i.e. θ_v as defined page 7). Rogers and Nielson's formula [Rog91A, Rog91B] is used for the calculation of diffusivity. `pw(x,y)` is an in-built power function that returns x^y.

Example 29 *Material properties for ERRICCA case 1.*

```
function e(i:itype;j:jtype;k:ktype):datatype;
begin (* Porosity *)
if znod(k)>-1 then e:=0.5 else e:=0.3;
end;

function m(i:itype;j:jtype;k:ktype):datatype;
var mres,depth:datatype;
begin  (* Moisture saturation, m=ew/e *)
depth:=znod(k);
mres:=0.20-0.4*depth;
```

```
if mres>1 then mres:=1;
if mres<0 then error_std('m','m<0!');
m:=mres;
end;

function beta(i:itype;j:jtype;k:ktype):datatype;
var ea,ew,L:datatype;
begin  (* Partition-corrected porosity *)
ew:=m(i,j,k)*e(i,j,k);  (* Water porosity *)
ea:=e(i,j,k)-ew;        (* Air porosity   *)
L := 0.3565;            (* L Ostwald      *)
beta:=ea+L*ew;
end;

function D(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
const Da=1.1e-5;
var e1,b1,m1:datatype;
begin (* Bulk diffusivity *)
e1:=e(i,j,k);
m1:=m(i,j,k);
b1:=beta(i,j,k);
D:=b1*Da*e1*exp(-6*m1*e1-6*pw(m1,14*e1));
end;

function G(i:itype;j:jtype;k:ktype):datatype;
var Ema,rhog,etot:datatype;
begin (* Radon generation rate *)
Ema:=10.0;    (* Emanation rate, atoms/kg/s *)
rhog:=2.65e3; (* Grain density, kq/m3       *)
etot:=e(i,j,k);
if etot<=0 then error_std('G','etot<=0!');
G:=rhog*(1-etot)/etot*lambda_use*Ema;
end;

function lambda(i:itype;j:jtype;k:ktype):datatype;
begin  (* Decay constant *)
Lambda:=lambda_use;;
end;
```

where we have set:

```
  e_def      := e;
  beta_def   := beta;
  G_def      := G;
  D_def      := D;
  lambda_def := lambda;
```

and where `lambda_use` is a user-defined constant set to $2.09838 \cdot 10^{-6}$ s^{-1}.

8 Flux probes (`Flx1`, `Flx2` etc.)

The primary output of many simulations is the total flux across some plane surface. For example, in house simulations the primary output is the radon entry rate or the soil-gas entry rate into the house.

8.1 Fluxes between individual pairs of control volumes

The basic fluxes in `RnMod3d` are those that go between individual pairs of (adjacent) control volumes. An example is the flux called j_e in Figure 11, page 35.

This is the flux in the x-direction between control volume `P` and `E`. Such fluxes can be found with the function called `node_flux(dir,i,j,k)`. The parameter `dir` tells which side of control volume `(i,j,k)` that is considered: `west`, `east`, `south`, `north`, `bottom`, or `top`. For example, to find the east-side flux (j_e) of control volume `(3,70,4)` simply call the function as:

```
writeln('je = ',node_flux(east,3,70,4));
```

The same value be found if the west-side flux of the adjacent control volume at (i.e. at `(4,70,4)`) is looked up:

```
writeln('jw = ',node_flux(west,4,70,4));
```

Units If a soil-gas problem is considered, then `node_flux` returns a flow of soil gas in units of $\mathrm{m}^3\,\mathrm{s}^{-1}$. If a radon problem is considered, then the result is in units of $\mathrm{Bq\,s}^{-1}$. If the flux density is needed, then the flux should be divided by the area of the interface between the control volumes. This area can be found as described in Section 5.9.

8.2 `update_flxval`

`RnMod3d` has a way for keeping track of fluxes involving many control volumes. Essentially it is possible to ask `RnMod3d` to integrate fluxes over specific areas (not just between single pairs of control volumes). These "flux measurement probes" can be used to monitor fluxes wherever the user wants.

The flux probes are called `Flx1`, `Flx2` etc. These probes are positioned by the user through the user-defined procedure pointed to by the control variable `flux_def`. The idea can be explained with the example given below:

Example 30 *Simple flux measurements.*

```
procedure myfluxes(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then update_flxval(Flx1,top,i,j,k,plus);
end;
```

where we have set `flux_def` equal to `myfluxes`. This procedure defines `Flx1` as the grand sum of the fluxes through the "top" faces of all single control volumes that are part of the xy-plane pin-pointed by the `in_plane` function (Ω).

$$\texttt{Flx1} = \sum_{\Omega} j_{\mathrm{top}}(i,j,k) \tag{45}$$

Essentially, the measurements occur along individual connectors. In the case of `noFlow` or `nill` connectors, there will be no contribution to the flux measurement. For connectors of the type `std`, the flux will be assessed using an approximate versions of equation 41 for radon and equation 43 for soil gas.

It is possible to use any of the control-volume faces for flux measurements–not just the "top" as in the above example. To do that simply call to `update_flxval` with the second parameter set to `bottom`, `east`, `west`, `north` or `south`.

`plus` *and* `minus` Fluxes are taken to be positive if they are in the direction of the x, y or z axis. The last parameter in the `update_flxval`-call can be used to change the sign when the control-volume fluxes are added. `plus` means no change of sign, `minus` means that the sign should be changed. In example 30, `Flx1` will therefore be positive if the flux is in the (positive) direction of the z-axis. A more complicated example, demonstrates the use of `plus` and `minus`

Example 31 *More complex flux measurements.*

```
procedure myfluxes(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then
              begin  (* zFix1 plane *)
                update_flxval(Flx1,top,i,j,k,plus);
                update_flxval(Flx3,top,i,j,k,plus);
              end;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then
              begin (* zFix2 plane *)
                update_flxval(Flx2,bottom,i,j,k,plus);
                update_flxval(Flx3,bottom,i,j,k,minus);
              end;
end;
```

In this example, `Flx1` is the flux across the `zFix1` plane as before. The new fluxes are `Flx2` and `Flx3`. `Flx2` is the flux across the `zFix2` plane, and `Flx3` is the difference between `Flx1` and `Flx2`. Adding fluxes in this fashion is useful for example, in house simulations with more than one entry point.

Flux probes at `nill` *connectors always return zero*

Sometimes it is important to use the "correct" control-volume face for the flux measurements. In the example below, we consider a cubic grid (it could be a cubic sample of concrete). We want to measure the flux through each of the six faces (for example, it could be the total radon exhalation from the sample). If we use `myfluxes_not_ok` for the flux measurements the model will report all fluxes `Flx1` to `Flx6` to be zero! The problem is that we conduct flux measurements on the outer rim of the cube (i.e. between the outer control-volume nodes and nowhere; these connectors are of the type called `nill`). The fluxes defined by `myfluxes_ok` are the correct ones.

Example 32 *Flux measurements at grid boundaries.*

```
procedure mygrid;
begin
set_FixVal(xFix1,0.0);  (* x-axis *)
set_FixVal(xFix2,2.0);
set_axis_single(xFix1,xFix2,5,Focus,1.0);
set_FixVal(yFix1,0.0);  (* y-axis *)
set_FixVal(yFix2,2.0);
set_axis_single(yFix1,yFix2,5,Focus,1.0);
set_FixVal(zFix1,0.0);  (* z-axis *)
set_FixVal(zFix2,2.0);
set_axis_single(zFix1,zFix2,5,Focus,1.0);
end;

procedure myfluxes_not_ok(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix1,
            j,yFix1,yFix2,     (* xFix1 plane *)
            k,zFix1,zFix2) then update_flxval(Flx1,west,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix2,xFix2,
            j,yFix1,yFix2,      (* xFix2 plane *)
            k,zFix1,zFix2) then update_flxval(Flx2,east,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix1,      (* yFix1 plane *)
            k,zFix1,zFix2) then update_flxval(Flx3,south,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix2,yFix2,      (* yFix2 plane *)
            k,zFix1,zFix2) then update_flxval(Flx4,north,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix1,
            j,yFix1,yFix2,      (* zFix1 plane *)
            k,zFix1,zFix1) then update_flxval(Flx5,bottom,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,      (* zFix2 plane *)
            k,zFix2,zFix2) then update_flxval(Flx6,top,i,j,k,plus);
end;

procedure myfluxes_ok(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix1,
            j,yFix1,yFix2,     (* xFix1 plane *)
            k,zFix1,zFix2) then update_flxval(Flx1,east,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix2,xFix2,
            j,yFix1,yFix2,      (* xFix2 plane *)
            k,zFix1,zFix2) then update_flxval(Flx2,west,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix1,      (* yFix1 plane *)
            k,zFix1,zFix2) then update_flxval(Flx3,north,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix2,yFix2,      (* yFix2 plane *)
            k,zFix1,zFix2) then update_flxval(Flx4,south,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix1,
            j,yFix1,yFix2,      (* zFix1 plane *)
            k,zFix1,zFix1) then update_flxval(Flx5,top,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,      (* zFix2 plane *)
            k,zFix2,zFix2) then update_flxval(Flx6,bottom,i,j,k,plus);
end;
```

8.3 FlxVal

After each `run_model` call, the results of flux measurements are stored in an array called `FlxVal`. This array contains two components: `J` and `Q`. In radon simulations, `J` is the calculated flux of radon in Bq s^{-1} and `Q` is the (imported) soil-gas flow rate in $m^3\,s^{-1}$. For example, we can write the result of the `Flx2` probe measurements as follows:

```
  writeln('The results are: ',FlxVal[Flx2].J,' Bq/s ',
                              FlxVal[Flx2].Q,' m3/s ');
```

In a soil-gas simulation `FlxVal[Flx2].J` is the calculated soil-gas flow rate in $m^3\,s^{-1}$, and `FlxVal[Flx2].Q` has no meaning.

This is discussed in Section 3.

8.4 Standard flux probe output

The results of flux measurements are available as standard output in the `LOG`-file and on the screen. The output is discussed in Section 4.36. The output can be turned on and off with the control variables:

```
  wr_final_results_log
  wr_final_results_screen
```

Preliminary flux estimates can also be monitored as `RnMod3d` solves the problem iteratively. This is done with:

```
  wr_flux_during_calc_log
  wr_flux_during_calc_screen
```

In fact, the flux measurements should normally be part of the requirement for convergence. This is done with the control variable `flux_convset` which can contain a Pascal set of flux probes. For example, `Flx2`, `Flx4` and `Flx6` can be included in the convergence test with:

```
 flux_concset:=[Flx2,Flx4,Flx6];
```

See page 25 and Section 10.4 for further details.

9 Field probes (`Obs1`, `Obs2` etc.)

`RnMod3d` is also equipped with a framework for doing radon concentration measurements (in radon simulations) and pressure measurements (in soil-gas simulations). The probes are called `Obs1`, `Obs2` etc.

9.1 `ObsVal`

Probe `Obs2` can be set to monitor the radon concentration in control volume `(i,j,k) = (3,4,2)` as follows:

Example 33 *Simple radon concentration probe.*

```
procedure myprobes;
begin
obsval[obs2]:=GP[3]^[4]^[5].c;
end;
```

The probe definition must be linked to `RnMod3d` with the control variable assignment:

```
  probe_def:=myprobes;
```

The example makes direct access to the main data structure `GP`, where

```
  GP[i][j][k].c
```

is the field value. `GP` also has a record that tells if the field value is valid or not:

```
  GP[i][j][k].valid_fieldvalue
```

This can be used as follows:

Example 34 *Simple radon concentration probe with test of* `valid_fieldvalue`.

```
procedure myprobes;
var i:itype;j:jtype;k:ktype;
begin
i:=3; j:=4; k:=2;
obsval[obs2]:=-999;
if GP[i]^[j]^[k].valid_fieldvalue then
   obsval[obs2]:=GP[i]^[j]^[k].c;
end;
```

This is one way to avoid problems with "dead nodes" (see page 40). Further details about `GP` are given in Section 14.

After each `run_model` call, the results of field-value measurements can be found in the array called `ObsVal`. This array is similar to the one used for flux measurements (`FlxVal`, see Section 8.3). In radon simulations `ObsVal` is the concentration of radon in Bq m^{-3}. In soil-gas simulations, `ObsVal` is the pressure in Pa. For example, we can write the result of the `Obs4` probe measurements as follows: *Units*

```
  writeln('The result is: ',ObsVal[Obs4].c,' Bq/m3 ')
```

Normally, we are not interested in field values for specific control volumes `(i,j,k)` since their significance change with the grid resolution. Instead we need to find field values for physical (x, y, z) locations. `RnMod3d` has four procedures/function for this purpose. These are described in the Section 9.3 to 9.8.

9.2 Standard field probe output

The results of field measurements are available as standard output in the `LOG`-file and on the screen. The output is discussed in Section 4.38. The output can be turned on and off with the control variables:

```
  wr_final_results_log
  wr_final_results_screen
```

Preliminary field probe estimates can also be monitored as `RnMod3d` solves the problem iteratively. This is done with:

```
  wr_probes_during_calc_log
  wr_probes_during_calc_screen
```

In fact, the field probe measurements should normally be part of the requirement for convergence. This is done with the control variable `probe_convset` which can contain a Pascal set of field probes. For example, `Obs2`, `Obs4` and `Obs6` can be included in the convergence test with:

```
 probe_concset:=[Obs2,Obs4,Obs6];
```

See page 25 and Section 10.4 for further details.

9.3 `fieldvalue`

Assume we need to estimate the radon concentration at some physical location (x, y, z). We can use the function:

```
  fieldvalue(xp,yp,zp,valid)
```

for this purpose. The parameter `valid` is a boolean return variable that tells if the call was successful or not. The function finds the field value by linear interpolation among the nearest control volumes. To monitor the radon concentration at (x, y, z) = (2.3 m,−0.2 m,10 m) with probe `Obs4` we could define a procedure as follows:

Example 35 *Radon concentration probe at physical location* (x, y, z)*.*

```
procedure myprobes;
var cc:datatype; valid:boolean;
begin
cc:=fieldvalue(2.3,-0.2,10,valid);
if valid then obsval[obs4]:=cc else obsval[obs4]:=0
end;
end;
```

9.4 `fieldvalue2D`

In two-dimensional simulations (set by the control variable `geometry`), there is no y-coordinate. A special two-dimensional version of `fieldvalue` therefore is available:

```
  fieldvalue2D(xp,zp,valid)
```

9.5 `get_fieldvalue`

In simulations of actual soil-gas radon measurements the physical probe locations may not be known exactly. For example, we may not know the exact depth from which some specific soil-gas radon is taken. For example, we may assess that the sampling depth is "1 m ± 5 cm", where the 5 cm is one standard uncertainty. In a model simulation of the sampling, we may want to assess the influence of the uncertainty of the sampling depth on the radon concentration determination. Clearly, the answer depend on the gradient of the radon concentration field at the sampling location. `RnMod3d` has a (very) simple procedure which can be used for the assessment:

```
  get_fieldvalue(xp,dxp,yp,dyp,zp,dzp,c,dc,valid)
```

`(xp,yp,zp)` is the (x, y, z)-coordinate of the field location of interest. `dxp` is the uncertainty of the `xp`-coordinate. `dyp` and `dzp` are the uncertainties of the two other coordinates. The estimated field value is returned in the variable `c` and the associated uncertainty is in `dc`. The variable `valid` tells if the estimated result is valid or not. The uncertainty is estimated as:

$$\mathrm{d}c = \sqrt{\left(\frac{\partial c}{\partial x}\mathrm{d}x\right)^2 + \left(\frac{\partial c}{\partial y}\mathrm{d}y\right)^2 + \left(\frac{\partial c}{\partial z}\mathrm{d}z\right)^2} \tag{46}$$

9.6 `get_fieldvalue2D`

The two-dimensional version of the previous procedure is:

```
  get_fieldvalue2D(xp,dx,zp,dz,c,dc,valid)
```

9.7 `get_avgfield`

To get the average field over an entire region, the procedure:

```
  get_avgfield(x1,x2,y1,y2,z1,z2,ddd,c,dc)
```

can be used. The region is a box with the physical coordinates given by `x1`, `x2`, `y1`, `y2`, `z1` and `z2`. `ddd` is the resolution (e.g. 0.01 meter). The main result is returned in the variable `c`. The variable `dc` returns the variability of the result.

9.8 `get_avgfield2D`

The two-dimensional version of the previous procedure is:

```
get_avgfield2D(x1,x2,z1,z2,ddd,c,dc)
```

10 Solution procedure

Essentially, `RnMod3d` solves a matrix equation of the form:

$$\mathbf{A}\vec{c} = \vec{b} \tag{47}$$

This equation is set up on the basis of equation 44, page 14. **A** is a matrix of coefficients. These tell how the field quantity (i.e. pressure or radon) moves from one control volume to another. Hence, matrix elements reflect material properties like diffusivity, the size of control volumes etc. Luckily most of the elements are zero as transport can take place only between adjacent control volumes. $\vec{c}$ represents a field of radon concentrations or pressures. If there are 10 000 nodes in the grid then $\vec{c}$ is a column vector with 10 000 elements. Likewise, **A** is a matrix with 10 000 by 10 000 elements. Finally, $\vec{b}$ is a vector with coefficients that relate to the source term. In radon problems, $\vec{b}$ reflects the radon generation rate. In time-dependent problems, $\vec{b}$ also include information about the field at the previous time step.

Because of the shear size of a typical matrix **A**, this equation cannot be solved by simple matrix inversion. Instead iterative solution procedures are used. The iterative solution procedures work as follows: First, a solution $\vec{c}_0$ is guessed. Then on the basis of the procedure, an improved guess $\vec{c}_1$ is found. From this, a new field $\vec{c}_2$ is found etc. This is continued until convergence is met.

10.1 First guess

There are two possible initial field guesses:

- If the control variable `import_final_field_guess` is set to true, then the model imports the initial field guess from the file with the name given by `import_field_name`. The field could come from a file saved after an earlier calculation, where the computation was not carried out all the way to convergence. The earlier calculation could also have been subject to less strict criteria for convergence. See Section 4.19, page 17.
- If the control variable `import_final_field_guess` is set to false, then the initial field guess equals whatever field is stored in the main data structure `GP`. In the very first model run this field is zero all over: $\vec{c}_0 = \vec{0}$.

10.2 Relaxation

To minimize the time it takes to reach convergence, computations are often over-relaxed. The idea is quite simple. Take a look at one particular node in the grid. After the i'th iteration, the field value at this node is c_i. After the next iteration a new value called c_{i+1} is obtained. Each iteration leads to an improved estimate of the true value. In the beginning, relatively large steps are taken (i.e. the difference between c_i and c_{i+1} is large), but eventually step sizes get smaller. Now, if we know in what "direction" the true value can be found, why not take a larger step? With relaxation, we multiply the step size by a factor α:

$$c_{i+1,\mathrm{R}} = c_i + \alpha(c_{i+1} - c_i) \tag{48}$$

If the relaxation factor α is too large, unstability will result. The relaxation factor can be set by the user with the control variable `relax_factor`, see Section 4.52, page 25.

10.3 Iterative solution procedures

The two solution procedures available in `RnMod3d` are both iterative:

- Gauss-Seidel: This is a point-iterative solution procedure. The grid is swept point by point. For each point, we calculate an improved estimate of the field value directly from equation 49 as:

$$c_{P,i+1} = \frac{a_E c_{E,i} + a_W c_{W,i} + a_N c_{N,i} + a_S c_{S,i} + a_T c_{T,i} + a_B c_{B,i} + b}{a_P} \qquad (49)$$

  where $c_{P,i+1}$ is the new improved estimate and all other field values: $c_{E,i}$, $c_{W,i}$ etc. are from the previous iteration.

- Thomas: The Thomas algorithm is similar to that of Gauss-Seidel. The only difference is that the Thomas procedure works line by line. This means faster convergence. The reason is that e.g. the impact of boundary conditions can reach all the way to the other side of the computational plane in one single iteration. To further speed up convergence, the direction of lines is alternated from one iteration to the next: First, a line parallel to the x-axis is selected, then one parallel to the y-axis and finally one parallel to the z-axis.

The solution procedure is selected with the control variable `solver_def` (see Section 4.50).

10.4 Criteria for convergence and residuals

In `RnMod3d`, the convergence criterion consists of three elements:

- The first criteria for convergence is that all flux probes included in `flux_convset` change by less than the value given by `max_change` (see Section 4.53 and 4.59). For example, imagine that `flux_convset := [flx1,flx4]` and `max_change := 1e-4`, then convergence is not met before the results for flux probe `flx1` and probe `flx4` change by less than 0.01 % per iteration. The values of other flux probes (e.g. `flx2` and `flx3`) play no role for the convergence. Observe, that if the final value of one of the flux probes is close to zero, then this can be a problematic requirement. It is best to avoid flux probes with values close to zero in `flux_convset`. Flux probes can be located anywhere in the computational plane as described in Section 8. If none of the flux probes should be part of the convergence criteria, then simply use: `flux_convset := [ ]`. The convergence of flux measurements can be monitored during the iterative procedure as described Section 4.36, page 21.

- The second criteria for convergence is that all field value probes included in `probe_convset` change by less than the value given by `max_change`. For example, imagine that `probe_convset := [obs3]` and `max_change := 1e-4`, then convergence is not met before the results for probe `obs3` change by less than 0.01 % per iteration. It is best to avoid probes with values close to zero in `probe_convset`. The probes can be located anywhere in the computational plane as described in Section 9. It seems best to place probes close to regions of main interest. Probes can also be placed in "corners" of the computational plane where the field (by experience) takes a long time to settle down. The convergence of field-value measurements can be monitored during the iterative procedure as described Section 4.38, page 21.

- The final requirement for convergence is that the sum of residuals is less than `max_residual_sum` (i.e. sufficiently small). This criterion is based on the recommendations given by Patankar [Pa88, p. 236]. After the i'th iteration, the guessed solution of the matrix equation is $\vec{c}_i$. To evaluate how close this solution is to the right one, we insert $\vec{c}_i$ into equation 47 and calculate the residual vector $\vec{r}_i$:
$$\vec{r}_i = \mathbf{A}\vec{c}_i - \vec{b} \tag{50}$$
We then define the absolute sum of residuals R_i (after the i'th iteration) as:
$$R_i = \sum |r_i| \tag{51}$$
where (as before) the sum is over all nodes in the grid. In the end, R_i should approach zero. However, as already mentioned, we consider the problem to be solved when $R_i <$ `max_residual_sum`. To better understand the significance of R_i, it is sometime of interest to know the value:
$$R^0 = \sum |b| \tag{52}$$
where the sum is over all nodes in the grid. This is the value of R_i that is obtained when $\vec{c}_i = \vec{0}$. In the end, R_i should reach a value that is low compared with R^0. In fact, `RnMod3d` gives a warning if R_i multiplied by the constant `residual_sum_warning_limit` is not less than R^0. By default `residual_sum_warning_limit` is set to 100. The results of R_i, R^0 and the maximum value of $\vec{r}_i$ as well as its location in the computational plane can be output from `RnMod3d` during and after the iteration solution procedure, see Section 4.34. The value of R^0 is output as `Abs. sum of bs`.

It takes time to test for convergence. Therefore it is best not to do so in every single iteration. How often the convergence is tested can be set by the control variable `conv_evaluation_period`, see Section 4.55.

Convergence is not the only thing that controls when the iterative solution procedure stops. See `min_iterations` (Section 4.56), `max_iterations` (Section 4.57), and max_time (Section 4.58).

10.5 Scheme (space)

The coefficients a_E, a_W etc. in equation 44 can be calculated in a number of ways. Essentially, the different possibilities relate to the assumed field profile between adjacent nodes. In other words there are different interpolation schemes available. For example, the so-called central scheme is based on the assumption of a linear profile. This is a good approximation if diffusion dominates in the region between the two nodes. On the other hand, if the profile is dominated by advection, then the profile will be shifted to one side. This is used in the so-called up-wind scheme. In real problems, the best profile is somewhere between these two extremes. In `RnMod3d` the following schemes are available:

```
powerlaw
central
upwind
hybrid
exact
```

For example, to use the scheme based on the exact solution of the diffusive-advection equation, simple set the control variable `scheme` to `exact`. See Section 4.51, page 25.

10.6 Scheme (time)

The fully implicit scheme is used (see [Pa80, p. 56]). No alternatives have been implemented.

An important feature of the fully implicit scheme is that steady-state fields can be calculated in one single (large) time step. Another feature is that solutions are unconditionally stable. However, the accuracy is only first order in time, so small time steps are needed to ensure good accuracy [Ve95, p. 173].

11 Time dependency

11.1 `solution := steady`

If the control variable `solution` is set to `steady`, then `RnMod3d` performs a calculation as if the conditions defined by the coefficient functions (i.e. `D_def`, `beta_def` etc.) and the boundary conditions (i.e. `boundary_conditions_def`) have existed since $t = -\infty$. When `run_model` is called, the solution will reflect these conditions. The final solution does not depend on the initial field. This is a so-called steady-state solution.

11.2 `solution := unsteady`

If the control variable `solution` is set to `unsteady`, then `RnMod3d` performs a time-dependent calculation. Each time `run_model` is called, the solution is progressed by one single time step `dtim`. Normally it is necessary to split the simulation into many (small) time steps. Hence `run_model` is called many times.

The "global" time is given by the variable `tim`. Both `dtim` and `tim` are measured in seconds. Calculation of time-dependent problems are simple to set up. In the following example, we first calculate a steady-state field. Then we perform a time-dependent calculation where each time step is given by `dtim`. Initially, `dtim` is only 10 seconds, but we let `dtim` expand by 20 % in each step. After 12 hours (i.e. when `tim` $> 12 \cdot 3600$ seconds) we perform one additional steady-state calculation.

Example 36 *Prototype time-dependent problem.*

```
solution     := steady;
tim          := 0;
dtim         := 0;
run_model; (* Initial field at t=0 *)

solution     := unsteady;
dtim         := 10;
repeat
  dtim:=dtim*1.2; (* Take larger time steps      *)
  tim:=tim+dtim;  (* Update tim                  *)
  run_model;      (* Advance the field by dtim   *)
  writeln(''Results for time = ',tim/3600,' hr',' Flux = ',FlxVal[Flx1].j,'Bq/s')
until (tim>12*3600);

solution:= steady;
run_model;
close_model;
```

The only thing that binds two consecutive model runs together is the calculated field: The "old" field (in `GP`) tells how much radon (or soil gas pressure) is stored in the computational grid. The new model run simply updates the field in accordance with the problem specification. In fact almost everything is set up from scratch before each time step. Hence everything that controls coefficients and boundary conditions can be time-dependent. The sole purpose of the variable `tim` is to have a global time that can be referred to in the procedures that change in time. In other words, `tim` is not used explicitly by the model itself.

Observe, that if a control variable such as `wr_axes` is set to true (see Section 4.42, page 22) then the grid is output every time `run_model` is issued. In a time dependent problem, it is therefore best to set such control variables to false after the first run. Otherwise the `LOG`-file will be flooded.

Order of statements

In example 36, the order of the statements:

```
tim:=tim+dtim;
```

and

```
run_model;
```

is important. The reason is as follows:

1. `tim` is used to control changes in boundary conditions etc. as described in the following (see e.g. Section 11.4).
2. When the statement `run_model` is issued, the boundary conditions etc. must be those that prevail at `tim := tim + dtim`.

Problems may occur if the order of `tim:=tim+dtim` and `run_model` is reversed.

11.3 Initial conditions

There are three possible ways to specify initial conditions:

- The initial field may be read from a file. This is accomplished by setting the control variable: `import_initialfield` to true as described in Section 4.18, page 17. This is, however, only meaningful if the initial field has been calculated on the basis of a grid identical to that used in the (new) computation. Also observe, that after the first time step has been taken, `import_initialfield` should be set to false. A typical example of this type of initial condition is given next. The initial field is assumed to be in the file called `c0.dat`.

  Example 37 *Initial field in a file.*

  ```
  import_field_name    := 'c0.dat'
  import_initialfield := true;
  solution            := unsteady;
  dtim                := 200;
  tim                 := 0;
  repeat
    tim:=tim+dtim;
    run_model;
    import_initialfield := false; (* No further imports *)
  until (tim>12*3600);
  ```

  To store any field (for reuse as an initial field in some later calculation) simply use the control variable `export_field` (see Section 4.20, page 18).

- The initial field is specified in a function. Imagine that the initial field should equal 3000 Bq m^{-3} at all grid points. This can be done as follows. First, we define a function that describes the initial field:

  Example 38 *Initial field by function (part 1).*

```
function myfunction(i:itype; j:jtype; k:ktype):datatype;
begin
myfunction := 3000
end;
```

Then we make the appropriate reference in the body of the program with the control variable `initialfield_def`. For example, we may write:

Example 39 *Initial field by a function (part 2).*

```
initialfield_def    := myfunction;    (* Initial cond. by function  *)
solution            := unsteady;
dtim                := 200;
tim                 := 0;
repeat
  tim:=tim+dtim;
  run_model;
  initialfield_def:= nil;    (* No further initial fields *)
until (tim>12*3600);
```

It is easy to define more complicated fields. The same methods as given in the example page 47 can be used. Additional details can be found in Section 4.17, page 17.

- The initial field is "calculated on the fly". For example, we may start a model simulation by calculation of some steady-state field. This is the method demonstrated in example 36. The point is that all model runs (steady-state or time-dependent) end up with a field that can be used as initial condition for further computations.

11.4 Time-dependent boundary conditions

The first example shows how a boundary condition can change in time. We consider the problem when the pressure at the boundary (e.g. the atmospheric surface) changes periodically in time as:

$$p = \cos(\frac{2\pi}{T_0}t) \tag{53}$$

where T_0 is a period time (e.g. 12 hours). If the pressure at the boundary is called `fixed1`, then we can implement the problem as follows:

Example 40 *Time-dependent change of boundary conditions.*

```
procedure boundary_conditions(i:itype;j:jtype;k:ktype);
const T0=12*3600;
begin
cBC[fixed1]:=cos(2*pi/T0*tim);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix3,zFix3) then set_node(i,j,k,fixed1);
end;
```

As described in Section 6.7, page 41, the fixed-value nodes are controlled by `cBC[fixed1]`, `cBC[fixed2]` etc. It is possible also to change the types of nodes in time. For example, imagine that at `tim` equal to 200 seconds, the boundary at `zFix1` should change from being fixed at 0 to being closed off for transport. We could implement this as follows:

Example 41 *Time-dependent change of type of boundary conditions.*

```
procedure boundary_conditions(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=0;
if in_plane([inside,eqAB],
                i,xFix1,xFix2,
                j,yFix1,yFix2,
                k,zFix1,zFix1) then
begin
  if (tim<200) then
    set_node(i,j,k,fixed1)
  else
    set_node(i,j,k,free)
end;
end;
```

The use of `set_node` is described in Section 6.5, page 41.

11.5 Time-dependent material properties

Changes of coefficients in time, can be implemented as follows:

Example 42 *Time-dependent diffusivity.*

```
function D(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
var DD:datatype;
begin
DD:=1e-5;
if tim>6*3600 then DD:=1e-8;
D:=DD;
end;
```

In this example, the diffusivity changes from 10^{-5} to 10^{-8} m^2 s^{-1} as `tim` equals 6 hours. Other coefficients like porosity, radon generation rate etc. can be made time dependent in a similar fashion.

11.6 Time-dependent flow field of soil gas

In radon problems, the imposed flow field of soil gas may change in time. For example assume, that a flow field has been calculated previously, and that it is imported into the model run by setting `flowfield := import` (see Section 4.22, page 18). For the time period from 0 to 2 hours, we may want to use this flow field directly in the radon calculation. Then we may want to decrease the flow field to 30 % of the original value (see Section 4.23, page 18). When `tim` equals 12 hours, we may want to turn the flow field off. This can all be done as follows:

Example 43 *Time-dependent adjustment of flow field in a radon problem.*

```
solution     := unsteady;
dtim         := 300;
tim          := 0;
flowfactor   := 1.0;
repeat
  tim:=tim+dtim;
  run_model;
  wr_result_line; (* some user-defined procedure *)
  if (tim>2*3600) then flowfactor:=0.3;
  if (tim>12*3600) then flowfactor:=0.0;
until (tim>24*3600);
close_model;
```

Another possibility, is that the flow field changes altogether. For example, imagine two flow fields have been calculated and stored in the files: `Nwind.dat` and

Wwind.dat. The first could correspond to soil-gas flow created as a result of wind from the north. The other could reflect wind from the west. We may want to see the change in the radon field if the wind changes abruptly from north to west when tim equals 12 hours:

Example 44 *Time-dependent shift in flow field in a radon problem.*

```
solution       := unsteady;
dtim           := 300;
tim            := 0;
flowfactor     := 1.0;
flowfield_name := 'Nwind.dat'
repeat
  tim:=tim+dtim;
  run_model;
  if (tim>12*3600) then flowfield_name:='Wwind.dat';
until (tim>24*3600);
close_model;
```

11.7 Full time dependency (cBUF1, cBUF2 and qBUF)

In time-dependent problems, RnMod3d simply updates the main data structure GP by one time step dtim each time run_model is called. This procedure works well if the problem concerns only time-dependent soil-gas transport or if it concerns only time-dependent radon transport. In the general case, however, when both problems are time dependent, the radon simulation will destroy the state of the pressure field (in GP) and likewise, the pressure field simulation will destroy the state of the radon concentration field. The model cannot "remember" more than one field at a time. To treat such problems, it is therefore necessary to be able to store the state of all calculations in some other variable than the main data structure GP. RnMod3d can use two buffers called cBUF1 and cBUF2 for the purpose. These buffers are dynamic variables that are created only when needed. There is also a buffer called qBUF where the flow of soil gas can be stored. With these three buffers, RnMod3d can keep track of two time-dependent problems concurrently.

The use of buffers is controlled by the control variable use_fieldbuffer. With use_fieldbuffer set to cBUF1 the next run_model calculation is encapsulated by the field buffer cBUF1. This means that the first thing that happens after run_model has been called is that the main data structure GP is reset to the state in cBUF1 (the list of actions undertaken in run_model is described in Section 14.7). If there is no such state in cBUF1 (which is always the case in the first run in a job file), GP is not affected by this. Then the computations are performed by RnMod3d in the usual fashion. The last thing that happens before the run_model procedure ends is that the full state of the computed field is stored in the buffer cBUF1. Hence, the next time run_model is called with use_fieldbuffer set to cBUF1, the computations can resume from the state of this field. If the soil-gas problem is encapsulated by the buffer cBUF1, then the radon problem can be encapsulated by cBUF2.

The soil-gas and the radon problems are coupled to each other only by the flow field of soil gas $\vec{q}$ (see equation 40). Luckily radon is present only in trace levels, so the pressure field does not change with the radon concentration. Hence, there is no coupling from the radon field back to the soil-gas problem: The soil-gas problem is completely independent of the radon problem.

qBUF

There are two methods with which the field of soil-gas flows can be transferred from an "ongoing" soil-gas simulation to an "ongoing" radon problem:

- A file is used. This means that flowfield should be set to export in the soil-gas problem, and to import in the radon problem.

- The flow-field buffer (called `qBUF`) is used. In the soil-gas problem, `flowfield` should be set to `export_to_qBUF`. In the radon problem, `flowfield` should be set to `import_from_qBUF`.

A prototype job file with full time dependency is shown in the following example. Observe how control variables have been split into three groups:

- Those control variables that are common for both the soil-gas and the radon problem. These variables are given at the beginning of the main body of the job file, and as they will be not overwritten in the following, these settings remain valid throughout the job file. For example, both problems are calculated with the same grid: `grid_def := grid`.
- Those control variables that are specific for the soil-gas problem. These variables are collected in the procedure called `define_soilgas_problem`. For example, here the permeability of the soil is defined.
- Those control variables that are specific for the radon problem. These variables are collected in the procedure called `define_radon_problem`. For example, here the radon generation rate is defined.

Example 45 *Full time dependency.*

```
program fxxxxprg;
...
procedure define_soilgas_problem;
begin
use_fieldbuffer              := cBUF1;
flowfield                    := export_to_qbuf;
boundary_conditions_def      := boundary_conditions_soilgas;
D_def                        := D_soilgas;
e_def                        := e_soilgas;
beta_def                     := beta_soilgas;
G_def                        := G_soilgas;
lambda_def                   := lambda_soilgas;
...
end;

procedure define_radon_problem;
begin
use_fieldbuffer              := cBUF2;
flowfield                    := import_from_qbuf;
boundary_conditions_def      := boundary_conditions_radon;
D_def                        := D_Rn;
e_def                        := e_Rn;
beta_def                     := beta_Rn;
G_def                        := G_Rn;
lambda_def                   := lambda_Rn;
...
end;

begin (* main *)
runid                        := 'xxxx';
runtitle                     := 'Buffer test';
solution                     := unsteady;
geometry                     := cartesian3d;
grid_def                     := grid;
materials_def                := materials;
...
tim :=0;
dtim:=200;
repeat
  tim:=tim+dtim;
  define_soilgas_problem;
  run_model;

  define_radon_problem;
  run_model;

until (tim>1000);
close_model;
end.
```

dispose_fieldbuffer

If more problems of the above nature are conducted within the same job file, it may be necessary to reset the buffers. This can be done with the procedure `dispose_fieldbuffer`. An example shows what to do.

Example 46 *Use of* `dispose_fieldbuffer`*.*

```
...
begin (* main *)
runid                         := 'xxxx';
runtitle                      := 'Buffer test';
solution                      := unsteady;
geometry                      := cartesian3d;
grid_def                      := grid;
materials_def                 := materials;
...
tim :=0;
dtim:=200;
repeat
  tim:=tim+dtim;

  define_soilgas_problem;
  run_model;

  define_radon_problem;
  run_model;
until (tim>1000);

grid_def := some_new_grid;
dispose_fieldbuffer(cBUF1); (* Reset buffers *)
dispose_fieldbuffer(cBUF2);

tim :=0;
dtim:=200;
repeat
  tim:=tim+dtim;

  define_soilgas_problem;
  run_model;

  define_radon_problem;
  run_model;
until (tim>1000);
close_model;
end.
```

12 Special boundary conditions

The standard boundary conditions in `RnMod3d` are (as described in Section 6.1 and 6.2):

fixed-value conditions where the field is fixed at a given level regardless of the transport equations. For example, in a simulation of radon exhalation from the soil surface into open atmospheric air, we may want to set up a transport simulation for the soil where the concentration at the soil surface is always equal to 5 Bq m^{-3}. Such a condition is modelled by setting all control-volumes at the boundary to be of *type* `fixed1` where `cBC[fixed1]:=5`.

No-flow conditions where the flux is set to zero. For example, in a simulation of radon transport in soil, we may assume that at the ground-water level, there is no transport. This is accomplished by setting all bottom *connectors* of the control-volumes next to the ground water to be of type `noFlow`.

In some radon simulations it is necessary to enforce other boundary conditions. The most important case is when part of the porous medium is in direct contact with open air where radon may accumulate. This occurs, for example, in closed-chamber exhalation measurements. For example, a sample of concrete may be located in a small closed chamber where the air is well mixed by fans [An99^a, An99^b]. This section tells how to do treat such problems.

12.1 Trial-and-error by hand

Clearly the radon concentration in the chamber depends on the flux out of the sample. However, the opposite is normally also true: the flux depends on the radon concentration in the chamber. For example, the maximum flux out of the sample is when the chamber concentration is zero. If there are no other sources than the concrete sample and if the chamber is closed then in steady-state, the following mass balance is fulfilled:

$$J = \lambda\, V\, c \tag{54}$$

where J is the total exhalation rate out of the sample (Bq s^{-1}), λ is the decay constant (s^{-1}), V is the chamber volume (m^3), and c is the concentration of radon in the chamber (Bq m^{-3}). Simulation of this type of a problem with `RnMod3d` can be done as follows:

- The computational grid should only include the concrete. The chamber should not be made part of the grid because here the air is well mixed and the transport is not really covered by the transport equation solved by `RnMod3d`.
- Impose fixed-concentration nodes at the concrete-air boundary. If `fixed1` is used, then set `cBC[fixed1]:=0`.
- Perform a run with the model and calculate the flux of radon into the chamber. The calculated flux is then inserted into equation 54 and the corresponding chamber radon concentration is found. Observe the difference between the assumed chamber concentration (0 in the first run) and the calculated value.
- Now increase the imposed chamber concentration `cBC[fixed1]` by trial-and-error until there is consistency between flux and chamber concentration as given in equation 54.

12.2 `BC_running`

As described in the previous subsection, special boundary conditions can be handled by manual change of the value of a fixed concentration at the boundary. It is, however, sometimes better to let `RnMod3d` do the trial-and-error part of the problem. In particular, it is virtually impossible to solve time-dependent problems "by hand".

In the lack of a better name, the `RnMod3d` system for changing the boundary conditions during the iterative solution procedure is here called *running boundary conditions*. The following control variables are used for the purpose:

```
BC_running
BC_running_update_of_cBCs_def
BC_running_min_iterations
BC_running_max_residual_sum_before_new_BC
BC_running_convergence_def
wr_BC_running_messages_log
wr_BC_running_messages_screen
```

`BC_running`

This is a boolean variable. If it is set to `false` then no adjustment of boundary conditions are carried out. Hence this value must be set to `true` when ”running boundary conditions” are needed.

`BC_running_update_of_cBCs_def`

This is a pointer to a user-defined procedure that controls how the boundary conditions (e.g. `cBC[fixed1]`) are changed. To prevent unstable solutions the process is normally under-relaxed.

`BC_running_min_iterations`

This variable is of type integer. It sets the minimum number of iterations that `RnMod3d` needs to carry out before it attempts to change the boundary conditions. If the value is set too low, the solution procedure can become unstable.

`BC_running_max_residual_sum_before_new_BC`

This floating-point variable gives the maximum sum-of-residuals before `RnMod3d` attempts to change the boundary conditions. If the value is set too high, the solution procedure can become unstable.

`BC_running_convergence_def`

This is a pointer to a user-defined function that returns the value `true` if some user-defined criteria for convergence has been met. Otherwise it should return the value `false`. For example, in a simulation of exhalation from concrete into a chamber it can be tested if there is consistency between the assumed fixed-concentration and the calculated flux.

`wr_BC_running_messages_log`

This is a boolean variable that controls if `RnMod3d` outputs information about the problem to the `LOG`-file.

`wr_BC_running_messages_screen`

This is a boolean variable that controls if `RnMod3d` outputs information about the problem to the screen.

Example

An application of *running boundary conditions* will now be demonstrated. Imagine that a sample of concrete is placed in a chamber. The chamber volume is V and the total flux of radon from the sample into the chamber is called J. There are no other sources of radon in the chamber. The chamber is ventilated with radon-free air. The ventilation rate is λ_v in units of s^{-1} (i.e. the number of air-changes per second). The first task is to write a boundary condition for the chamber. There are three obvious possibilities:

- If the ventilation rate (or the chamber volume) is very large then the chamber radon concentration c_{ch} can be maintained at a near-zero level:
$$c_{\mathrm{ch}} \approx 0 \tag{55}$$
- If system is in steady-state, then the chamber radon concentration must fulfill:
$$J = (\lambda + \lambda_v)\, V\, c_{\mathrm{ch}} \tag{56}$$
where λ is the decay constant for radon.
- It the system is not in steady state then some initial condition must be described for $c_{\mathrm{ch}} = c_{\mathrm{ch}}(t)$ at time zero. For example, the concentration may initially be zero:
$$c_{\mathrm{ch}}(t=0) \approx 0 \tag{57}$$
For $t > 0$ the following condition applies:
$$V \frac{\mathrm{d}c_{\mathrm{ch}}}{\mathrm{d}t} = J - (\lambda + \lambda_v)\, V\, c_{\mathrm{ch}} \tag{58}$$

To simulate such conditions with RnMod3d, we first write a function that returns the value for c_{ch}:

```
function c_chamber:datatype;
const chamber_open=true; (* Open or close the chamber *)
      vol=0.050;         (* Volume is 50 L *)
      lamv=2/3600;       (* Air exchange rate is 2 times per hour *)
      lamd=2.098e-6;     (* Decay constant for radon-222 *)
      lam = lamd+lamv;
var J,dc_dt:datatype;
begin
J:=FlxVal[Flx1].J;  (* Read flux from probe Flx1 *)
if chamber_open then
  c_chamber:=0 (* free exhalation *)
else
  begin (* bound exhalation *)
    if solution=steady then
      c_chamber:=J/(lam*vol)
    else
      begin (* unsteady *)
        dc_dt:=(J-(lam*vol)*c_chamber_old)/vol;
        c_chamber:=c_chamber_old+dc_dt*dtim;
      end;
  end; (* bound exhalation *)
end;
```

where we assume that flux probe Flx1 monitors the total flux of radon out of the sample, and where

```
c_chamber_old
```

is a floating-point variable declared in the job file which is initially set to zero (i.e. before RnMod3d is called the first time).

The nodes at the boundary of the concrete is set to be of type `fixed1` and the chamber radon concentration is hence imposed with `cBC[fixed1]`. For example the (standard) boundary conditions can be programmed as:

```
procedure boundary_conditions(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=0;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then set_node(i,j,k,fixed1);
...
end;
```

A procedure is then needed that can adjust `cBC[fixed1]` in such a way that the desired boundary condition is fulfilled. The following procedure could be used:

```
procedure BC_running_update_of_cBCs;
const relax=0.7;
var cBC_old:datatype;
begin
cBC_old:=cBC[fixed1];
cBC[fixed1]:=cBC_old+relax*(c_chamber-cBC_old);
end;
```

Observe, that we under-relax the update of `cBC[fixed1]` compared to the situation where:

`cBC[fixed1]:=c_chamber;`

We also need a procedure that measures if there is consistency between the imposed chamber concentration (`cBC[fixed1]`) and the value that can be calculated from the boundary condition and the measured flux (`c_chamber`). For example, we could use the following function:

```
function BC_running_convergence:boolean;
const maxchange=1e-7;
begin
if (cBC[fixed1]>0) and
   (abs((cBC[fixed1]-c_chamber)/cBC[fixed1])<maxchange)
then
   BC_running_convergence:=true
else
   BC_running_convergence:=false;
end;
```

Then we just need to set the control variables to use the above procedures. For example the job file could look like this:

```
program F0027prg;
...
var chamber_open:boolean;
    c_chamber_old:datatype;
...
function c_chamber:datatype;
...
end;

function BC_running_convergence:boolean;
...
end;

procedure BC_running_update_of_cBCs;
...
end;

procedure boundary_conditions(i:itype;j:jtype;k:ktype);
...
end;
```

```
function initialfield(i:itype;j:jtype;k:ktype):datatype;
...
end;
...
begin (* main *)
runid                         := '0027';
runtitle                      := 'Test case';
geometry                      := cylindrical2d;
grid_def                      := grid;
force_new_grid_in_every_run   := false;
boundary_conditions_def       := boundary_conditions;
...
flux_convset                    := [flx1,flx2];
probe_convset                   := [obs1..obs3];
conv_evaluation_period          := 50;

BC_running                                := false;
BC_running_convergence_def                := nil;
BC_running_update_of_cBCs_def             := nil;
BC_running_min_iterations                 := 0;
BC_running_max_residual_sum_before_new_BC := 1e-5;
wr_BC_running_messages_log                := false;
wr_BC_running_messages_screen             := false;

min_iterations                  := 60;
max_iterations                  := 20000;
max_time                        := 15*60;
max_change                      := 1e-9;
max_residual_sum                := 1e-19;

solution      := steady;
tim           := 0;
dtim          := 0;
c_chamber_old := 0;
cBC[fixed1]   := 0;

run_model; (* Initial field at t= 0*)

solution     := unsteady;
dtim         := 1800;
c_chamber_old:=cBC[fixed1];

BC_running                                := true;
BC_running_convergence_def                := BC_running_convergence;
BC_running_update_of_cBCs_def             := BC_running_update_of_cBCs;;
BC_running_min_iterations                 := 130;
BC_running_max_residual_sum_before_new_BC := 1e-5;
wr_BC_running_messages_log                := false;
wr_BC_running_messages_screen             := false;

repeat
  tim:=tim+dtim;
  run_model;
  c_chamber_old:=cBC[fixed1];
until (tim>13*3600);

close_model;
end.
```

13 Output and debugging

RnMod3d can be set to generate various types of output. This is mainly controlled by those of the control variables in Section 4 that start with wr_. Output may be directed to the screen or to the LOG-file.

13.1 Standard files

Each model calculation is assigned an identification tag through the control variable called runid. If we set runid := '0997' and run the model, then standard output goes to the files listed in Table 3. The column named *file variable* shows the identification that can be used in Pascal write-statements to write to the files. For example, to write something to the standard result file, simply use:

```
writeln(RES,'Hi there');
```

The standard output files are assigned and opened during the call run_model. This means that the user cannot write to the files before run_model has been called. For example, the following sequence will give run-time error 103: *File not open.*

Example 47 *The following job file gives a run-time error.*

```
program F003prg;
...
begin (* main *)
writeln(RES,'Hi there');
run_model;
close_model;
end.
```

Example 48 *Correct use of* RES*-file.*

```
program F003prg;
...
begin (* main *)
run_model;
writeln(RES,'Hi there');
close_model;
end.
```

The standard RnMod3d files are closed again by close_model. Some of the default filenames in Table 3 may be changed with the control variables:

```
import_field_name
export_field_name
flowfield_name
```

This is explained in Section 4.

13.2 Other file output

Sometimes it is desirable to output results to non-standard files. This can be done easily. First, a text file variable must be declared, and then a file name should be assigned to it. Finally, the file should be opened and closed. For example:

Example 49 *User-defined output.*

```
program F0997prg;
...
var MyF:text;
```

Table 3. Standard `RnMod3d` *files if* `runid := '0997'`*. All these files are ASCII files.*

File name	File variable	Type of output	Purpose
`f0997LOG.dat`	`LOG`	Log file	User readable file
`f0997RES.dat`	`RES`	Generic result file	User readable file
`f0997_01.dat`	`PLT1`	Plot file	User readable file
`f0997_02.dat`	`PLT2`	Plot file	User readable file
`f0997_03.dat`	`PLT3`	Plot file	User readable file
`f0997_00.dat`		File with field (c)	Used by `RnMod3d`
`f0997FLW.dat`		File with soil-gas flow field ($\vec{q}$)	Used by `RnMod3d`
`f0997TMP.dat`		Reserved for later use	Used by `RnMod3d`

```
...
begin (* main *)
assign(MyF,'myfile.dat');
rewrite(MyF);
writeln(MyF,'Hi there');
run_model;
...
close_model;
close(MyF);
end.
```

13.3 Contour plots: `update_plotfile`

`RnMod3d` has a system for creating 2D plot files. These files can be used by software such as Surfer (Golden Software) to create contour plots of the calculated pressure or radon concentration fields. An example is shown in Figure 15. The user has to

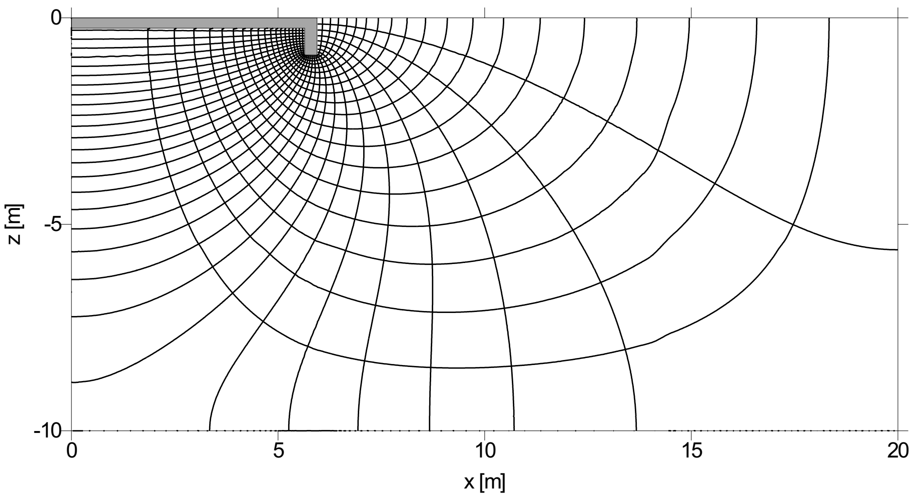


Figure 15. Example of calculated pressure field and streamlines of steady soil-gas entry into a slab-on-grade house. The pressure field is also shown. The contour plot was created with Surfer ver. 7 from Golden Software.

write a procedure with specifications about what should be output. For example, it may be desirable to avoid output for control volumes of the type `NOP` or to limit the output in other ways. An example will be shown in the following. In this case the name of the procedure is `myplots`. `RnMod3d` will use this procedure if the control variable `plotfiles_def` is set as follows:

```
plotfiles_def := myplots;
```

The default value for plotfiles_def is nil. In this case no plot files will be generated during a model run. To specify what should be output, the user needs to use the procedure:

```
update_plotfile(plt,dir)
```

where

plt is one of the standard file variables: PLT1, PLT2 or PLT3.

dir is one of the "directions": xdir, ydir or zdir.

The first parameter tells where the output should go. One of the three plotting file variables given in Table 3 can be used (e.g. PLT1). The second parameter gives the "direction" of the plot. For example, if xdir is selected, then the 2D plot will be perpendicular to the x-axis. Hence, a (y, z)-plot will be generated. An example of a plot file procedure is shown next:

Example 50 *A plot file procedure.*

```
procedure myplots(i:itype;j:jtype;k:ktype);
begin
if (j=2) and (GP[i]^[j]^[k].nodetyp<>NOP) then update_plotfile(plt1,ydir);
if (j=2) and (GP[i]^[j]^[k].mat=mat2) then update_plotfile(plt2,ydir);
if (k=5) then update_plotfile(plt3,zdir);
end;
```

The meaning is as follows: The output directed to the PLT1-file includes all non-NOP control volumes with j-index equal to 2. The PLT2-file gets the same type of output except that now only control volumes of material mat2 are included. Finally, the PLT3-file gets output for all control volumes with k-index equal to 5.

The PLT-output includes index and physical coordinates, field values at the control-volume nodes, coded node type (where 1=free, 2=fixed1 etc.), coded material type (where 2=mat1, 3=mat2 etc.), names of the node type, and names of material. The coded numbers for node type and materials are included to help create plots (e.g. mat1 can be colored in one color and mat2 in another).

Example 51 *Content of a* PLT*-file created with* update_plotfile(plt,dir) *where* dir *has been set to* ydir*.*

```
  i   k               x               z                c nodeno  matno node    mat
  1 124  0.000000E+0000  0.000000E+0000 -3.00000000E+0000      2      3 fixed1  mat2
  2   1  1.879347E-0001 -1.000000E+0001 -7.80006595E-0001      1      2 free    mat1
  2   2  1.879347E-0001 -9.984201E+0000 -7.80006595E-0001      1      2 free    mat1
...
  2  73  1.879347E-0001 -2.500000E-0001 -2.92318620E+0000      1      4 free    mat3
  2  74  1.879347E-0001 -2.453292E-0001 -2.92319051E+0000      1      4 free    mat3
...
  4 116  5.641896E+0000 -8.349609E-0002 -2.99910051E+0000      1      6 free    mat5
...
133 122  2.000000E+0001 -8.789062E-0004 -1.81602884E-0005      1      2 free    mat1
133 123  2.000000E+0001 -9.765625E-0005 -2.01780972E-0006      1      2 free    mat1
133 124  2.000000E+0001  0.000000E+0000  0.00000000E+0000      3      2 fixed2  mat1
```

13.4 Stream lines

In simulations of steady soil-gas transport it is useful to create a plot of the pressure field in the soil. This can be done with the procedure update_plotfile as described earlier. Often it is desirable also to calculate stream lines as this is a good way to visualize the flow. RnMod3d does not include a general procedure for this task. However, it is not difficult to write a procedure for this purpose. The streamlines in Figure 15 have been calculated with the procedure in Example 52.

Example 52 *Calculation of stream lines in the xz-plane.*

```
procedure wr_streamlines;
const streamlinefactor=1e8;
var i:itype;
    j:jtype;
    k:ktype;
    psi,psi_temp:datatype;
    QB,QW:datatype;
    LM:text;
begin
writeln('Write streamlines ....');
assign(LM,'caflow02.dat');
rewrite(LM);
psi_temp:=0;
i:=2;
j:=2;
k:=2;
repeat
  psi:=psi_temp;
  k:=1;
  repeat
    writeln(LM,xnod(i):20,' ',znod(k):20,' ',streamlinefactor*psi:20);
    QW:=GP[i]^[j]^[k].aW*(GP[i-1]^[j]^[k].c-GP[i]^[j]^[k].c);
    psi:=psi+QW;
    k:=k+1;
  until (k=kmax);
  QB:=GP[i]^[j]^[2].aB*(GP[i]^[j]^[1].c-GP[i]^[j]^[2].c);
  psi_temp:=psi_temp-QB;
  i:=i+1;
until (i=imax);
close(LM);
end;
```

13.5 Warnings

After computations with warnings, RnMod3d will show a little table indicating the number of warning flags raised during the run. Further information about where and why the warnings were given can normally be found in the LOG-file (search for warning).

Example 53 *Table of warnings that were issued during the execution of the job file.*

```
--------------------------------------------------------------------------
OBSERVE : Warnings were issued during this session.
Warning:              war_interpolation was issued   20 times
Warning:                      war_other was issued    1 times
Warning:                war_convergence was issued    1 times
Warning:                   war_residual was issued    1 times
--------------------------------------------------------------------------
```

RnMod3d uses the following types of warnings. The warnings are listed in order of importance.

war_interpolation This warning can almost always be ignored completely. The warning is issued by the procedures fieldvalue and fieldvalue2D. To find the field value at any location (x, y, z) these procedures perform interpolation between adjacent nodes. If one or more nodes are of the type NOP (and therefore without a valid field value) this warning is issued. The available (valid) nodes are used for the interpolation.

`war_other` This category contains warnings for problems not covered by the other groups. Examples include warnings from the solver that the maximum number of iterations was reached or that the grid has been redefined.

`war_fileimport` This warning is issued if attempts are made to import a file that does not exist or if the field in the imported file does not match the current grid.

`war_convergence` This warning tells that the run was stopped before the solver reached convergence. This may or may not be a real problem.

`war_residual` This warning signals that the sum of absolute residuals seems to be too large compared to the source term (b). See Section 10.4 for further details.

In jobs with many model runs, it may be useful to include the number of warnings in the result file. The user has access to the warning table through the array:

```
warning_table:array[warningtype] of longint;
```

For example, `warning_table[war_convergence]` equals 0 if all runs have reached convergence. The user can invoke warnings with the procedure:

```
warning_std(idst,message,war)
```

where `idst` and `message` are descriptive strings that tells where and why the warning was created. The type of warning is set by `war`. User-generated warnings should use `war_other`.

13.6 Error messages

Errors will halt `RnMod3d`. Normally, the error message includes the name of the procedure that generated the error and a brief message. If the error cannot be identified from this try to set `wr_details`, `wr_main_procedure_id`, or `wr_all_procedure_id` to `true`, and run the job file again. The user can invoke errors messages with the procedure:

```
error_std(idst,message)
```

where `idst` and `message` are descriptive strings that tells where and why the error occurred.

13.7 Critical evaluation of results

It is important to evaluate the output from `RnMod3d` critically. In particular, it is important to ascertain that the problem solved by `RnMod3d` is actually the problem wanted by the user. Here is a list of things to do:

- Try to start with a very simple version of the problem. Test each level of complexity, and do not add more complexity before tests have been passed.
- Test that the model agrees with simple calculations. For example, in radon simulations try to compare the deep-soil radon concentration with the analytical solution. If the soil is not very deep, then make it so or set the diffusivity to a low value.
- Test if the geometry has been set up correctly. For example, test that the volume of materials is as wanted. This can be done through the control variable `wr_material_volume` (see Section 4.47). This procedure also output material specific minimum and maximum radon concentrations (or pressures in a soil-gas simulation). Are these results as expected?

- Test that the boundary conditions have been set up correctly. For example try to change values and see if the system responds as intended. For example, in a soil-gas simulation involving constant pressures at one or more boundaries, try to set the pressure to zero or change the sign of pressures. In a steady flow situation, the flow of gas into the system must equal the flow out of the system at the other boundaries. Is this fulfilled in the model?
- If a flux measurement give zero result when it should not, then test if the probes are really located correctly (see page 53).
- Test that the solution is not sensitive to further grid refinement. For example, see what results are reached if the number of nodes in the grid is doubled (or halved). Make sure the grid is sufficiently fine in regions where large gradients occur. The technique describe in Section 5.11 may be useful for this purpose.
- In time-dependent problems, test that the solution is not sensitive to the selected time step `dtim`. Try to see what happens if `dtim` is doubled (or halved).
- Make plots of the calculated fields.

14 RnMod3d inside

This section explains a little about the inside part of `RnMod3d`. This information may be helpful during debugging. Fortunately, most variable names are long, descriptive and easy to read. For example, in the code file, the variable that flags if the buffer `cBUF1` has been created or not is called:

```
  cBUF1_has_been_created
```

It is a boolean variable, and can take only the values `true` or `false`. Likewise enumerated data types have been used for many variables. This is discussed in Section 14.6.

14.1 Index coordinates: `i`, `j`, and `k`

The index coordinates `i`, `j`, and `k` are used in `RnMod3d` to refer to specific control volumes. These variables are restricted in range by the associated types defined as follows:

```
  itype = 1..imaxTot;
  jtype = 1..jmaxTot;
  ktype = 1..kmaxTot;
```

The constants `imaxTot` etc. are set by the user as described in Section 5.1.

14.2 The main data structure: `GP`

The main data structure in `RnMod3d` is `GP` ("grid pointer"). This structure contains information about all control volumes. Each node in `GP` point to data which has been declared as follows:

```
nodedatatype=record
  c:datatype;
  aE,aW,aN,aSS,aT,aB,b,ap,ap_old_dt:datatype;
  qE,qW,qN,qS,qT,qB:datatype;
  nodetyp:nodetyptype;
  Wcon,Econ,Scon,Ncon,Bcon,Tcon:nodecontype;
  mat:mattype;
  valid_fieldvalue:boolean;
end;
```

The meaning is as follows:

- Field values. The field value of control volume `(i,j,k)` is stored as:

  `GP[i]^[j]^[k].c`

- a-coefficients. The a_E-coefficient of control volume `(i,j,k)` (see equation 44, page 14) is stored as:

  `GP[i]^[j]^[k].aE`

  The other a-coefficients are stored in a similar fashion. Observe, that the coefficient `ap_old_dt` is not part of equation 44. This coefficient has been introduced to maintain conservation of mass even if β changes from one time step to another. Without going into details, we observe, that `aP_old` (see [Pa80]) should correspond to the time when the last field was calculated. If we ignore transport, generation and decay we have (symbolically) : $\beta(1) \cdot c_{\mathrm{a}}(1) = \beta(0) \cdot c_{\mathrm{a}}(0)$, where 0 and 1 represent old and new, respectively. In terms of coefficients this means, that `ap_new*ca = ap_old*ca_old`.

- Material type (i.e. `mat1`, `mat2` etc.) is stored as:

  `GP[i]^[j]^[k].mat`

- Soil-gas fluxes. The soil-gas flux through the east interface of the control volume `(i,j,k)` is:

  `GP[i]^[j]^[k].qE`

  Fluxes through the other interfaces are stored as `qW`, `qN` etc.

- Node type. The type of node (e.g. `free` or `fixed1`) of the control volume `(i,j,k)` is given by:

  `GP[i]^[j]^[k].nodtyp`

- Connector type. The type of connector (e.g. `std` or `NoFlow`) through the east interface of the control volume `(i,j,k)` is given by:

  `GP[i]^[j]^[k].Econ`

  Connectors for the other interfaces are stored as `Wcon`, `Ncon` etc.

- Valid field value. `GP` also contains a flag that tells if the field value is valid or not. That is given by:

  `GP[i]^[j]^[k].valid_fieldvalue`

14.3 Other variables

This is a list of some other variables that sometimes are needed in job files:

`cBC`: The fixed values used in fixed-value boundary conditions `fixed1` etc. See Section 6.7.

`FlxVal`: The results of flux measurements with `Flx1` etc. See Section 8.3.

`Obsval`: The results of field measurements with `Obs1` etc. See Section 9.1.

`wFixVal`: The values of fix points `xFix1` etc. See Section 5.4.

`x[i]`, `y[j]`, `z[k]`, `dx[i]`, `dy[j]`, and `dz[k]`. The location and size of individual control volumes. See Section 5.9

14.4 datatype

All floating-point computations are done with variables of the type called `datatype`. By default `datatype` is set to equal `extended`. To decrease the use of memory or to test the sensitivity of the results to the internal number representation `datatype` should be set to `double`, `real` or `single`.

14.5 Memory

Maximum grid size

Memory is allocated dynamically (during runtime) for the main data structure `GP`. Section 5.1 tells how the maximum grid size can be changed. Other data structures are static variables.

14.6 Enumerated types

Wherever meaningful, enumerated data types have been used in `RnMod3d`. For example, variables that hold the type of node of a control volume are declared to be of type `nodetyptype`, which in turn is declared as:

```
nodetyptype = (nop,
               free,
               fixed1,
               fixed2,
               fixed3,
               fixed4,
               fixed5,
               NodX);
```

The use of enumerated types has four implications: (1) job-file assignments like: `geometry := cartesian3D` are readable, (2) it is easy to find the possible geometries implemented in `RnMod3d` (just look up the declaration of `geometry` in the source file), (3) should there be any problems with one of the geometries, it is relatively easy to identify those places in the source code where that geometry is treated, and finally (4) enumerated types can be used in Pascal set calculations (see example page 25). Most enumerated types have predefined functions that can convert variables to descriptive strings. For example, the main program file contains a function that can be used to print out the value of a variable of the type `nodetyptype`:

```
function nodetyp_string(x:nodetyptype):string;
var st:string;
begin
case x of
  NOP:        st:='NOP    ';
  free:       st:='free   ';
  fixed1:     st:='fixed1 ';
  fixed2:     st:='fixed2 ';
  fixed3:     st:='fixed3 ';
  fixed4:     st:='fixed4 ';
  fixed5:     st:='fixed5 ';
  NodX:       st:='unchanged ';
else
  st:='Unknown !!';
end; (* case *)
nodetyp_string:=st;
end;
```

These functions can be useful during debugging.

14.7 Sequence of actions in run_model

RnMod3d starts to do computations when the procedure run_model is called. The model may find a steady-state field (if solution has been set to steady) or it may advance the field by one single time step dtim (if solution has been set to unsteady). To use RnMod3d with confidence, it is important to know the sequence of actions in the procedure run_model. The details can be read from the exact programming of run_model in the file R3Main03.pas. To learn more, it may also be useful to let RnMod3d echo procedure names etc. during runtime. This can be done with the control variables:

```
wr_details
wr_main_procedure_id
wr_all_procedure_id
```

The following gives a summary of the actions in run_model:

1. Initially two things can happen:
   - If this is the first run_model call in a job file then all variables will be initialized. For example the entire field in the main data structure GP is set to zero:

     ```
     GP[i]^[j]^[k].c:=0
     ```

   - If this is *not* the first run in a session, then two situations can occur:
     - If the control variable use_fieldbuffer is set to cBUF1, then the state of RnMod3d is set back to the state RnMod3d ended up in the last time run_model was called with use_fieldbuffer set to cBUF1. Likewise, if use_fieldbuffer is set to cBUF2, RnMod3d is restored to that found in the field buffer cBUF2. If nothing has yet been saved in the buffer, then nothing is restored. Hence GP will not be changed (for this reason). This situation is identical to than discussed next where use_fieldbuffer has been set to no_cBUF.
     - If the control variable use_fieldbuffer is set to no_cBUF, then no change of the main data structure GP is performed at this stage. Hence the results of the previous run (still present in GP) are used as a starting point in the current computation (unless changed in one of the steps listed below).
2. If needed, then a grid is generated. Three situations can occur:
   - If this is the first run in a job file, then a grid is generated in accordance with the procedure pointed to be grid_def.
   - If the control variable grid_def has changed from a previous call of run_model, then a new grid is generated (as specified by the procedure now pointed to by grid_def).
   - If the control variable force_new_grid_in_every_run has been set to true, then a new grid is always generated with the procedure *now* pointed to by grid_def. This may or may not be a truly new grid (see Section 4.7).
3. A flow field of soil gas may be imported. Clearly this is only meaningful if a radon problem is solved. This field corresponds to $\vec{q}$ in Box 1, page 11. Three situations exist:
   - If the control variable flowfield has been set to import, then a flow field is imported from the file given by flowfield_name.

- If the control variable `flowfield` has been set to `import_from_qbuf` then a flow field is imported from the flow-field buffer called `qBUF`. Of course this is possible only if a flow field has been calculated (and saved in the buffer) earlier in the job file.
- For all other settings of `flowfield`, the flow field will be set to zero all over.

4. All nodes and connectors are always (even in time-dependent problems) set back to the default configuration. As described in Section 6.3, the computational domain now simulates a "closed box".
5. Nodes and connectors are then changed (from the default settings) in accordance with the procedure pointed to by `boundary_conditions_def`.
6. If this run is the first in a time-dependent problem (i.e. if `solution` is set to `unsteady`) then an initial field may be set up as follows:
   - If `import_initialfield` is set to `true` then an initial field is imported from the file given by `import_field_name`.
   - If `initial_field_def` is not set to `nil` then an initial field is set up from the procedure pointed to by `initial_field_def`.
   - In all other cases, the field in the grid pointer `GP` is taken to be the initial field. The following situations occur:
     – If this is the first run in the job file, then the initial field is zero at all nodes except those fixed to certain values as given by the procedure pointed to by `boundary_conditions_def`.
     – If this is not the first run in the job file, then the result of the previous calculation is now used as initial field in this run. Observe that the previous run could have been a steady-state calculation as well as a time step in a time-dependent calculation.
7. The procedure `materials_def` is called. This means that the control volumes in the computational domain are set to be of materials `mat1`, `mat2` etc.
8. The coefficient matrix is then set up. This involves calling the user-defined procedures:
   - `beta_def`
   - `e_def`
   - `G_def`
   - `D_def`
   - `lambda_def`

   If any of the nodes are of the type `fixed1`, `fixed2` etc. then the values of these nodes are set to equal the values in `cBC`.
9. The problem has now been fully specified. Before it is solved it is possible to make a guess of the solution. If `import_finalfield_guess` is set to `true` then such a guess is imported from the file given by `import_field_name`. If such a guess is not imported, then the field present in `GP` is used as initial guess. For example, in time-dependent problems the result of the previous time step is usually a good guess of the next one.
10. The procedure `find_field` is then called. This procedure in turn calls the solver pointed to by `solver_def`. When convergence has been reached (or the solver was stopped for other reasons), the final field is returned in `GP`. During the iterative solution procedure it is possible to revise the boundary conditions (e.g. `cBC[fixed1]`) as `RnMod3d` calls the procedure pointed to by

```
BC_running_update_of_cBCs_def
```

Such *running boundary conditions* are described in Section 12. Furthermore, the procure pointed to by:

```
user_procedure_each_iter_def
```

is called during each iteration (see Section 4.28).

11. Various types of output are generated.

12. In a soil-gas problem, it is meaningful to store the resulting flow from node to node as a flow field. It can then be used in a later radon simulation. Three situations occur:
    - If the control variable `flowfield` has been set to `export` then a flow field is exported to the file given by `flowfield_name`.
    - If the control variable `flowfield` has been set to `export_to_qbuf` then a flow field is exported to the flow-field buffer. It can then be used later in the current session.
    - For all other settings of `flowfield`, the flow field will not be stored.

13. Finally the state of `RnMod3d` may be stored for later use:
    - If the control variable `use_fieldbuffer` is set to `cBUF1` or `cBUF2`, then the present state of `RnMod3d` is stored in buffer `cBUF1` or `cBUF2`, respectively.
    - If `use_fieldbuffer` is set to `no_cBUF`, then the state of `RnMod3d` is not stored in a buffer.

    The results of the computations (`GP`), will however, in all cases remain intact and can be used in additional `run_model` calls within the same job file. However, observe, that all information (in bufferes or `GP`) are always lost when the job file ends. To transfer information from one job to another, results have to be stored in files.

15 Benchmark tests

To verify that `RnMod3d` gives accurate results, it is necessary to perform benchmark tests on the basis of problems with known solutions. This section contains some simple examples. `RnMod3d` has also been compared with other radon models for more complicated problems (without known solutions) [An99^a].

15.1 `F0100prg`: Steady flow of soil gas

This problem concerns steady Darcy flow of soil gas in a 1 m x 1 m x 3 m column with homogeneous sand. The gas permeability k of the sand is $2 \cdot 10^{-10}$ m^2 and the dynamic viscosity μ is $17.5 \cdot 10^{-6}$ Pa s. The disturbance pressure is 0.0 Pa at the bottom of the column and -3.0 Pa at the top. Other column sides are closed off for transport.

Analytical solution

The exact flow through the column is:

$$\begin{aligned} Q &= A\frac{k}{\mu}\frac{\Delta p}{L} && (59) \\ &= 1\ \mathrm{m}^2 \frac{2\cdot 10^{-10}\ \mathrm{m}^2}{17.5\cdot 10^{-6}\ \mathrm{Pa\,s}}\frac{3\ \mathrm{Pa}}{3\ \mathrm{m}} && (60) \\ &= 1.14285714\ldots\cdot 10^{-5}\ \mathrm{m}^3\,\mathrm{s}^{-1} && (61) \end{aligned}$$

The pressure in the center of the column is -1.5 Pa.

Model implementation

The full job file can be found in Appendix A. Observe, that the "diffusivity" is set to $\frac{k}{\mu}$. This is in accordance with the formalism presented in Table 2, page 12. Also, observe, that other soil parameters are set to zero. Since the flow is steady, it is not necessary to assign any value to the "partition-corrected porosity" β. In unsteady flow simulations β must be set to ϵ_a/P_0. A flux-measurement probe called `Flx1` is placed at the bottom of the column, and another probe called `Flx2` is placed at the top. A pressure probe called `obs1` is placed in the center of the column (i.e. at $(x, y, z) = (0.5\ \mathrm{m}, 0.5\ \mathrm{m}, -1.5\ \mathrm{m})$).

Results

The model gives the following results:

`Flx1`: J $= 1.1428571\cdot 10^{-5}\ \mathrm{m}^3\,\mathrm{s}^{-1}$

`Flx2`: J $= 1.1428571\cdot 10^{-5}\ \mathrm{m}^3\,\mathrm{s}^{-1}$

`Obs1`: c $= -1.500$ Pa

which is in perfect agreement with the true result. Additional results can be found in the `LOG`-file, which is listed in Appendix B.

15.2 `F0101prg`: Steady diffusion of radon

This problem is case 0 of the ERRICCA model intercomparison exercise described [An99^a]. The problem concerns steady diffusion of radon in a sand column. The job file `F0101prg.dpr` is listed in Appendix C. The following results are obtained:

`Flx1`: J $= 0.000\ \mathrm{Bq\,s}^{-1}$

`Flx2`: J $= 4.72197\cdot 10^{-2}\ \mathrm{Bq\,s}^{-1}$

`Obs1`: c $= 4.19221\cdot 10^{4}\ \mathrm{Bq\,m}^{-3}$

The `Flx1` flux measurement verify that the no-flow boundary condition is fulfilled at the bottom. The `Flx2` result is in good agreement with the true result: $J = 4.722828\cdot 10^{-2}$.

15.3 `F0102prg`: Diffusion and advection of radon

This problem concerns a column of homogeneous sand of height $L = 5$ m and cross-sectional area 1 m^2. Both steady Darcy flow of soil gas and combined diffusion and advection of radon are considered. The problem is sketched in Figure 16. The sand has the following properties:

The sand is homogeneous and dry

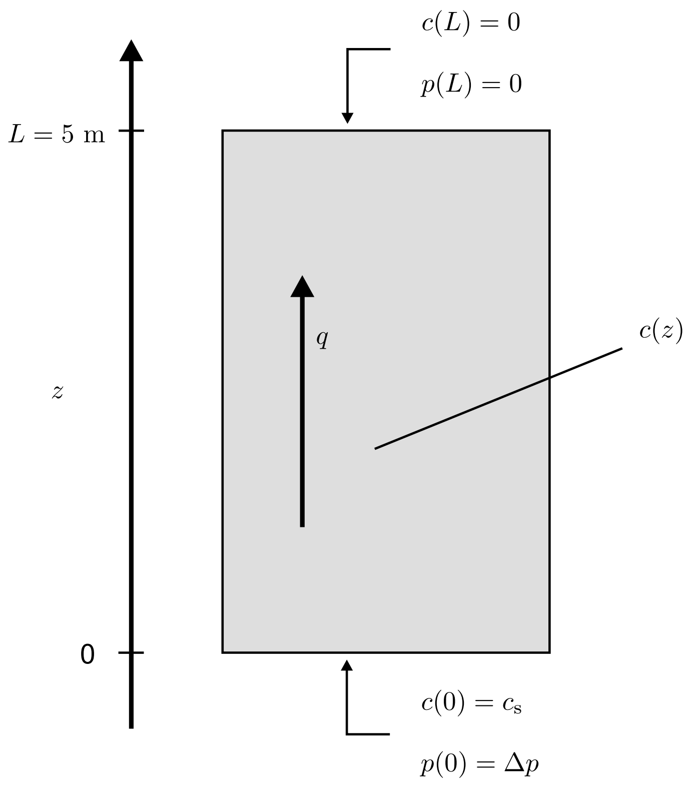


Figure 16. Sketch of the problem treated by `F0102prg`.

The gas permeability k is 10^{-11} m^2

The radon generation rate G equals $\lambda \cdot 10000$ Bq m^{-3}

The porosity ϵ is 0.3

The bulk diffusivity D is 10^{-6} m^2 s^{-1}

The following boundary conditions apply for the flow of soil gas:

At $z = L$, the disturbance pressure is 0 Pa.

At $z = 0$, the disturbance pressure is Δp. Three cases will be investigated:

- $\Delta p = -100$ Pa
- $\Delta p = 0$ Pa
- $\Delta p = 100$ Pa

Other boundaries are closed for transport.

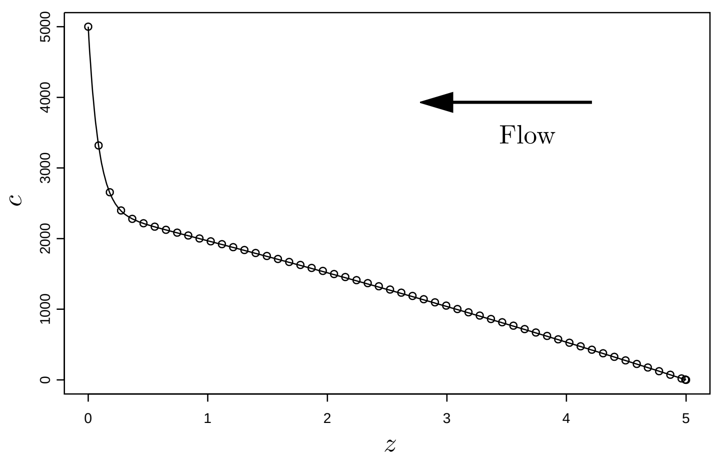
The following boundary conditions apply for the transport of radon:

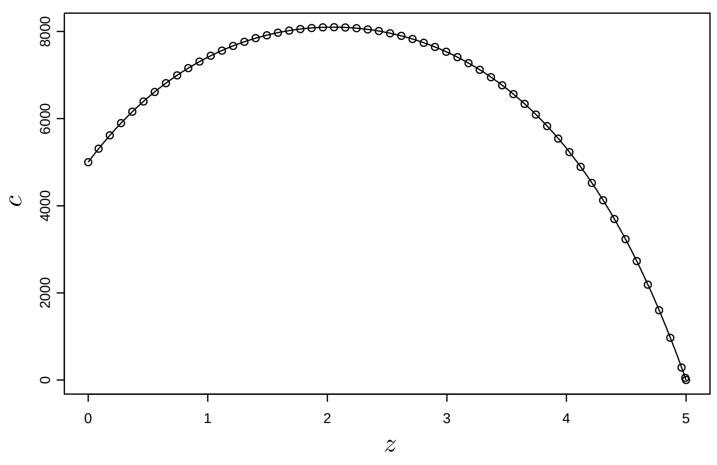
At $z = L$, the radon concentration is set to 0.

At $z = 0$, the radon concentration is $c_s = 5000$ Bq m^{-3}.

Other boundaries are closed for transport.

The decay constant (λ) is set to $2.09838 \cdot 10^{-6}$ s^{-1} and the dynamic viscosity (μ) is set to $17.5 \cdot 10^{-6}$ Pa s.





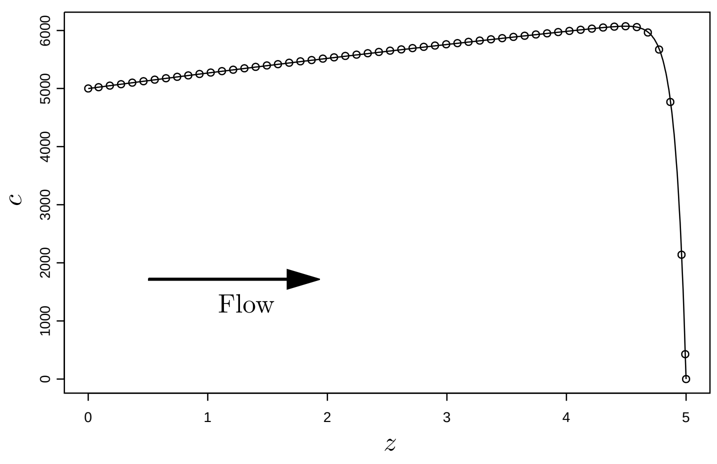


Figure 17. Radon concentration profiles in the sand column in `F0102prg`. *z is measured in* m *and c in* $\mathrm{Bq\,m^{-3}}$. *The three plots correspond to Δp equal to -100 Pa (top), 0 Pa (middle), and 100 Pa (bottom). The exact results are shown as lines.* `RnMod3d` *results are shown as circles.*

Analytical solution

The analytical solution to the above problem can be found in [Co81, p. 26]. The radon concentration in the column ($0 \le z \le L$) is:

$$\begin{aligned} c(z) &= \frac{G}{\lambda}\left(1 - \frac{\exp(\frac{qz}{2D})\sinh\frac{L-z}{\Lambda} + \exp\frac{-q(L-z)}{2D}\sinh\frac{z}{\Lambda}}{\sinh\frac{L}{\Lambda}}\right) \\ &\quad + c_{\mathrm{s}}\exp(\frac{qz}{2D})\frac{\sinh\frac{L-z}{\Lambda}}{\sinh\frac{L}{\Lambda}} \end{aligned} \tag{62}$$

where the soil-gas flow rate in the direction of the z-axis is:

$$q = \frac{k}{\mu}\frac{\Delta p}{L} \tag{63}$$

and

$$\Lambda = \left(\frac{q^2}{4D^2} + L_d^{-2}\right)^{-0.5} \tag{64}$$

and where L_d is the diffusion lenght:

$$L_d = \sqrt{\frac{D}{\epsilon\lambda}} \tag{65}$$

The flux at $z = L$ is:

$$\begin{aligned} j &= \left(-D\frac{\mathrm{d}c}{\mathrm{d}z} + qc\right)\Bigg|_{z=L} \\ &= \frac{G}{\lambda}\left(\frac{q}{2} + \frac{D}{\Lambda}\frac{\cosh\frac{L}{\Lambda} - \exp\frac{qL}{2D}}{\sinh\frac{L}{\Lambda}}\right) + c_{\mathrm{s}}\frac{D}{\Lambda}\frac{\exp\frac{qL}{2D}}{\sinh\frac{L}{\Lambda}} \end{aligned} \tag{66}$$

Model implementation

Appendix D shows how the problem with $\Delta p = -100$ Pa has been implemented in `RnMod3d`. The job file also contains the exact solution.

Comparison of results

Table 4 and Figure 17 show that there is good agreement between the results of `RnMod3d` and the analytical solution.

Table 4. Results for the radon flux at $z = L$.

P	`RnMod3d`	Exact	Deviation
Pa	$\mathrm{Bq\,s^{-1}}$	$\mathrm{Bq\,s^{-1}}$	%
-100	$5.4545 \cdot 10^{-4}$	$5.4820 \cdot 10^{-4}$	-0.5
0	$7.7855 \cdot 10^{-3}$	$7.7896 \cdot 10^{-3}$	-0.05
100	$7.0965 \cdot 10^{-2}$	$7.0973 \cdot 10^{-2}$	-0.01

15.4 `F0103prg`: Time-dependent flow of soil gas

This case concerns unsteady Darcy flow of gas in a finite soil column of height ℓ. Initially at $t = 0$, the disturbance-pressure field in the column equals zero. Then at $t = 0$, the disturbance pressure at one end of the column ($z = \ell$) starts to oscillate as:

$$p_{\mathrm{atm}}(t) = p_1\sin(\omega t + \varphi) \tag{67}$$

where p_1 is the amplitude of the variations (e.g. 1 Pa), and where the period time is:

$$T = \frac{2\pi}{\omega} \tag{68}$$

The disturbance pressure field at the other end of the column ($z = 0$) remains at zero. Inside the soil column, the disturbance-pressure field $p(z,t)$ is governed by equations 42 and 43. The problem geometry is sketched in Figure 18.

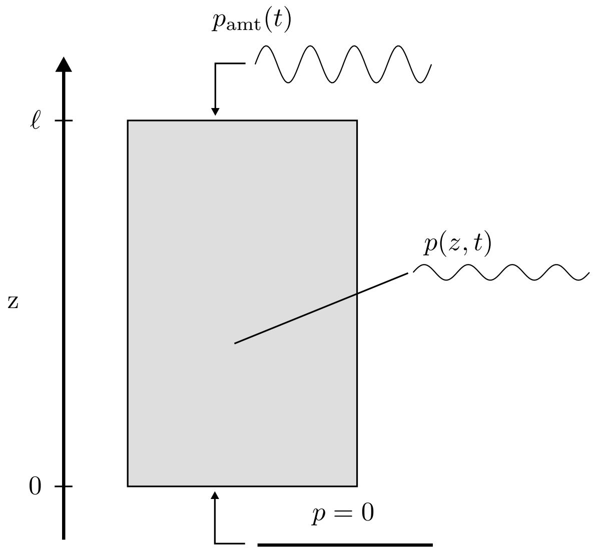


Figure 18. Sketch of problem treated by `F0103prg`*: The disturbance pressure at the boundary at $z = \ell$ starts to oscillate at $t = 0$. After some transient period, this in turn starts oscillations of the same frequency in the soil. The phase and amplitude change with z (and soil parameters).*

Analytical solution

The exact solution to the problem ($0 \leq z \leq \ell$ and $t \geq 0$) can be found in Carslaw and Jaeger [Car59, p. 105]:

$$\begin{aligned} p(z,t) &= p_1 A \sin(\omega t + \varphi + \phi) + \\ & 2p_1 \pi D_p \sum_{n=1}^{\infty} \frac{n(-1)^n (D_p n^2 \pi^2 \sin\epsilon - \omega \ell^2 \cos\epsilon)}{D_p^2 n^4 \pi^4 + \omega^2 \ell^4} \sin\left(\frac{n\pi z}{\ell}\right) \exp\left(-D_p n^2 \pi^2 t / \ell^2\right) \end{aligned} \tag{69}$$

where

$$A = \left| \frac{\sinh \theta z(1+i)}{\sinh \theta \ell (1+i)} \right| \tag{70}$$

$$\phi = \arg\left(\frac{\sinh \theta z(1+i)}{\sinh \theta \ell (1+i)} \right) \tag{71}$$

and where

$$\theta = \left(\frac{\omega}{2D_p} \right)^{\frac{1}{2}} \tag{72}$$

D_p is defined in Section 3.3.

Parameters

We consider the following parameters:

$\ell = 5.0$ m

$k = 10^{-14}$ m^2

$P_0 = 1.0 \cdot 10^5$ Pa

$\epsilon_{\mathrm{a}} = 0.2$

$\mu = 17.5 \cdot 10^{-6}$ Pa s

$p_1 = 3.0$ Pa

$T = 10$ hours (period time)

$\varphi = \pi/2$

Observe, that with $\varphi = \pi/2$, the pressure at $z = \ell$ will "jump" from 0 to $p = 3$ Pa at $t = 0+$.

Model implementation

The model implementation is shown in Appendix E. The properties of the soil are set in accordance with Table 2. In particular observe, that the "diffusivity" is set to $\frac{k}{\mu}$, and that `beta` is set to $\frac{\epsilon_{\mathrm{a}}}{P_0}$.

The conditions at `tim:=0` are calculated with `solution:=steady`. For the subsequent runs, `solution` is set to `unsteady`. Only the boundary condition `fixed2` at the atmospheric surface changes in time. The main results of the computations are written to the file `F0103RES.dat`. The output includes pressures in the center of the column:

- `obs1` at $z = 0.2$ m
- `obs2` at $z = 1.0$ m
- `obs3` at $z = 2.5$ m
- `obs4` at $z = 4.0$ m
- `obs5` at $z = 4.8$ m

The computations are stopped after 4 cycles (i.e. when `tim` equals 40 hours).

Evaluation

As can be seen from Figure 19, the results obtained with `RnMod3d` agree well with those of the exact analytical solution in equation 69.

Additional comments

Riley et al. use the same test as just discussed to verify their model code called `RapidSTART` [Ri99].

`RnMod3d` has been tested against also the case described in Carslaw and Jaeger as *2.6 Semi-infinite solid. Surface temperature a harmonic function of the time* [Car59, p. 64]. Again near-perfect agreement between `RnMod3d` results and those of the exact analytical solution was obtained. However, the analytical solution was relatively difficult to integrate numerically[6].

[6]Peter Kirkegaard at Risø is thanked for performing the integration.

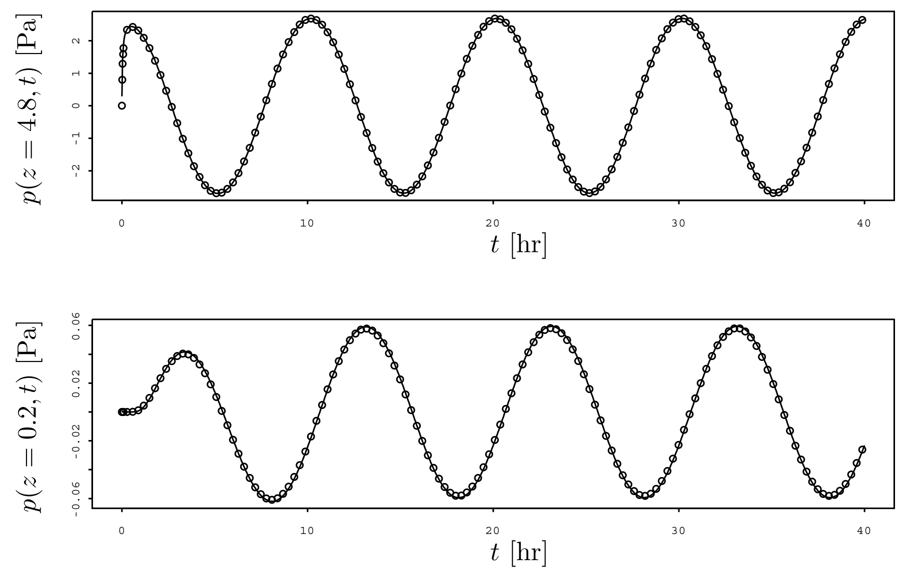


Figure 19. Comparison of results obtained with `F0103prg` *(circles) and the analytical solution given in equation 69 (line). The top plot is the disturbance pressure at $z = 4.8$ m. The bottom plot is for $z = 0.2$ m.*

16 House simulation example

This section demonstrates how `RnMod3d` can be set up to do calculations of soil gas and radon entry into a house. The house sketched in Figure 20(A) is considered. It is a 100 m^2 slab-on-grade house. For simplicity, the house is chosen to be cylindrical. There is a 3 mm gap of air between the slab and the footer along the full 35 m perimeter of the house. This is clearly an important route of entry, however, transport can also take place through the concrete slab. A highly permeable layer of gravel exists below the slab. The house is located on a 10 m thick soil block of 20 m radius. The house is constantly depressurized 1 Pa relative to the outdoors. Further details about parameters etc. will be given in the following.

Geometry

Figure 20(A) shows the geometry of the house. The house is cylindrical, so we set

```
geometry := cylindrical2D;
```

With the fix points `xFix1` to `xFix5` and `zFix1` to `zFix5`, it is possible to give an accurate representation of all geometrical features of the house. Table 5 gives the dimensions of the different components. For example, the width (i.e. the radius) of the slab is 5.6419 m, such that in fact the area of the floor is the required 100 m^2. Accordingly, we assign the fix points as follows:

```
set_FixVal(xFix1,0.000); (* x-axis *)
set_FixVal(xFix2,Lx_slab-Lx_gap);
set_FixVal(xFix3,Lx_slab);
set_FixVal(xFix4,Lx_slab+Lx_footer);
set_FixVal(xFix5,Lx_soil);
```

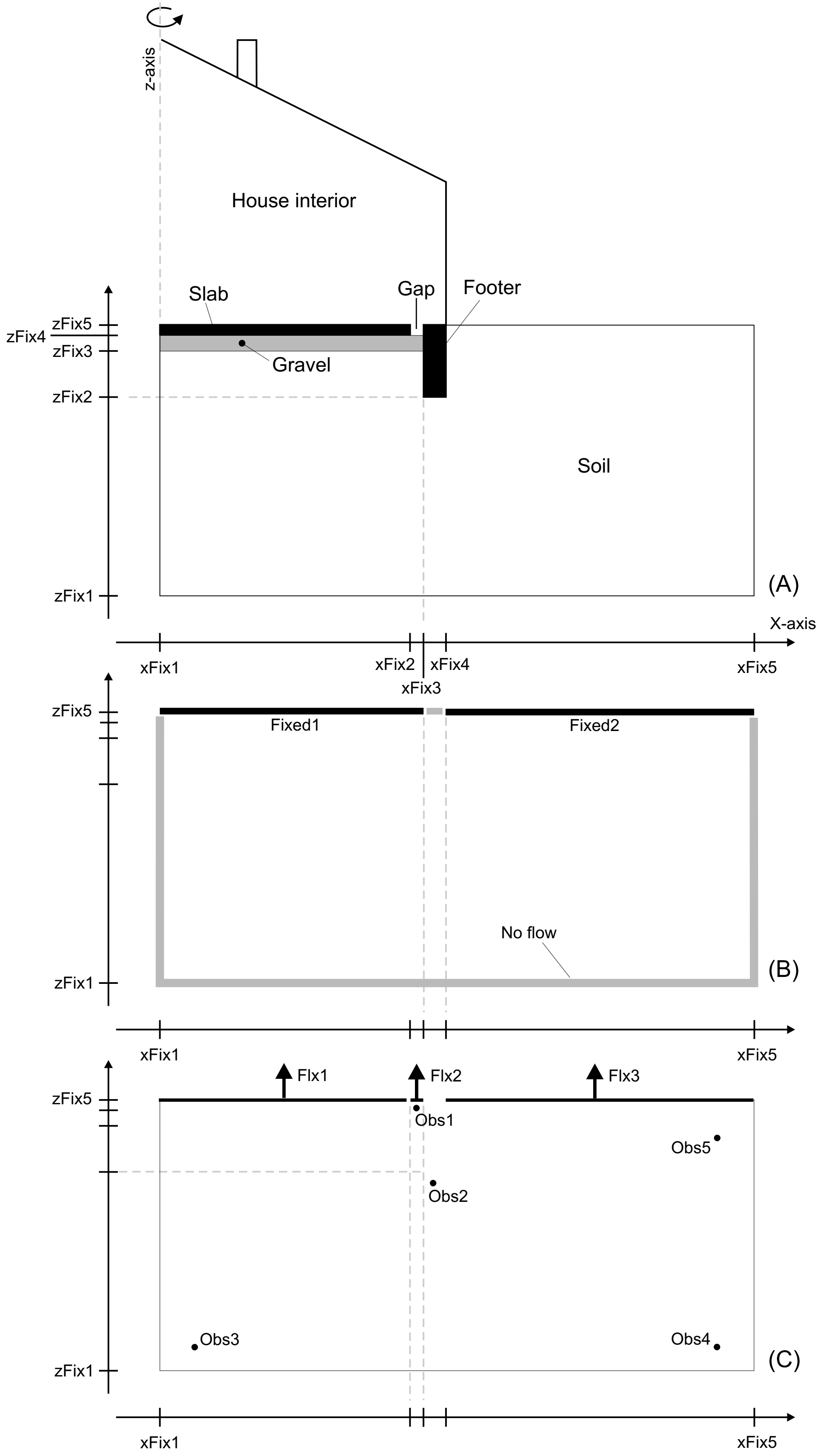


Figure 20. Sketch of slab-on-grade house.

Table 5. Geometry for the slab-on-grade house. All dimensions are in meters.

	Width (x)	Thickness (z)
Soil	`Lx_soil = 20.0`	`Lz_soil = 10.0`
Slab	`Lx_slab = 5.6419`	`Lz_slab = 0.10`
Gravel	(as Lx_slab)	`Lz_gravel = 0.15`
Footer	`Lx_footer = 0.3`	`Lz_footer = 0.80`
Gap	`Lx_gap = 0.003`	(as `Lz_slab`)

```
set_FixVal(zFix1,-Lz_soil); (* z-axis *)
set_FixVal(zFix2,-Lz_footer);
set_FixVal(zFix3,-Lz_slab-Lz_gravel);
set_FixVal(zFix4,-Lz_slab);
set_FixVal(zFix5, 0.00);
```

Observe, that the x-axis goes from 0 to 20 m, and that the z-axis goes from -10 m at the bottom of the soil to 0 at the atmospheric surface.

Boundary conditions

The soil-gas and the radon problems are based on the same types of boundary conditions. As sketched in Figure 20(B) the interface between the house and the slab is set to the fixed-value boundary condition: `fixed1`. Likewise, the atmospheric surface is set to `fixed2`. The rest of the boundary is set to no-flow conditions.

In the soil-gas problem we force the house to be depressurized 1 Pa relative to the atmospheric surface. This is programmed with the assignments:

```
cBC[fixed1]:=-1; (* Pa *)
cBC[fixed2]:= 0;
```

For the radon problem, we set both indoor and outdoor radon concentrations to zero:

```
cBC[fixed1]:= 0; (* Bq/m3 *)
cBC[fixed2]:= 0;
```

Flux probes: `Flx1` etc.

We are interested in the fluxes indicated in Figure 20(C). `Flx1` gives the flux through the slab (i.e. through the concrete). `Flx2` is the flux through the gap. `Flx3` is the flux into the atmosphere. The total entry rate into the house is given by `Flx4`. This is the sum of `Flx1` and `Flx2`. Observe, that all fluxes are taken to be positive if they are in the direction of the z-axis.

Field probes: `Obs1` etc.

To monitor the pressure and radon concentration fields, the probes `Obs1` to `Obs5` are placed as indicated in Figure 20(C). For example, `Obs2` is located just below the footing. The exact positions of the probes can be read from the job file. Notice that `wFixVal` is used to find the location of fix points (see Section 5.4).

Materials

Table 6 lists the materials used in the computations and their parameters. Other constants are: $\mu = 18 \cdot 10^{-6}$ Pa s, $\lambda = 2.09838 \cdot 10^{-6} \mathrm{s}^{-1}$, $L = 0.3$ (see equation 19), and ρ_g=$2.7 \cdot 10^3$ kg m^{-3}.

Table 6. Parameters used in the calculations for the slab-on-grade house: gas permeability (k), radium-226 concentration (A_{Ra}), fraction of emanation f, total porosity (ϵ), volumetric water content (θ_{v}), and bulk diffusivity (D).

	Name of	k	A_{Ra}	f	ϵ	θ_{v}	D
	materials	m^2	Bq kg^{-1}				m^2 s^{-1}
Soil	`mat1`	10^{-11}	40	0.2	0.25	0.2	$4.3 \cdot 10^{-7}$
Slab	`mat2`	10^{-15}	50	0.1	0.20	0	$2.0 \cdot 10^{-8}$
Gravel	`mat3`	$5 \cdot 10^{-9}$	40	0.2	0.40	0	$1.8 \cdot 10^{-6}$
Footer	`mat4`	10^{-15}	0	0	0.20	0	10^{-10}
Gap	`mat5`	$7.5 \cdot 10^{-7}$	0	0	1.00	0	$1.2 \cdot 10^{-5}$

Names are assigned to the different materials. For example, the slab material is called `mat2`. In the job file, the procedure `materials` defines exactly what part of the computational grid that contains `mat2`. This information is used in other parts of the job file. For example, the function where the porosity is defined looks like this:

```
function e_radon(i:itype;j:jtype;k:ktype):datatype;
var ee:datatype;
begin
ee:=0;
  case materials(i,j,k) of
    mat1: ee:=0.25;
    mat2: ee:=0.20;
    mat3: ee:=0.40;
    mat4: ee:=0;
    mat5: ee:=1.0;
  else
    error_std('e_radon','Unknown material');
  end;
  e_radon:=ee;
end;
```

The radon generation rate (G) is calculated from equation 20 on the basis of the given radium concentrations and fractions of emanation. The permeability assigned to the gap is calculated from [An92]:

$$k = \frac{d^2}{12} \tag{73}$$

where d is the width of the gap. For a 3 mm gap, this corresponds to $k = 7.5 \cdot 10^{-7}$ m^2. The other parameters for the gap corresponds to free air.

Job file

The complete job file is shown in Appendix F.

Results

The main results of the computations are output to the `LOG`-file as:

```
The total soil-gas entry into the house is (Flx4)      =
1.6532545E-0005 m3/s

The total radon entry into the house is (Flx4)         =
1.9368863E+0000 Bq/s
```

An extended version of this house simulation can be found in [An99^c]. Figure 15, page 74 shows pressure contours and streamlines.

A F0100prg.dpr

```
program F0100prg;
(* --------------------- RnMod3d jobfile ---------------------- *)
(* Project: User guide example: Steady soil-gas flow, 1D       *)
(* Created: May 24, 1999                                       *)
(* Revised: July 17, 2000                                      *)

{$I R3dirs03}
uses R3Defi03,R3Main03,R3Writ03;

procedure grid;
begin
set_FixVal(xFix1,0.0);
set_FixVal(xFix2,1.0);
set_axis_single(xFix1,xFix2,1,FocusA,1.0);

set_FixVal(yFix1,0.0);
set_FixVal(yFix2,1.0);
set_axis_single(yFix1,yFix2,1,FocusA,1.0);

set_FixVal(zFix1,-3.0);
set_FixVal(zFix2, 0.0);
set_axis_double(zFix1,zFix2,30,30,FocusA,FocusB,1.1,1.1,0.5);
end;

procedure boundary_conditions(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=0;
cBC[fixed2]:=-3.0;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then set_node(i,j,k,fixed1);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then set_node(i,j,k,fixed2);
end;

procedure fluxes(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then update_flxval(Flx1,top,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then update_flxval(Flx2,bottom,i,j,k,plus);
end;
```

```
procedure probes;
var cc:datatype; valid:boolean;
begin
cc:=fieldvalue(0.5,0.5,-1.5,valid);
if not valid then cc:=0.0;
obsval[obs1]:=cc;
end;

function materials(i:itype;j:jtype;k:ktype):mattype;
begin
materials:=mat1;
end;

function e(i:itype;j:jtype;k:ktype):datatype;
begin
e:=0;
end;

function beta(i:itype;j:jtype;k:ktype):datatype;
begin
beta:=0;
end;

function D(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
var mu:datatype;
begin
mu:=17.5e-6;
D:=2e-10/mu;
end;

function G(i:itype;j:jtype;k:ktype):datatype;
begin
G:=0;
end;

function lambda(i:itype;j:jtype;k:ktype):datatype;
begin
lambda:=0;
end;

begin (* main *)
runid                          := '0100';
runtitle                       := 'User guide example: Steady soil-gas flow, 1D';
solution                       := steady;
geometry                       := cartesian3d;
Ly                             := 1.0;
grid_def                       := grid;
force_new_grid_in_every_run    := false;
boundary_conditions_def        := boundary_conditions;
flux_def                       := fluxes;
probe_def                      := probes;
materials_def                  := materials;
e_def                          := e;
beta_def                       := beta;
G_def                          := G;
lambda_def                     := lambda;
D_def                          := D;
initialfield_def               := nil;
import_initialfield            := false;
import_finalfield_guess        := false;
export_field                   := false;
use_fieldbuffer                := no_cBUF;
flowfield                      := none;
flowfactor                     := 1.0;
```

```
import_field_name              := '';
export_field_name              := '';
flowfield_name                 := '';
plotfiles_def                  := nil;
user_procedure_each_iter_def   := nil;
wr_details                     := false;
wr_main_procedure_id           := false;
wr_all_procedure_id            := false;
wr_iteration_line_log          := false;
wr_iteration_line_screen       := true;
wr_residual_during_calc_log    := false;
wr_residual_during_calc_screen := false;
wr_flux_during_calc_log        := false;
wr_flux_during_calc_screen     := false;
wr_probes_during_calc_log      := false;
wr_probes_during_calc_screen   := false;
wr_final_results_log           := true;
wr_final_results_screen        := true;
wr_axes                        := true;
wr_nodes                       := false;
wr_node_numbers                := true;
wr_node_sizes                  := false;
wr_coefficients                := false;
wr_materials_volumes           := true;
warning_priority_log           := war_other;
warning_priority_screen        := war_other;
solver_def                     := Find_better_field_thomas;
scheme                         := exact;
relax_factor                   := 1.0;
flux_convset                   := [flx1,flx2];
probe_convset                  := [obs1];
conv_evaluation_period         := 100;
min_iterations                 := 50;
max_iterations                 := 5000;
max_time                       := 5*60;
max_change                     := 1e-9;
max_residual_sum               := 3e-20;
dtim                           := 0;
BC_running                     := false;
BC_running_update_of_cBCs_def  := nil;
BC_running_min_iterations      := 100;
BC_running_max_residual_sum_before_new_BC := 1e-9;
BC_running_convergence_def     := nil;
wr_BC_running_messages_log     := false;
wr_BC_running_messages_screen  := false;
press_enter_wanted             := true;

run_model;
close_model;
end.
```

B Output: F0100LOG.dat

```
---------------------------------------------------------------------------
Description   : Radon and soil gas transport model
Program name  : RnMod3d (Copyright, Risoe National Laboratory, Denmark)
Version       : Version 0.8 (Sep. 15, 1997 - July 18, 2000)
Documentation : User's Guide to RnMod3d, Risoe-R-1201(EN)
---------------------------------------------------------------------------
* Time =  19-07-2000 09:55:04
**** LOG File  : f0100LOG.dat
```

```
**** RES File  : f0100RES.dat
**** RUN ID    : 0100
**** RUN TITLE : User guide example: Steady soil-gas flow, 1D
---------------------------------------------------------------------------
wr_memory_status
  imax     =      1 jmax   =      1 kmax    =      1
  imaxTot  =    100 jmaxTot=    100 kmaxTot =    200
---------------------------------------------------------------------------
wr_memory_status
  imax     =      3 jmax   =      3 kmax    =     63
  imaxTot  =    100 jmaxTot=    100 kmaxTot =    200
---------------------------------------------------------------------------
wr_count_nodes
* Type and number of nodes incl. boundary conditions :
*        NOP        244
*        free       305
*        fixed1       9 value =  0.00000000000E+0000
*        fixed2       9 value = -3.00000000000E+0000
*        fixed3       0 value =  0.00000000000E+0000
*        fixed4       0 value =  0.00000000000E+0000
*        fixed5       0 value =  0.00000000000E+0000
*     unchanged       0
*          Total    567
---------------------------------------------------------------------------
**** RUN ID    : 0100
**** RUN TITLE : User guide example: Steady soil-gas flow, 1D
***** Converged
---------------------------------------------------------------------------
Iteration =     301 (5000) Time =   0.01 min (  5.00) Residual =  2.07E-0021
* Abs. sum of bs          =  0.00000E+0000
* Abs. sum of residuals   =  2.07079E-0021 (change = -9.99979E-0001)
* Max residual            =  2.11758E-0022 (change = -9.99851E-0001)
* Max residual at (i,j,k) = (   2,   2,  62)
* Max residual at (x,y,z) = ( 5.000E-0001, 5.000E-0001,-1.716E-0002)
Flx1  : J =  1.1428571E-0005 ( change =  9.4091795E-0019 ) Q =  0.0000000E+0000
Flx2  : J =  1.1428571E-0005 ( change =  0.0000000E+0000 ) Q =  0.0000000E+0000
Flx3  : J =  0.0000000E+0000 ( change =  0.0000000E+0000 ) Q =  0.0000000E+0000
Flx4  : J =  0.0000000E+0000 ( change =  0.0000000E+0000 ) Q =  0.0000000E+0000
Flx5  : J =  0.0000000E+0000 ( change =  0.0000000E+0000 ) Q =  0.0000000E+0000
Obs1  : c = -1.5000000E+0000 ( change =  5.7824116E-0019 )
Obs2  : c =  0.0000000E+0000 ( change =  0.0000000E+0000 )
Obs3  : c =  0.0000000E+0000 ( change =  0.0000000E+0000 )
Obs4  : c =  0.0000000E+0000 ( change =  0.0000000E+0000 )
Obs5  : c =  0.0000000E+0000 ( change =  0.0000000E+0000 )
---------------------------------------------------------------------------
wr_material_volumes_etc (volume-averaged field values)
  mat           Avg(conc)           Activity             Volume          N  N_invalid
 mat1  -1.500000000E+0000  0.000000000E+0000  3.000000000E+0000         61        506
  mat           Min(conc)   i   j   k           x           y           z
 mat1  -2.982838178E+0000   2   2  62  5.000E-0001  5.000E-0001 -1.716E-0002
  mat           Max(conc)   i   j   k           x           y           z
 mat1  -1.779212754E-0002   2   2   2  5.000E-0001  5.000E-0001 -2.982E+0000
Total geometric volume      =  3.00000000000E+0000
Total activity              =  0.00000000000E+0000
Overall mean concentration  = -1.50000000000E+0000
---------------------------------------------------------------------------
wr_axes_proc
axis   i        x[i]      x[i+1]        dx[i]         dcdx   dcdxnorm  Fixpts
   x   1     0.00000     0.00000    0.0000000  9.758E-0019 0.00000000   xFix1
   x   2     0.00000     1.00000    1.0000000  1.084E-0019 0.00000000       -
   x   3     1.00000     1.00000    0.0000000  0.000E+0000 0.00000000   xFix2

axis   j        y[j]      y[j+1]        dy[j]         dcdy   dcdynorm  Fixpts
   y   1     0.00000     0.00000    0.0000000  9.758E-0019 0.00000000   yFix1
   y   2     0.00000     1.00000    1.0000000  1.084E-0019 0.00000000       -
   y   3     1.00000     1.00000    0.0000000  0.000E+0000 0.00000000   yFix2

axis   k        z[k]      z[k+1]        dz[k]         dcdz   dcdznorm  Fixpts
   z   1    -3.00000    -3.00000    0.0000000  0.000E+0000 0.00000000   zFix1
   z   2    -3.00000    -2.96442    0.0355843  3.814E-0002 0.69579013       -
   z   3    -2.96442    -2.92372    0.0406923  4.178E-0002 0.76227505       -
   z   4    -2.92372    -2.88085    0.0428727  4.361E-0002 0.79566866       -
   z   5    -2.88085    -2.83650    0.0443531  4.492E-0002 0.81951967       -
   z   6    -2.83650    -2.79101    0.0454874  4.595E-0002 0.83830367       -
   ...
   z  60    -0.11493    -0.07357    0.0413539  4.030E-0002 0.73527063       -
   z  61    -0.07357    -0.03432    0.0392507  3.679E-0002 0.67114101       -
   z  62    -0.03432     0.00000    0.0343236  1.716E-0002 0.31309835       -
   z  63     0.00000     0.00000    0.0000000  0.000E+0000 0.00000000   zFix2
---------------------------------------------------------------------------
* Time =  19-07-2000 09:55:05
```

C F0101prg.dpr

```
program F0101prg;
(* --------------------- RnMod3d jobfile ---------------------- *)
(* Project: User guide example: Steady radon diffusion, 1D      *)
(* Created: May 24, 1999                                         *)
(* Revised: July 17, 2000                                        *)

{$I R3dirs03}
uses R3Defi03,R3Main03,R3Writ03;

procedure grid;
begin
set_FixVal(xFix1,0.0);
set_FixVal(xFix2,1.0);
set_axis_single(xFix1,xFix2,1,FocusA,1.0);

set_FixVal(yFix1,0.0);
set_FixVal(yFix2,1.0);
set_axis_single(yFix1,yFix2,1,FocusA,1.0);

set_FixVal(zFix1,-3.0);
set_FixVal(zFix2, 0.0);
set_axis_double(zFix1,zFix2,30,30,FocusA,FocusB,1.1,1.1,0.5);
end;

procedure boundary_conditions(i:itype;j:jtype;k:ktype);
begin
cBC[fixed2]:=1000;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then set_node(i,j,k,fixed2);
end;

procedure fluxes(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then update_flxval(Flx1,top,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then update_flxval(Flx2,bottom,i,j,k,plus);
end;

procedure probes;
var cc:datatype; valid:boolean;
begin
cc:=fieldvalue(0.5,0.5,-1.5,valid);
if not valid then cc:=0.0;
obsval[obs1]:=cc;
end;

function materials(i:itype;j:jtype;k:ktype):mattype;
begin
materials:=mat1;
end;

function e(i:itype;j:jtype;k:ktype):datatype;
begin
e:=0.3;
end;

function beta(i:itype;j:jtype;k:ktype):datatype;
begin
beta:=e(i,j,k);
end;

function D(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
begin
D:=9.9e-7;
end;

function G(i:itype;j:jtype;k:ktype):datatype;
begin
G:=0.12974983
end;

function lambda(i:itype;j:jtype;k:ktype):datatype;
begin
lambda:=2.09838e-6;
end;

begin (* main *)
runid                          := '0101';
runtitle                       := 'User guide example: Steady radon diffusion, 1D';
solution                       := steady;
geometry                       := cartesian3d;
Ly                             := 1.0;
grid_def                       := grid;
force_new_grid_in_every_run    := false;
boundary_conditions_def        := boundary_conditions;
flux_def                       := fluxes;
probe_def                      := probes;
materials_def                  := materials;
e_def                          := e;
beta_def                       := beta;
G_def                          := G;
lambda_def                     := lambda;
D_def                          := D;
initialfield_def               := nil;
import_initialfield            := false;
import_finalfield_guess        := false;
export_field                   := false;
use_fieldbuffer                := no_cBUF;
```

```
flowfield                      := none;
flowfactor                     := 1.0;
import_field_name              := '';
export_field_name              := '';
flowfield_name                 := '';
plotfiles_def                  := nil;
user_procedure_each_iter_def   := nil;
wr_details                     := false;
wr_main_procedure_id           := false;
wr_all_procedure_id            := false;
wr_iteration_line_log          := false;
wr_iteration_line_screen       := true;
wr_residual_during_calc_log    := false;
wr_residual_during_calc_screen := false;
wr_flux_during_calc_log        := false;
wr_flux_during_calc_screen     := true;
wr_probes_during_calc_log      := false;
wr_probes_during_calc_screen   := false;
wr_final_results_log           := true;
wr_final_results_screen        := true;
wr_axes                        := true;
wr_nodes                       := false;
wr_node_numbers                := true;
wr_node_sizes                  := false;
wr_coefficients                := false;
wr_materials_volumes           := false;
warning_priority_log           := war_other;
warning_priority_screen        := war_other;
solver_def                     := Find_better_field_thomas;
scheme                         := exact;
relax_factor                   := 1.0;
flux_convset                   := [flx2];
probe_convset                  := [obs1];
conv_evaluation_period         := 200;
min_iterations                 := 100;
max_iterations                 := 5000;
max_time                       := 5*60;
max_change                     := 1e-9;
max_residual_sum               := 1e-15;
dtim                           := 0;
BC_running                     := false;
BC_running_update_of_cBCs_def  := nil;
BC_running_min_iterations      := 100;
BC_running_max_residual_sum_before_new_BC := 1e-9;
BC_running_convergence_def     := nil;
wr_BC_running_messages_log     := false;
wr_BC_running_messages_screen  := false;
press_enter_wanted             := true;

run_model;
close_model;
end.
```

D F0102prg.dpr

```
program f0102prg;
(* --------------------- RnMod3d jobfile ---------------------- *)
(* Project: User guide example:                                *)
(*          Steady radon diffusion + advection, 1D.            *)
(* Created: May 24, 1999                                       *)
(* Revised: July 17, 2000                                      *)
```

```
{$I R3dirs03}
uses R3Defi03,R3Main03,R3Writ03;

const lambda_use = 2.09838e-6;
      mu         = 17.5e-6;

var ksoil,cS,dP,velocity,Lz,Dsoil,esoil,Gsoil:datatype;

procedure grid;
begin
set_FixVal(xFix1,0.0);
set_FixVal(xFix2,1.0);
set_axis_single(xFix1,xFix2,1,FocusA,1.0);

set_FixVal(yFix1,0.0);
set_FixVal(yFix2,1.0);
set_axis_single(yFix1,yFix2,1,FocusA,1.0);

set_FixVal(zFix1, 0.0);
set_FixVal(zFix2,  Lz);
set_axis_double(zFix1,zFix2,30,30,FocusA,FocusB,2,2,0.5);
end;

procedure boundary_conditions_Soilgas(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=dP;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then set_node(i,j,k,fixed1);
cBC[fixed2]:=0.0;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then set_node(i,j,k,fixed2);
end;

procedure boundary_conditions_Rn(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=cS;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then set_node(i,j,k,fixed1);
cBC[fixed2]:=0.0;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then set_node(i,j,k,fixed2);
end;

procedure fluxes(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then update_flxval(Flx1,top,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then update_flxval(Flx2,bottom,i,j,k,plus);
end;
```

```
procedure probes;
var cc:datatype; valid:boolean;
begin
cc:=fieldvalue(0.5,0.5,Lz/2,valid);
if not valid then cc:=0.0;
obsval[obs1]:=cc;
end;

function materials(i:itype;j:jtype;k:ktype):mattype;
begin
materials:=mat1;
end;

function e_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
e_soilgas:=0;
end;

function beta_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
beta_soilgas:=0;
end;

function D_soilgas(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
begin
D_soilgas:=ksoil/mu;
end;

function G_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
G_soilgas:=0;
end;

function lambda_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
lambda_soilgas:=0;
end;

function e_Rn(i:itype;j:jtype;k:ktype):datatype;
begin
e_Rn:=esoil;
end;

function beta_Rn(i:itype;j:jtype;k:ktype):datatype;
begin
beta_Rn:=e_Rn(i,j,k);
end;

function D_Rn(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
begin
D_Rn:=Dsoil;
end;

function G_Rn(i:itype;j:jtype;k:ktype):datatype;
begin
G_Rn:=Gsoil;
end;

function lambda_Rn(i:itype;j:jtype;k:ktype):datatype;
begin
lambda_Rn:=lambda_use;
end;

function sinh(x:datatype):datatype;
```

```
var z:datatype;
begin
z:=exp(x);
sinh:=(z-1/z)/2
end;

function cosh(x:datatype):datatype;
var z:datatype;
begin
z:=exp(x);
cosh:=(z+1/z)/2
end;

function c_exact(z:datatype):datatype;
var v,D,cinf,s,alpha,Ld:datatype;
(* See NBS technical note 1139, p. 26 *)
begin
D:=Dsoil;
v:=velocity;
alpha:=v/2/D;
Ld:=sqrt(D/esoil/lambda_use);
s:=1/sqrt(sqr(alpha) + sqr(1/Ld));
cinf:=Gsoil/lambda_use;
c_exact:=cinf*(1-1/sinh(Lz/s)*(exp(v*z/2/D)*sinh((Lz-z)/s) + exp(-v*(Lz-z)/2/D)*sinh(z/s)))+
         cS*exp(v*z/2/D)*sinh((Lz-z)/s) / sinh(Lz/s);
end;

function j_exact:datatype;
var v,D,cinf,s,alpha,Ld:datatype;
(* See NBS technical note 1139, p. 26 *)
begin
D:=Dsoil;
v:=velocity;
alpha:=v/2/D;
Ld:=sqrt(D/esoil/lambda_use);
s:=1/sqrt(sqr(alpha) + sqr(1/Ld));
cinf:=Gsoil/lambda_use;
j_exact:=cinf*(V/2 + D/s/sinh(Lz/s)*(cosh(Lz/s)-exp(v*Lz/2/D)))+
         cS*D/s*exp(v*Lz/2/D)/sinh(Lz/s);
end;


procedure wr_flux;
begin
writeln(LOG,' dP = ',dP:6:2);
writeln(LOG,'RnMod3d Rn flux at z=0: ',FlxVal[flx2].j:16,' Bq/m2/s');
writeln(LOG,'Exact Rn flux at z=0:   ',j_exact:16,' Bq/m2/s');
writeln(LOG,'Deviation:              ',100*(FlxVal[flx2].j-j_exact)/j_exact:16:4,' %');
end;

procedure wr_profile;
var Nsteps,zstart,zstop,dzz,zz,cc:datatype;
    valid:boolean;
begin
(* This procedure finds the field at (x,y,z) where x=0.5m           *)
(* and y=0.5m, and z is looped through the values from top to       *)
(* bottom.                                                          *)
Nsteps:=800;
if not (wFixVal[zFix1].defined and wFixVal[zFix2].defined) then
  error_std('wr_profile','Undefined fixpoints!');
zstart:=wFixVal[zFix1].w;
zstop :=wFixVal[zFix2].w;
dzz:=(zstop-zstart)/Nsteps;
zz :=zstart;
```

```
writeln(RES,'z':12,',','c':12,',','cexact':12);
while (zz<zstop) do
  begin
    cc:=fieldvalue(0.5,0.5,zz,valid);
    if valid then
      writeln(RES,zz:12:6,',',cc:12:6,',',c_exact(zz):12:6);
    zz:=zz+dzz;
  end;
end;

begin (* main *)
runid                          := '0102';
runtitle                       := 'User guide example: Steady Rn diff. and adv.';
solution                       := steady;
geometry                       := cartesian3d;
Ly                             := 1.0;
grid_def                       := grid;
flux_def                       := fluxes;
probe_def                      := probes;
materials_def                  := materials;
flux_convset                   := [flx1,flx2];
probe_convset                  := [obs1];
conv_evaluation_period         := 200;
min_iterations                 := 100;
max_iterations                 := 5000;
wr_axes                        := false;
wr_node_numbers                := false;
wr_materials_volumes           := false;

(* User-defined constants *)
Lz    := 5;      (* Column depth        *)
ksoil := 1e-11;  (* Soil permeability   *)
cS    := 5000;   (* Radon conc. at z=0  *)
dP    := -100;   (* Pressure difference *)
Dsoil := 1e-6;   (* Diffusivity         *)
esoil := 0.3;    (* Porosity            *)
Gsoil := 10000*lambda_use;  (* Generation rate *)

velocity:=ksoil/mu*dP/Lz;

(* First, the soil gas problem *)
boundary_conditions_def      := boundary_conditions_soilgas;
D_def                        := D_soilgas;
e_def                        := e_soilgas;
beta_def                     := beta_soilgas;
G_def                        := G_soilgas;
lambda_def                   := lambda_soilgas;
flowfield                    := export_to_qBUF;
relax_factor                 := 1.9;
max_change                   := 1e-12;
max_residual_sum             := 3e-16;
run_model;

(* Second, the radon problem *)
flowfield                    := import_from_qBUF;
boundary_conditions_def      := boundary_conditions_Rn;
D_def                        := D_Rn;
e_def                        := e_Rn;
beta_def                     := beta_Rn;
G_def                        := G_Rn;
lambda_def                   := lambda_Rn;
relax_factor                 := 1.0;
max_change                   := 1e-12;
max_residual_sum             := 3e-16;
```

```
run_model;

wr_flux;
wr_profile;
close_model;
end.
```

E F0103prg.dpr

```
program f0103prg;
(* --------------------- RnMod3d jobfile ---------------------- *)
(* Project: User guide example: Transient gas flow (1D)        *)
(* Created: October 12, 1999                                    *)
(* Revised: July 17, 2000                                       *)

{$I R3dirs03}
uses R3Defi03,R3Main03,R3Writ03;

const mu     = 17.5e-6;
      easoil = 0.2;
      ksoil  = 1e-14;
      Tper   = 10*3600;
      phi    = pi/2;
      p1     = 3;
      P0     = 100000;
      omega  = 2*pi/Tper;
      Dp     = ksoil*P0/easoil/mu;
      Lz     = 5.0;

      z_obs1 = 0.2; (* probe locations *)
      z_obs2 = 1.0;
      z_obs3 = 2.5;
      z_obs4 = 4.0;
      z_obs5 = 4.8;

procedure grid;
begin
set_FixVal(xFix1,0.0);
set_FixVal(xFix2,1.0);
set_axis_single(xFix1,xFix2,1,FocusA,1.0);

set_FixVal(yFix1,0.0);
set_FixVal(yFix2,1.0);
set_axis_single(yFix1,yFix2,1,FocusA,1.0);

set_FixVal(zFix1,0.0);
set_FixVal(zFix2,Lz);
set_axis_double(zFix1,zFix2,10,10,FocusA,FocusB,2,2,0.5);
end;

procedure boundary_conditions_Soilgas(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=0;
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then set_node(i,j,k,fixed1);

cBC[fixed2]:=0;
if tim>0 then
  cBC[fixed2]:=p1 * sin(omega*tim + phi);
```

```
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then set_node(i,j,k,fixed2);
end;

procedure fluxes(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix1,zFix1) then update_flxval(Flx1,top,i,j,k,plus);
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix2,zFix2) then update_flxval(Flx2,bottom,i,j,k,plus);
end;

procedure probes;
var valid:boolean;
begin
obsval[obs1]:=fieldvalue(0.5,0.5, z_obs1,valid);
obsval[obs2]:=fieldvalue(0.5,0.5, z_obs2,valid);
obsval[obs3]:=fieldvalue(0.5,0.5, z_obs3,valid);
obsval[obs4]:=fieldvalue(0.5,0.5, z_obs4,valid);
obsval[obs5]:=fieldvalue(0.5,0.5, z_obs5,valid);
end;

function materials(i:itype;j:jtype;k:ktype):mattype;
begin
materials:=mat1;
end;

function e_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
e_soilgas:=0;
end;

function beta_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
beta_soilgas:=easoil/P0;
end;

function D_soilgas(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
begin
D_soilgas:=ksoil/mu;
end;

function G_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
G_soilgas:=0;
end;

function lambda_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
lambda_soilgas:=0;
end;

begin (* main *)
runid                        := '0103';
runtitle                     := 'User guide example: Transient gas flow in slab';
solution                     := steady;
geometry                     := cartesian3d;
grid_def                     := grid;
```

```
force_new_grid_in_every_run  := false;
boundary_conditions_def      := boundary_conditions_soilgas;
flux_def                     := fluxes;
probe_def                    := probes;

materials_def                := materials;
D_def                        := D_soilgas;
e_def                        := e_soilgas;
beta_def                     := beta_soilgas;
G_def                        := G_soilgas;
lambda_def                   := lambda_soilgas;

flux_convset                 := [flx2];
probe_convset                := [obs1,obs2,obs3,obs4,obs5];
conv_evaluation_period       := 400;
min_iterations               := 70;
max_iterations               := 10000;
max_time                     := 5*60;
max_change                   := 1e-12;
max_residual_sum             := 3e-9;

wr_iteration_line_screen     := false;
wr_final_results_screen      := false;
wr_axes                      := false;
wr_node_numbers              := false;
wr_materials_volumes         := false;


(* First do steady-state for t=0 *)
tim:=0;
run_model;

(* Then do the unsteady part *)
solution:=unsteady;
dtim:=Tper/500;

(* Write header w. labels *)
writeln('tim/Tper':16,' ','dtim/Tper':16,' ','cBC[fixed2]':16,' ','obsval[obs5]':16);
writeln(RES,'tim':16,', ',
            'hr':16,', ',
            'Patm':16,', ',
            'Q1':16,', ',
            'Q*2':16,', ',
            'P1':16,', ',
            'P2':16,', ',
            'P3':16,', ',
            'P4':16,', ',
            'P5':16);

repeat
  writeln(tim/Tper:16:4,' ',dtim/Tper:16:4,' ',cBC[fixed2]:16:4,' ',obsval[obs5]:16:4);
  writeln(RES,tim:16,', ',
              tim/3600:16,', ',
              cBC[fixed1]:16,', ',
              FlxVal[Flx1].j:16,', ',
              FlxVal[Flx2].j:16,', ',
              obsval[obs1]:16,', ',
              obsval[obs2]:16,', ',
              obsval[obs3]:16,', ',
              obsval[obs4]:16,', ',
              obsval[obs5]:16);
  tim:=tim+dtim;
  run_model;
until tim>4*Tper;
```

```
wr_gridfiles;
close_model;
end.
```

F F0130prg.dpr

```
program F0130prg;
(* -------------------- RnMod3d jobfile --------------------- *)
(* Project: User guide example:                               *)
(*          House simulation (slab-on-grade)                  *)
(* Created: September 26, 1998                                *)
(* Revised: July 18, 2000                                     *)

{$I R3dirs03}
uses R3Defi03,R3Main03,R3Writ03;

const
LambdaRn222    = 2.09838e-6; (* 1/s *)
mu             = 18.0e-6;    (* Pa s *)
rho_g          = 2.7e3;      (* kg/m3 *)
LOstwald       = 0.30;       (* water/gas partitioning *)
deltaP         = -1.0;       (* Pa *)

(* Horizontal (x) dimensions, m *)
Lx_soil        =  20.00;
Lx_slab        =  5.6419;
Lx_footer      =  0.300;
Lx_gap         =  0.003;

(* Vertical (z) dimensions, m *)
Lz_soil        = 10.00;
Lz_slab        =  0.10;
Lz_gravel      =  0.15;
Lz_footer      =  0.80;

(* Radium-226 concentration, Bq/kg *)
ARa_soil       = 40;
ARa_slab       = 50;
ARa_gravel     = 40;
ARa_footing    = 0;
ARa_gap        = 0;

(* Fraction of emanation, - *)
f_soil         = 0.2;
f_slab         = 0.1;
f_gravel       = 0.2;
f_footing      = 0;
f_gap          = 0;

(* Porosity, - *)
etot_soil      = 0.25;
etot_slab      = 0.20;
etot_gravel    = 0.40;
etot_footing   = 0.20;
etot_gap       = 1.00;

(* Volumetric water content, - *)
msat_soil      = 0.20;
msat_slab      = 0.0;
msat_gravel    = 0.0;
```

```
msat_footing   = 0.0;
msat_gap       = 0.0;

(* Bulk diffusivity, m2/s *)
D_soil         = 4.3e-7;
D_slab         = 2.0e-8;
D_gravel       = 1.8e-6;
D_footing      = 1.0e-10;
D_gap          = 1.2e-5;

(* Gas permeability, m2 *)
k_soil         = 1e-11;
k_slab         = 1e-15;
k_gravel       = 5e-9;
k_footing      = 1e-15;
k_gap          = 7.5e-7;

procedure grid;
begin
(* x-axis *)
set_FixVal(xFix1,0.000);
set_FixVal(xFix2,Lx_slab-Lx_gap);
set_FixVal(xFix3,Lx_slab);
set_FixVal(xFix4,Lx_slab+Lx_footer);
set_FixVal(xFix5,Lx_soil);
set_axis_double(xFix1,xFix2,15,15,FocusB,FocusB,2.1,3.0,0.97);
set_axis_single(xFix2,xFix3,5,FocusA,1.5);
set_axis_double(xFix3,xFix4,4,4,FocusA,FocusB,2.0,2.0,0.5);
set_axis_single(xFix4,xFix5,20,FocusA,2.5);

(* z-axis *)
set_FixVal(zFix1,-Lz_soil);
set_FixVal(zFix2,-Lz_footer);
set_FixVal(zFix3,-Lz_slab-Lz_gravel);
set_FixVal(zFix4,-Lz_slab);
set_FixVal(zFix5, 0.00);
set_axis_double(zFix1,zFix2,6,14,FocusA,FocusB,2.0,2.0,0.5);
set_axis_double(zFix2,zFix3,6,8,FocusA,FocusB,1.8,1.8,0.5);
set_axis_double(zFix3,zFix4,15,5,FocusB,FocusB,2.0,2,0.95);
set_axis_single(zFix4,zFix5,4,FocusB,3.0);
end;

procedure boundary_conditions_soilgas(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=deltaP;
cBC[fixed2]:=0;
if in_plane([inside,eqAB], (* Observe: Full slab, not just the gap *)
            i,xFix1,xFix3,
            j,yFix1,yFix2,
            k,zFix5,zFix5) then
  change_node(i,j,k,fixed1,ConX,ConX,ConX,ConX,ConX,ConX);
if in_plane([inside,eqAB], (* Atmospheric surface *)
            i,xFix4,xFix5,
            j,yFix1,yFix2,
            k,zFix5,zFix5) then
  change_node(i,j,k,fixed2,ConX,ConX,ConX,ConX,ConX,ConX);
end;

procedure boundary_conditions_radon(i:itype;j:jtype;k:ktype);
begin
cBC[fixed1]:=0;
cBC[fixed2]:=0;
if in_plane([inside,eqAB], (* Observe: Full slab, not just the gap *)
            i,xFix1,xFix3,
            j,yFix1,yFix2,
            k,zFix5,zFix5) then
  change_node(i,j,k,fixed1,ConX,ConX,ConX,ConX,ConX,ConX);
if in_plane([inside,eqAB], (* Atmospheric surface *)
            i,xFix4,xFix5,
            j,yFix1,yFix2,
            k,zFix5,zFix5) then
  change_node(i,j,k,fixed2,ConX,ConX,ConX,ConX,ConX,ConX);
end;

procedure fluxes(i:itype;j:jtype;k:ktype);
begin
if in_plane([inside,eqAB],
            i,xFix1,xFix2,
            j,yFix1,yFix2,
            k,zFix5,zFix5) then
  begin
    update_flxval(Flx1,bottom,i,j,k,plus); (* slab *)
    update_flxval(Flx4,bottom,i,j,k,plus); (* add to total house entry *)
  end;
if in_plane([inside,eqAB],
            i,xFix2,xFix3,
            j,yFix1,yFix2,
            k,zFix5,zFix5) then
  begin
    update_flxval(Flx2,bottom,i,j,k,plus); (* gap *)
    update_flxval(Flx4,bottom,i,j,k,plus); (* add to total house entry *)
  end;
if in_plane([inside,eqAB],
            i,xFix4,xFix5,
            j,yFix1,yFix2,
            k,zFix5,zFix5) then
  update_flxval(Flx3,bottom,i,j,k,plus); (* atm. surface *)
end; (* fluxes *)

procedure probes;
var c1,dc1:datatype;
    valid1:boolean;
begin
get_fieldvalue2d((wFixVal[xfix2].w+wFixVal[xfix3].w)/2,0.0005,
                  wFixVal[zfix4].w,0.000001,c1,dc1,valid1);
obsval[obs1]:=c1;
get_fieldvalue2d((wFixVal[xfix3].w+wFixVal[xfix4].w)/2,0.001,
                  wFixVal[zfix2].w-0.05,0.02,c1,dc1,valid1);
obsval[obs2]:=c1;
get_fieldvalue2d(wFixVal[xfix1].w+0.3,0.001,
                 wFixVal[zfix1].w+0.3,0.02,c1,dc1,valid1);
obsval[obs3]:=c1;
get_fieldvalue2d(wFixVal[xfix5].w-0.3,0.001,
                 wFixVal[zfix1].w+0.3,0.02,c1,dc1,valid1);
obsval[obs4]:=c1;
get_fieldvalue2d(wFixVal[xfix5].w-0.3,0.001,
                 wFixVal[zfix5].w-0.3,0.02,c1,dc1,valid1);
obsval[obs5]:=c1;
end; (* probes *)

function materials(i:itype;j:jtype;k:ktype):mattype;
var mat:mattype;
begin
mat:=mat1; (* soil *)
if in_region(i,xFix1,xFix2,[inside,eqab],
             j,yFix1,yFix2,[inside,eqab],
             k,zFix4,zFix5,[inside,eqab]) then mat:=mat2; (* slab *)
if in_region(i,xFix1,xFix3,[inside,eqab],
             j,yFix1,yFix2,[inside,eqab],
             k,zFix3,zFix4,[inside,eqab]) then mat:=mat3; (* gravel *)
if in_region(i,xFix3,xFix4,[inside,eqab],
             j,yFix1,yFix2,[inside,eqab],
             k,zFix2,zFix5,[inside,eqab]) then mat:=mat4; (* footing *)
if in_region(i,xFix2,xFix3,[inside,eqab],
             j,yFix1,yFix2,[inside,eqab],
             k,zFix4,zFix5,[inside,eqab]) then mat:=mat5; (* gap *)
materials:=mat;
end; (* materials *)

function m(i:itype;j:jtype;k:ktype):datatype;
var mm:datatype;
begin
mm:=0;
case materials(i,j,k) of
  mat1: mm:=msat_soil;
  mat2: mm:=msat_slab;
  mat3: mm:=msat_gravel;
  mat4: mm:=msat_footing;
  mat5: mm:=msat_gap;
else
  error_std('m','Unknown material');
end;
m:=mm;
end;

function e_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
e_soilgas:=0;
end;

function beta_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
beta_soilgas:=0;
end;

function G_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
G_soilgas:=0;
end;

function Lambda_soilgas(i:itype;j:jtype;k:ktype):datatype;
begin
Lambda_soilgas:=0;
end;

function D_soilgas(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
var kk:datatype;
begin
kk:=0;
case materials(i,j,k) of
  mat1: kk:=k_soil;
  mat2: kk:=k_slab;
  mat3: kk:=k_gravel;
  mat4: kk:=k_footing;
  mat5: kk:=k_gap;
else
    error_std('D_soilgas','Unknown material');
end;
D_soilgas:=kk/mu
end;

function e_radon(i:itype;j:jtype;k:ktype):datatype;
```

```
var ee:datatype;
begin
ee:=0;
case materials(i,j,k) of
  mat1: ee:=etot_soil;
  mat2: ee:=etot_slab;
  mat3: ee:=etot_gravel;
  mat4: ee:=etot_footing;
  mat5: ee:=etot_gap;
else
    error_std('e','Unknown material');
end;
e_radon:=ee;
end;

function beta_radon(i:itype;j:jtype;k:ktype):datatype;
var ea,ew:datatype;
begin
ew:=m(i,j,k)*e_radon(i,j,k);
ea:=e_radon(i,j,k)-ew;
beta_radon:=ea+LOstwald*ew;
end;

function G_radon(i:itype;j:jtype;k:ktype):datatype;
var GG,ee,lam:datatype;
begin
GG:=0;
ee:=e_radon(i,j,k);
lam:=lambdaRn222;
case materials(i,j,k) of
  mat1: GG:=rho_g*(1-ee)/ee*lam*f_soil    * ARa_soil;
  mat2: GG:=rho_g*(1-ee)/ee*lam*f_slab    * ARa_slab;
  mat3: GG:=rho_g*(1-ee)/ee*lam*f_gravel  * ARa_gravel;
  mat4: GG:=rho_g*(1-ee)/ee*lam*f_footing * ARa_footing;
  mat5: GG:=rho_g*(1-ee)/ee*lam*f_gap     * ARa_gap;
else
  error_std('G','Unknown material');
end;
G_radon:=GG;
end;

function Lambda_radon(i:itype;j:jtype;k:ktype):datatype;
begin
Lambda_radon:=LambdaRn222;
end;

function D_radon(dir:dirtype;i:itype;j:jtype;k:ktype):datatype;
var DD:datatype;
begin
DD:=0;
case materials(i,j,k) of
  mat1: DD:=D_soil;
  mat2: DD:=D_slab;
  mat3: DD:=D_gravel;
  mat4: DD:=D_footing;
  mat5: DD:=D_gap;
else
    error_std('D_radon','Unknown material');
end;
D_radon:=DD;
end;

begin (* main *)
runid                        := '0130';
```

```
solution                     := steady;
geometry                     := cylindrical2d;
grid_def                     := grid;
flux_def                     := fluxes;
probe_def                    := probes;
materials_def                := materials;
wr_iteration_line_screen     := true;
wr_flux_during_calc_screen   := true;
wr_axes                      := false;

(* First do the soil-gas simulation *)
runtitle                     := 'Slab-on-grade house (pressure)';
boundary_conditions_def      := boundary_conditions_soilgas;
e_def                        := e_soilgas;
beta_def                     := beta_soilgas;
G_def                        := G_soilgas;
lambda_def                   := lambda_soilgas;
D_def                        := D_soilgas;
import_finalfield_guess      := true;
export_field                 := true;
flowfield                    := export;
import_field_name            := 'PRES00.dat';
export_field_name            := import_field_name;
relax_factor                 := 1.98;
flux_convset                 := [flx1..flx3];
probe_convset                := [obs1..obs4];
conv_evaluation_period       := 300;
min_iterations               := 150;
max_iterations               := 10000;
max_time                     := 60*60;
max_change                   := 1e-10;
max_residual_sum             := 1e-8;
run_model; (* Soil gas run *)

(* Then do the radon simulation *)
runtitle                     := 'Slab-on-grade house (radon)';
boundary_conditions_def      := boundary_conditions_radon;
e_def                        := e_radon;
beta_def                     := beta_radon;
G_def                        := G_radon;
lambda_def                   := lambda_radon;
D_def                        := D_radon;
import_finalfield_guess      := true;
export_field                 := true;
flowfield                    := import;
import_field_name            := 'Rn0000.dat';
export_field_name            := import_field_name;
relax_factor                 := 1.0;
flux_convset                 := [flx1..flx3];
probe_convset                := [obs1..obs4];
conv_evaluation_period       := 300;
min_iterations               := 150;
max_iterations               := 20000;
max_time                     := 60*60;
max_change                   := 1e-10;
max_residual_sum             := 1e-8;
run_model; (* Radon run *)

writeln(LOG,'The total soil-gas entry into the house is (Flx4)      = ',FlxVal[Flx4].Q:16,' m3/s');
writeln(LOG,'The total radon entry into the house is (Flx4)         = ',FlxVal[Flx4].J:16,' Bq/s');
close_model;
end.
```

References

[An92] C.E. Andersen: Entry of soil gas and radon into houses. Risø-R-623(EN), Risø National Laboratory, DK-4000 Roskilde, Denmark, 1992.

[An99^a] C.E. Andersen, D. Albarracín, I. Csige, E.R. van der Graaf, M. Jiránek, B. Rehs, Z. Svoboda, and L. Toro: ERRICCA radon model intercomparison exercise, Risø-R-1120(EN), Risø National Laboratory, DK-4000 Roskilde, Denmark, 1999 (This document can be downloaded from Risø's web-site: `www.risoe.dk`).

[An99^b] C.E. Andersen: Radon-222 exhalation from Danish building materials: H+H Industri A/S results. Risø-R-1135(EN), Risø National Laboratory, DK-4000 Roskilde, Denmark, 1999 (This document can be downloaded from Risø's web-site: `www.risoe.dk`).

[An99^c] C.E. Andersen: Numerical modelling of radon-222 entry into houses: An outline of techniques and results. Presented at *Radon in the living environment*, April 19–23, 1999, Athens, Greece as abstract no. 64. Submitted for publication in the workshop proceedings.

[Bi60] R.B. Bird, W.E. Stewart, and E.N. Lightfoot: Transport phenomena. John Wiley & Sons, 1960.

[Car59] H.S. Carslaw and J.C. Jaeger: Conduction of Heat in Solids. Second edition. Oxford Science Publications, Oxford, 1959.

[Cl79] H.L. Clever (ed.): Solubility data series. Volume 2. Krypton, xenon and radon - gas solubilities. Pergamon Press, 1979.

[Co81] R. Collé, R.J. Rubin, L.I. Knab, and J.M.R. Hutchinson: Radon transport through and exhalation from building materials: A Review and assessment. NBS Technical Note 1139. National Bureau of Standards, U.S. Department of Commerce, 1981.

[Do92] P.A. Domenico and F.W. Schwartz: Physical and Chemical Hydrogeology. John Wiley and Sons, 1990.

[Fe99] J.H. Ferziger and M. Perić: Computational methods for fluid dynamics. Second edition. Springer, Berlin, Germany, 1999.

[He96] R. Helmig: Einführung in die numerischen Methoden der Umweltströmungsmechanik. Institut für Computer Anwendungen im Bauingenieurwesen, Techniche Universität Braunschweig, Germany, 1996.

[Ho94] D.J. Holford: Rn3D: A finite element code for simulating gas flow and radon transport in variably saturated, nonisothermal, porous media: User's manual, version 1.0. Pacific Northwest Laboratory, USA, PNL-8943, 1994.

[Lo87] C.O. Loureiro: Simulation of the steady-state transport of radon from soil into houses with basements under constant negative pressure. LBL-24378, Lawrence Berkeley Laboratory, CA 94720, USA, 1987.

[Na92] W.W. Nazaroff: Radon transport from soil to air. *Review of Geophysics*, vol. 30(2), pp. 137–160, 1992.

[Na88] W.W. Nazaroff, B.A. Moed, and R.G. Sextro: Soil as a source of Indoor Radon: Generation, Migration, and Entry. IN: W.W. Nazaroff and A.V. Nero (eds.). Radon and its Decay Products in Indoor Air. Wiley-Interscience, 1988.

[Pa80] S.V. Patankar: Numerical heat transfer and fluid flow. Hemisphere Publishing Corporation, New York, 1980.

[Pa88] S.V. Patankar: Elliptic systems: Finite-difference method I. IN: W.J. Minkowycz, E.M. Sparrow, G.E. Schneider, and R.H. Pletcher: Handbook of numerical heat transfer. John Wiley & Sons Inc., New York, 1988.

[Ri99] W.J. Riley, A.L. Robinson, A.J. Gadgil, and W.W. Nazaroff: Effects of variable wind speed and direction on radon transport from soil into buildings: Model developement and exploratory results. *Atmospheric Environment*, vol. 33, pp. 2157–2168, 1999.

[Rog91A] V.C. Rogers and K.K. Nielson: Multiphase radon generation and transport in porous material. *Health Physics*, vol. 60, no. 6 (June), pp. 807–815, 1991.

[Rog91B] V.C. Rogers and K.K. Nielson: Correlations for predicting air permeabilities and ^{222}radon diffusion coefficients of soils. *Health Physics*, vol. 61, no. 2 (August), pp. 225–230, 1991.

[Sp98] W.H. van der Spoel: Radon transport in sand: A laboratory study. Ph.D. dissertation, Technical University Eindhoven, the Netherlands, ISBN 90-386-0647-8, 1998.

[Th97] N.R. Thomson, J.F. Sykes, and D. van Vliet: A numerical investigation into factors affecting gas and aqueous phase plumes in the subsurface. *Journal of Contaminant Hydrology*, vol. 28, pp. 39–70, 1997.

[Ve95] H.K. Versteeg and W. Malalasekera: An introduction to computational fluid dynamics. The finite volume method. Longman, Edinburg, England, 1995.

[Wa94] J.W. Washington, A.R. Rose, E.J. Ciolkosz, and R.R. Dobos: Gaseous diffusion and permeability in four soil profiles in central Pennsylvania. *Soil Science*, vol. 157(2), pp. 65–76, 1994.

[Wo92] C.S. Wong, Y-P. Chin, and P.M. Gschwend: Sorption of radon-222 to natural sediments. *Geochimica et Cosmochimica Acta*, vol. 56, pp. 3923–3932, 1992.

Bibliographic Data Sheet **Risø-R-1201(EN)**

Title and author(s)

Radon transport modelling: User's guide to `RnMod3d`

Claus E. Andersen

ISBN	ISSN
87-550-2734-2 (printed edition) 87-550-2733-4 (internet edition)	0106-2840

Dept. or group	Date
Nuclear Safety Research Department	August, 2000

Groups own reg. number(s)	Project/contract No.

Pages	Tables	Illustrations	References
116	6	20	25

Abstract (Max. 2000 char.)

`RnMod3d` is a numerical computer model of soil-gas and radon transport in porous media. It can be used, for example, to study radon entry from soil into houses in response to indoor-outdoor pressure differences or changes in atmospheric pressure. It can also be used for flux calculations of radon from the soil surface or to model radon exhalation from building materials such as concrete.
The finite-volume model is a technical research tool, and it cannot be used meaningfully without good understanding of the involved physical equations. Some understanding of numerical mathematics and the programming language Pascal is also required. Originally, the code was developed for internal use at Risø only. With this guide, however, it should be possible for others to use the model.
Three-dimensional steady-state or transient problems with Darcy flow of soil gas and combined generation, radioactive decay, diffusion and advection of radon can be solved. Moisture is included in the model, and partitioning of radon between air, water and soil grains (adsorption) is taken into account. Most parameters can change in time and space, and transport parameters (diffusivity and permeability) may be anisotropic.
This guide includes benchmark tests based on simple problems with known solutions. `RnMod3d` has also been part of an international model intercomparison exercise based on more complicated problems without known solutions. All tests show that `RnMod3d` gives results of good quality.

Descriptors INIS/EDB

ADVECTION; BUILDING MATERIALS; COMPUTERIZED SIMULATION; COMPUTER PROGRAM DOCUMENTATION; DIFFUSION; ENVIRONMENTAL TRANSPORT; FINITE DIFFERENCE METHOD; GAS FLOW; HOUSES; RADON 222; R CODES; SOILS

RISØ

ISBN 87-550-2734-2
ISBN 87-550-2733-4 (Internet)
ISSN 0106-2840

Copies of this publication
are available from

Risø National Laboratory
Information Service Department
P.O. Box 49
DK-4000 Roskilde
Denmark
Telephone +45 4677 4004
risoe@risoe.dk
Fax +45 4677 4013
Website www.risoe.dk