# Read me for a TOUGH2 postprocessing program in MATLAB

WRITTEN BY

## Antonio Pio Rinaldi

LAWRENCE BERKELEY NATIONAL LABORATORY

 $\begin{array}{c} {\rm July} \ 8, \ 2014 \\ 00{:}25 \end{array}$ 

## Contents

In	trod	uction	4
	Cell	Array	4
		to install	6
1	Hov	v to load TOUGH MESH data in MATLAB	6
	1.1	Example of RMESH use	6
	1.2	Alternative RMESH script to read group (TOUGH2 domains)	
		informations	8
<b>2</b>	Hov	v to extract TOUGH2 printout output	8
	2.1	Example: extracting data from output	9
3	Plot	ting	13
	3.1	Plotting simulation results for a 2D-axisymmetric regular grid	14
	3.2	Plotting simulation results for a 3D regular grid	15
		Plotting simulation results for a 2D non-regular grid	17

# List of Figures

1	Matlab workspace after execution of script RMESH.m	7
2	Example variables after running RMESH.m	8
3	Matlab command prompt and workspace after execution of	
	script READ_DATA	10
4	Variables after running the script READ_DATA	11
5	Variables after running the script READ_DATA with option	
	"Compute Flow"	12
6	Plotting of a 2D axisymmetric domain simulation results. (Left)	
	Plotting with command "contourf" on same discretization as	
	the mesh. (Right) Plotting with command "image" on a finer	
	mesh discretization, and including the fluid flow arrows	15
7	Plotting of a 3D domain simulation results. Upper row:	
	(Left) Plotting with command "image" on a finer mesh dis-	
	cretization on the plane XY at $z=-1810$ . (Right) Plotting with	
	command "image" on a finer mesh discretization on the plane	
	YZ at $x=-12.5$ . Lower row: Plotting with command "slice"	
	with two different view orientation (90 degrees rotation) $\ldots$	16

8	Left: 2D irregular mesh. Right: MATLAB prompt and vari-	
	ables after running the "main.m" file in the example. Note the	
	elevate number of point in x-direction $(Coor\{1,1\})$ , resulting	
	becuase the mesh is not regular	18
9	Plotting of a 2D irregular domain simulation results.	18

## Introduction

This is a sort of "How to.." guide to use the program to load data from TOUGH2 output and to calculate gravity changes and vertical ground displacement. The program loads quite properly data from TOUGH2 outputs (and it works quite well with different EOS modules, I'm using it with EOS1, EOS2, EOS7, EOS7r, EOS7ca, ECO2N/M/H, and ECBM), with different kind of meshgrids (1D, 2D, 2D axysimmetrical, and 3D regular and irregular). The user should be familiar with "cell" array (i.e. a matrix with every elements corresponding to another matrix). Cell are simple to understand, to call a matrix within a cell array simply use braces ({}) instead of round parentheses. E.g.:

- A is a cell array with dimension 5x5

- A{2,5} addresses the matrix in the cell that, for example, has dimension  $20\mathrm{x}20$ 

- A $\{2,5\}(1,20)$  returns the element a row 1 column 20 of the matrix in A $\{2,5\}$ Below the explanation for MATLAB User's Guide.

## Cell Arrays

Cell arrays in MATLAB are multidimensional arrays whose elements are copies of other arrays. A cell array of empty matrices can be created with the cell function. But, more often, cell arrays are created by enclosing a miscellaneous collection of things in curly braces, {}. The curly braces are also used with subscripts to access the contents of various cells. For example,

### A = magic(4); C ={A sum(A) prod(prod(A))}

produces a 1-by-3 cell array. The three cells contain the magic square, the row vector of column sums, and the product of all its elements. When C is displayed, you see

C =

[4x4 double] [1x4 double] [20922789888000]

This is because the first two cells are too large to print in this limited space, but the third cell contains only a single number, 16!, so there is room to print it.

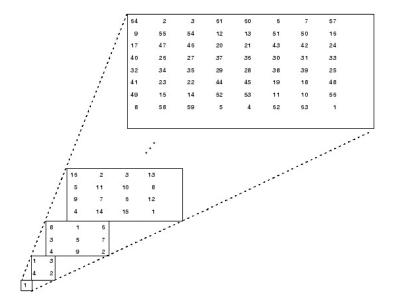
Here are two important points to remember. First, to retrieve the contents of one of the cells, use subscripts in curly braces. For example, C{1} retrieves the magic square and C{3} is 16!. Second, cell arrays contain copies of other arrays, not pointers to those arrays. If you subsequently change A, nothing happens to C.

You can use three-dimensional arrays to store a sequence of matrices of the same size. Cell arrays can be used to store a sequence of matrices of different sizes. For example,

produces a sequence of magic squares of different order:

```
M =
```

[		1]
[	2x2	double]
[	3x3	double]
Γ	4x4	double]
[	5x5	double]
Γ	6x6	double]
Ε	7x7	double]
[	8x8	double]



You can retrieve the 4-by-4 magic square matrix with  $M\{4\}$ 

## How to install

The installation is simple and straightforward. Just place the files in a folder of your choice, and then add the folder to the MATLAB path. In MATLAB menu:

 $File \rightarrow Set \ Path... \rightarrow Add \ with \ Subfolder...$ 

and then you just need to select the folder containing the Matlab scripts and "Save". After doing this you can call the command listed in this "ReadMe" from the MATLAB command window.

## 1 How to load TOUGH MESH data in MAT-LAB

The function to load mesh information is "RMESH" (file RMESH.m). The use is simple and does not require any input:

[coor ID coor\_mesh conne ec e cc]=RMESH()

- coor: this variable returns a cell array {1,*i*}, with *i* representing the direction (i=1:3 for 3D or i=1:2 for for 2D axisymmetric). Each element of the matrix {1,*i*}(*j*) represents the coordinate found by the script in the *i*-direction. Note that these coordinates are not block by block, but simply the system coordinates. It may be of use for plotting and only for a regular mesh!!
- *ID:* this variable returns the name label of each gridblock of the mesh in a vector ID(n. of elem., 5 char).
- coor\_mesh: this variable returns a cell array  $\{1,i\}$ , with *i* representing the direction (i=1:3 for 3D or i=1:2 for for 2D axisymmetric). Each element of the single matrix  $\{1,i\}(j)$  represents the coordinate block-by-block in the *i*-direction. Hence each matrix coor\_mesh $\{1,i\}$  is basically a vector with the value of the *i*-coordinate for each block of the mesh.
- conne, ec, e, and cc are only used internally for the connection table and are used to calculate the proper fluid flow for a single block instead of the flow through the connection.

## 1.1 Example of RMESH use

Let's start from a simple TOUGH2 mesh (8 blocks only). Note that to use RMESH, the file containing the mesh only should be rename as MESH (the

file used i	n this example can be found i	n the folder	"examp	les/RMESH")
ELEME5 NX=	2 NY= 2 NZ= 2			
A11 1	10.1000E+010.1000E+01	0.500	0.500	-0.500
A21 1	10.1000E+010.1000E+01	0.500	0.500	-1.500
A12 1	10.1000E+010.1000E+01	0.500	1.500	-0.500
A22 1	10.1000E+010.1000E+01	0.500	1.500	-1.500
A11 2	10.1000E+010.1000E+01	1.500	0.500	-0.500
A21 2	10.1000E+010.1000E+01	1.500	0.500	-1.500
A12 2	10.1000E+010.1000E+01	1.500	1.500	-0.500
A22 2	10.1000E+010.1000E+01	1.500	1.500	-1.500
CONNE				
A11 1A11 2	10.5000E+000.5000E-	+000.1000E+01		
A11 1A12 1	20.5000E+000.5000E-	+000.1000E+010.0	000E+00	
A11 1A21 1	30.5000E+000.5000E-	+000.1000E+010.1	.000E+01	
A21 1A21 2	10.5000E+000.5000E-	+000.1000E+01		
A21 1A22 1	20.5000E+000.5000E-	+000.1000E+010.0	000E+00	
A12 1A12 2	10.5000E+000.5000E-	+000.1000E+01		
A12 1A22 1	30.5000E+000.5000E-	+000.1000E+010.1	.000E+01	
A22 1A22 2	10.5000E+000.5000E-	+000.1000E+01		
A11 2A12 2	20.5000E+000.5000E-	+000.1000E+010.0	000E+00	
A11 2A21 2	30.5000E+000.5000E-	+000.1000E+010.1	.000E+01	
A21 2A22 2	20.5000E+000.5000E-	+000.1000E+010.0	000E+00	
A12 2A22 2	30.5000E+000.5000E-	+000.1000E+010.1	.000E+01	

In Matlab command prompt simply enter the folder containing the mesh and use the following command:

#### >> [coor ID coor\_mesh]=RMESH();

the workspace should results something like in Fig. 1.

The variable *coor* is a cell array with dimension <1,3 cell> and each matrix (or simply vector) of the cell has only 2 elements (since the mesh is regular with 2 elements for each *i*-direction). For example Fig. 2(left) shows the matrix *coor*{1,1}, i.e. the coordinates in the *x*-direction (*i*=1). Note

File Edit Debug Pa	rallel Desktop V	Window Help
1 🗃 🔏 🐂 🛍 🤊	🖻 ն 🦚 🤊 (	🛛 🥝 /users/rinaldi/Ubuntu/Desktop/programma postproc/exa 💌
Shortcuts 🛃 How to Add	What's New	
Current Directory ×	r ⊷ □ Works	► × マ → □ Command Window
1 1 1 1 1	· · · · · · · · · · · · · · · · · · ·	1 New to MATLAB? Watch this <u>Video</u> , see <u>Demos</u> , or read <u>Getting Sta</u>
Name 🔺	Value	>> [coor ID coor_mesh]=RMESH();
ab ID	<8x5 char>	Reading file MESH
🖸 coor	<1x3 cell>	Reading time: 0.04
🖸 coor_mesh	<1x3 cell>	»
	4 1	

Figure 1: Matlab workspace after execution of script RMESH.m

×₹	+ <b>1</b>					🛃 Variable Edi	tor						
		<b>X</b> 🖣	b 🛍	2	M	- 🔏 - 🛍	Stack:	Base	A V			88	
×₹		coor{	1,1}			×≀⊡ ID		× 7		coor_me	esh{1,1}		
🗄 co	or{1,1} <1x2	double>				ab ID <8x5 char>		\rm со	or_mesh{1,	1} <8x1 do	ouble>		
	1	2	3	4			0		1	2	3	4	
1	0.5000	1.5000			0	ID =		1	0.5000				1
2			0		U	100000		2	0.5000			0	
3			0			A11 1		3	0.5000			0	
4						A21 1		4	0.5000			-0	
5						A12 1 A22 1		5	1.5000			3	
6						A22 1 A11 2		6	1.5000			3	
7			0			A11 2 A21 2	U	7	1.5000			3	
8			0		R.	A12 2		8	1.5000			0	K
9						A22 2		9				0	4
10					•	ALL L	•	10				0	
	$\bigcirc$			).4	•				$\bigcirc$			).4	•

Figure 2: Example variables after running RMESH.m

that if the mesh is not regular (i.e. not regular brick) the variable *coor* must be ignored.

The variable ID is a char vector with dimension  $\langle 8,5 \text{ char} \rangle$  where 8 is the number of elements in the mesh with a 5 character label for the block. The variable ID is shown in Fig. 2(center).

Finally *coor\_mesh* is a cell array with dimension <1,3 cell> and each *i*th element of this cell is a vector of double containing the coordinates of each block in the *i*-direction (8 elements for each vector/matrix). Fig. 2(right) shows the output for *coor\_mesh* for *i*=1.

## 1.2 Alternative RMESH script to read group (TOUGH2 domains) informations

An alternative way to call "RMESH" by including the groups (TOUGH2 domains) is to call the following:

[group coor ID coor\_mesh]=RMESH\_group()

however this script will not provide any information regarding the connection, and the first output variable (group) is a char vector with dimension <N. elements,15 char>. The last 5 character of each elements represent the TOUGH2 domain assigned to that gridblock.

## 2 How to extract TOUGH2 printout output

The printout from a TOUGH2 output files can be extracted with the function "READ\_DATA" (file READ\_DATA.m.). This script is basically the MAT-

LAB version for the fortran ext program (see TOUGH2 website). The usage is simple and require as a input only the filename of the TOUGH2 output. A command can be set to specify whether flow data are needed or not (note that using such an optional command may drastically increase the execution speed). Note also that this program doesn't read the secondary variables if included in the TOUGH2 output. It read just the main printout variables and, if requested, it computes the fluxes in each grid block, since in the TOUGH2 output fluxes are given for the connection and not for the grid block

### [OUT times]=READ\_DATA(file,command)

### **INPUT** variables

- *file*: TOUGH2 output file name (REQUIRED)
- *command:* this command must be the exact string "Compute Flow" to read the flow printout from the TOUGH output (optional)

#### **OUTPUT** variables

- OUT: Cell array  $1 \times L$  where L is the number of the printout found in TOUGH2 output file. Each cell represent a printout in the TOUGH output, and each cell is a  $N \times M$  matrix where N is the number of elements in the meshgrid and M is the number of the variables such as pressure, temperature, etc..
- *times:* Array  $1 \times L$  of the printout time.

## 2.1 Example: extracting data from output

In order to load the file from a TOUGH2 output, simply place the MESH and the output file in the same folder (or enter the folder containing both files). Then on MATLAB command prompt simply use the following command:

#### >> [out times]=READ\_DATA('TOUGH2out');

where out and times are MATLAB variable, and *TOUGH2out* is an output file (example in folder: "examples/loading\_data"). The script will start with reading the MESH, will indicate whether the meshgrid is 1D, 2D, or 3D, and it will start looking for printout within the output file. The printouts are the same as specified in block TIMES in TOUGH2, or at specific time-step as specified in block PARAM. If no problems occur, the MATLAB command prompt and workspace should result as shown in Fig. 3.

In this example, the meshgrid is 2D (axisymmetric) with 2580 elements, and 16 printouts were found in the output file. For each printout is also specified at what time (in seconds) the printout occur. Execution times are also listed: the script takes about 1 s to read the mesh, and 49 s to load the data (on old MacBook Pro laptop).

After reading the MESH, the script will save the mesh information in a file called *mesh\_data.mat* that can be load into matlab. This is particularly useful for large mesh that need time to be read. In this way everytime simulation are performed with the same MESH all the information are already stored in a MATLAB file and do not need to be read from a text file once again (you only need to place the file *mesh\_data.mat* in the same folder with the TOUGH2 output file).

After execution, only two variables are in the workspace: the cell array out and a vector times (Fig. 4). The vector times contains 16 elements corresponding to the 16 times at which a printout occurred (in seconds). out is a  $<1\times16>$  cell array (one element for each printout), and every element of the cell array is a matrix  $<2580\times12>$ . There is a row for every elements in the meshgrid, and the columns represents the element-based variables found

File Edit Debug	Parallel Desktop Wir	MATLAB 7.6.0 (R2008a) dow Help
1 6 % h 🛍	ッ い 🎝 🖬 🗐	/Users/rinaldi/Ubuntu/Desktop/programma postproc/examples/Ic 💌 🖻
Shortcuts 🖪 How to Ad	id 🖪 What's New	
Current Directory	🗙 🔻 🖛 🗖 Workspa	× ₹ → □ Command Window
🖻 🖬 🐿 🛍 🍯	Base 🛊	New to MATLAB? Watch this <u>Video</u> , see <u>Demos</u> , or read <u>Getting Started</u> .
Name 🔺	Value	>> [out times]=READ_DATA('TOUGH2out');
🖸 out 🗄 times	<1x16 cell> <1x16 double>	Reading file MESH Reading time: 0.97
		Mesh 2D with 2580 elements
		Loading data from "TOUGH2OUT" Found printout(1):.259200E+07 13 Found printout(2):.518400E+07 18 Found printout(3):.777600E+07 23 Found printout(4):.10360E+08 27 Found printout(5):.129600E+08 40 Found printout(7):.181440E+08 46 Found printout(1):.25120E+08 50 Found printout(1):.252200E+08 54 Found printout(1):.252200E+08 54 Found printout(1):.25120E+08 66 Found printout(1):.336960E+08 71 Found printout(1):.336960E+08 74 Found printout(1):.336960E+08 74 Found printout(1):.358800E+08 74 Found printout(1):.358800E+08 74 Found printout(1):.358800E+08 74 Found printout(1):.358800E+08 84 Found printout(1):.358800E+08 84 Found printout(1):.358800E+08 84 Found printout(1):.3462880E+08 84 F
	) 4 1	

Figure 3: Matlab command prompt and workspace after execution of script READ\_DATA

in the TOUGH2 printout. The variables depends on the EOS used. In this example with EOS2 and an axisymmetric grid we have:

- Column 1: x-coordinate of the center node;
- Column 2: z-coordinate of the center node;
- Column 3: pore pressure;
- Column 4: temperature;
- Column 5: gas saturation;
- Column 6: CO<sub>2</sub> partial pressure;
- Column 7: average CO<sub>2</sub> mass fraction;
- Column 8: CO<sub>2</sub> mass fraction in gas;
- Column 9: CO<sub>2</sub> mass fraction in liquid;
- Column 10: capillarity pressure;
- Column 11: gas density;

1 6 & 4	1 7 C h 1 1 0	Current Directory: /User	rs/rinaldi/Ubuntu/Desktop/programm	na postproc/examples/loading_data 🔻 🖻
	to Add 💽 What's New			
	× ₹ + □		🛃 Variable Editor	
Current Dir 🕨		~ ~ ~		
🔨 👻 🛊	Na 🔏 🖻	🛍 🎍 🔤 -	🔏 🔹 🚹 Stack: 🛛 Base	
ime ≜	🗙 🗖 🗖 times	□ 5 X	out	× ₹ □ out{1,2}
out	H times <1x16 double>	O out <1x16 cell>		H out{1,2} <2580x12 double>
times	1 2	1	2 3 4	1 2 3 4
	1 2592000 5184000	1 <2580x12 double>	<2580x1 <2580x1 <2580x1	
	2	2		1 12.5000 -2.5100 224930 97.0220 0 2 37.5000 -2.5100 224880 97 0
	3	3		3 62.5000 -2.5100 224520 96.8580 0
	4	4		4 87.5000 -2.5100 221090 95.4340 C
	5	5		5 112.5000 -2.5100 220170 95.0420 0
	6	6		6 137.5000 -2.5100 219160 94.6020 C
	7	7		7 162.5000 -2.5100 217860 94.0230 0
	8	8		8 187.5000 -2.5100 216310 93.3150 C
	9	9		9 212.5000 -2.5100 214560 92.4740 0
	10	10		10 237.5000 -2.5100 212390 91.3510 0
	. 11	11		11 275 -2.5100 206320 87.5550 0
	12	- 12		<ul> <li>12 325 -2.5100 201130 82.8070 0</li> </ul>
	13	13		13 375 -2.5100 192110 74.2450 0
	14	14		14 425 -2.5100 180210 63.3070 C
	15	15		15 475 -2.5100 166740 52.1990 0
	16	16		16 525 -2.5100 148150 38.9540 0
	17	17		17 575 -2.5100 139560 33.8020 C
	18	18		18 625 -2.5100 136590 34.5620 C
	19	19		19 675 -2.5100 135310 33.8060 C
	20	20		20 725 -2.5100 133940 31.9850 0
	21	21		21 787.9000 -2.5100 131750 28.9590 C
	22	22		22 883.2000 -2.5100 128490 24.9320 0
	23	23		23 1028 -2.5100 125780 22.1770 C
	24	24	Y	24 1247 -2.5100 124260 21.2630
	25	25	A	25 1578 -2.5100 124170 21.0730
	26	26	¥	2081 -2.3100 124220 21.0700
		$\bigcirc$	) 4 F	

Figure 4: Variables after running the script READ\_DATA

- Column 12: liquid density;

If a 3D mesh is used, then column 1-3 are for the coordinates (x-, y, and z-, respectively) and the total number of column will be 13. Obviously all these variables will depend on the EOS used in the modeling: for example if the module ECO2N is used with a 3D mesh the total numbers of columns will be 14.

In order to include in the variable *out* also the informations regarding the flow through the connections, the script should be run using the following command:

#### >> [out times]=READ\_DATA('TOUGH2out', 'Compute Flow');

Both the mesh reading and the printout search will take much longer, since connection information have to be read from the MESH and output file. Another MATLAB file will be save (*mesh\_conne.mat*), and execution of the READ\_DATA command will take about 7 seconds to read the MESH and 170 s to read data for the 16 printouts (on old MacBook Pro laptop).

Figure 5 shows an example of variable after execution with "Compute Flow" option. The script will read the connection-based variables within the printouts and will calculated the corresponding flow (e.g. heat, fluid, etc.) in the different direction for each element. For EOS2 there are 6

		ent Directory. /users/maidi/obd	intu/Desktop/programma postproc	/examples/loading_data	
Shortcuts 🛃 How to Ad	i 🛃 What's New				
Current Directory	× * → □		🛃 Variable Editor		
×	Nà 👗 🖻	🛍 🦢 🔟 - 🔏	- 🚹 Stack: Base 🛟		
lame 🔺	× ₹ 🗖 out	× -	οι	ut{1,1}	
out 🛛	O out <1x16 cell>				
times		13 14	15 16 17	18 19 20	21 22 23
	1 <2580x24 double> <258		-859.3010 1.0313e 1.4393e		
	2	2 100 0.0146 112.8992		5.5772e 3.6049e 3.3902e	
	3	3 100 0.1366 112.5904		5.5613e 3.2678e 3.3774e	
	4	4 100 0.1574 109.3213		5.4000e 3.7344e 3.2403e	
	5			5.3579e 1.6354e 3.2049e	
	6	6 200 0.0859 107.5012		5.3109e 1.9545e 3.1650e	
	7			5.2499e 2.3998e 3.1132e	
	8	8 .00 0.1271 104.7390		5.1770e 2.7613e 3.0515e	
	9	9 00 0.1572 102.9855		5.0938e 3.2521e 2.9815e	
	10			4.9893e 5.1087e 2.8950e	
	11			4.6719e 5.4126e 2.6453e	
	- 12			4.3579e 5.7393e 2.4192e	
	13			3.7906e 8.4370e 2.0245e	
	15			3.0455e 1.0076e 1.5366e	
	15			2.2533e 1.2063e 1.0729e	
	16			1.3076e 9.1097e 5.3368e	
	17	17 00 0.1464 16.6033		1.6894e 2.5251e 1.0460e	
	18	18 00 0.0187 18.5987		2.3921e 2.1683e 2.5019e	
	19	19 00 0.0930 17.6732		2.3835e 3.6004e 1.5260e	
	20	20 00 0.1451 15.1351		2.1414e 2.7528e 1.0335e	
	20	21 300 0.1555 10.8770		1.6404e 2.0495e 5.5244e	
	22	22 00 0.1065 5.2377		8.1149e 8.8090e 1.5052e	
	23	23 300 0.0404 1.3899		8.0628e 1.3811e 6.3073e	
	23	24 00 0.0079 0.1520			0 1.3694e1.6110e 2.5352e-
	25	25 .00 7.9848e0.0787			0 -1.3387e1.8230e 3.3457e-
	26	26 00 -3.1421e0.0495			0 -2.1179e1.4494e1.2128e
	27	27 100 -4.9439e 0.0965			0 -2.2054e7.2948e1.1813e.
	27	22 00 -2 2703 0 3290			0 -2.2034e7.2948e1.1813e 0 -9.0058a - 2.0728a4.4362a

Figure 5: Variables after running the script READ\_DATA with option "Compute Flow"

connection-based variables and 2 direction for a 2D-axysimmetric grid, then the total number of printout variables for each element will be 24 (12 from element-based variables + 6 connection-based variables  $\times$  2 directions). For the current example we have:

- Column 13: heat flow x-direction;
- Column 14: heat flow z-direction;
- Column 15: enthalpy flow x-direction;
- Column 16: enthalpy flow z-direction;
- Column 17: fluid (gas+aqueous) flow x-direction;
- Column 18: fluid (gas+aqueous) flow z-direction;
- Column 19: gas flow x-direction;
- Column 20: gas flow z-direction;
- Column 21: aqueous flow x-direction;
- Column 22: aqueous flow z-direction;
- Column 23:  $CO_2$  flow x-direction;
- Column 24: CO<sub>2</sub> flow z-direction;

The number of connection-based variable will also depend on the EOS used. For example for ECO2N the connection printout gives 9 variables, then a total of 27 variables when translated to element-based in a 3D meshgrid. Accounting for the previous 14 (position, pressure, etc.) the total number of columns for ECO2N will be 41!

## 3 Plotting

Nice figures of a TOUGH2 output can be produced by using the MATLAB commands "griddata", "contourf", and "image". The command "griddata" is very useful to redistribute the output from TOUGH2 into a MATLAB matrix, either using the original mesh discretization or refining the mesh for plotting purpose only. This latter case is extremely useful (or better the only solution I found so far) when the TOUGH2 grid is not regular.

# 3.1 Plotting simulation results for a 2D-axisymmetric regular grid

For a regular 2D-axisymmetric we have 2 dimension only. It results pretty easy then to plot the distribution of TOUGH2 output variables on a surface image. The files to run this examples can be found in the folder "examples/plotting/2D\_axisymmetric", including a main file that run what described in this section.

After reading the printouts from a TOUGH2 outfile ( $TOUGHout_2Daxi$ ), and loading the mesh information from file  $mesh_conne.mat$  created after reading the output, we first assigned the coordinates to the vector X and Z,:

```
X=Coor{1,1};
Z=Coor{1,2};
```

the cell *Coor* is included in file *mesh\_conne.mat*. Then we create a MATLAB mesh using the command *"meshgrid"*:

[Xcoord Zcoord]=meshgrid(X,Z);

At this point we are ready to assign the values from TOUGH2 printout to the MATLAB grid. First we extract the variable to plot (block-by-block single vector as described earlier, and in this case the temperature) and assign the x- and z-coordinates block-by-block to the variables  $x\_mesh$  and  $z\_mesh$ :

```
Temperature=out{1,16}(:,4);
x_mesh=out{1,1}(:,1);
z_mesh=out{1,1}(:,2);
```

Finally we use *"griddata"* to assign the value from a block-by-block vector to a matrix:

var\_image=griddata(x\_mesh,z\_mesh,Temperature,Xcoord,Zcoord);

The variable *var\_image* is ready to be plot. Since we are using the same discretization as the TOUGH2 mesh, it is convenient to plot simply by using the command *"contourf"*:

contourf(X,Z,var\_image)

The results is show in Figure 6 (left). However, for a better figure we may want to redistribute the TOUGH2 output on a finer mesh using some interpolation. To do so we first create some linearly distributed coordinates with more point (300 in this case, compared to the 30 for X or 86 for Z of the used TOUGH2 mesh for this example), then we create the MATLAB grid, and interpolated the block-by-block TOUGH2 output in the MATLAB grid by using "griddata" again:

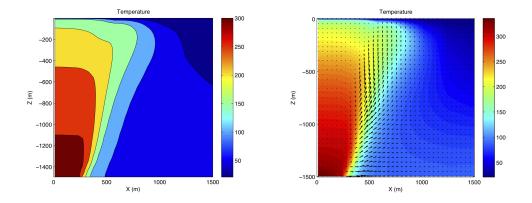


Figure 6: Plotting of a 2D axisymmetric domain simulation results. (Left) Plotting with command "contourf" on same discretization as the mesh. (Right) Plotting with command "image" on a finer mesh discretization, and including the fluid flow arrows.

xlin=linspace(min(X),1500,300); zlin=linspace(min(Z),max(Z),300);

[Xcoord,Zcoord]=meshgrid(xlin,zlin);

```
var_image=griddata(x_mesh,z_mesh,Temperature,Xcoord,Zcoord);
```

In this case we can use the command "image" that produces a much smother

```
image(xlin,zlin,var_image,'Cdatamapping','scaled')
axis image
set(gca,'YDir','normal','XLim',[0 1500])
```

Moreover, we can do a similar interpolate to superimpose the fluid flow to the plotted variable for example, by using the command "quiver" as explained in the file "main.m". The final figure is shown in Figure 6(right).

## 3.2 Plotting simulation results for a 3D regular grid

Plotting a 3D simulation results can be tricky. However, using MATLAB we can easily plot the results in a plane horizontal to the main axes (similar to the 2D case), or we can use the command *"slice"* for a multiple plane plot. The files to run this examples can be found in the folder "examples/plotting/3D", including a main file that run what described in this section.

We load data from the file  $TOUGHout_3D$  (this time no flow) and save the out variables for printout and mesh. We have a single printout this time

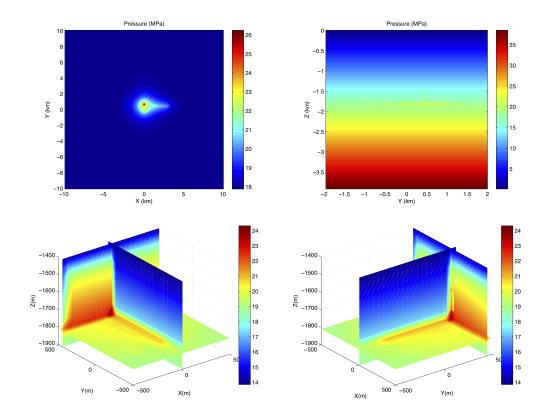


Figure 7: Plotting of a 3D domain simulation results. **Upper row**: (Left) Plotting with command "*image*" on a finer mesh discretization on the plane XY at z=-1810. (Right) Plotting with command "*image*" on a finer mesh discretization on the plane YZ at x=-12.5. Lower row: Plotting with command "*slice*" with two different view orientation (90 degrees rotation)

to make the example faster. At the beginning everything is similar to what we have done before, however in order to plot at fixed depth (plot in XY-plane) we need to find in the block-by-block vector only the blocks that are at fixed depth. Let's say we want to plot at z = -1810 (note that this number has to be one coordinate of the original TOUGH2 mesh)

```
Press_at_fix_z=Pressure(z_mesh==-1810);
x_mesh_at_fix_z=x_mesh(z_mesh==-1810);
y_mesh_at_fix_z=y_mesh(z_mesh==-1810);
```

and after this we can plot refining the mesh and using the command *"image"* as in the previous example. Resulting images are shown in Figure 9 for the plane XY (left) and YZ (right)

Another way to plot a 3D simulation is by using the command "slice",

which also imply the creation of 3D matrix in MATLAB and the use of the command *"griddata3"* that really slow down the process of plotting:

```
[Xco,Yco,Zco] = meshgrid(-500:50:500,-500:50:500,-1900:50:-1400);
var_image=griddata3(x_mesh,y_mesh,z_mesh,Pressure,Xco,Yco,Zco);
```

the advantage is that we do not need to select the variable to plot at a certain depth and/or x/y, the disavantage is that we need to run "griddata3" on the entire domain (in this case with about 30000 elements). As matter of fact the code will run slow, and we need to constrain our MATLAB mesh with 50 m blocks on a restricted domain (first of the above command lines). Then we choose where to slice the domain and run the command "slice" to plot:

xslice=[12.5]; yslice=[450]; zslice=[-1810]; hslice=slice(Xco,Yco,Zco,var\_image,xslice,yslice,zslice);

# 3.3 Plotting simulation results for a 2D non-regular grid

For an irregular mesh the only way to plot a plane figure is using the refined mesh in MATLAB and plot using *"image"* or *"contourf"*. The files to run this examples can be found in the folder "examples/plotting/2D\_irregular", including a main file that run what described in this section.

Fig. 8(left) show the mesh used in this example. After running the "main.m" file something similar to what show in Fig. 8(right) should result in your MATLAB prompt: note the elevate number of point in  $Coor\{1, 1\}$ , that is roughly close to the total number of gridblock (6070). The subroutine RMESH partially failed in reading the mesh: since the mesh is not regular in x-direction, the subroutine cannot calculate the absolute coordinates in X. However, RMESH can still output the x-coordinate block by block (x\_mesh).

In TOUGH2 a 2D-plane mesh can be simply represented by a 3D with a single block in a certain direction (y in this example). As a consequence of this the RMESH will read this mesh as 3D, and produce the output for the Y as well, but that in fact is a vector with a single number. However be careful if plotting the fluxes, because you need to account this kind of simulation as performed on a 3D mesh (see paragraph on "How to load TOUGH2 output").

Finally, since the discretization is not regular, we cannot use the command "meshgrid" using the mesh discretization, but we need to re-discretize the MATLAB mesh as we did previously, and the use the command "griddata" as usual. The resulting plot for this example is shown in Fig. ??.

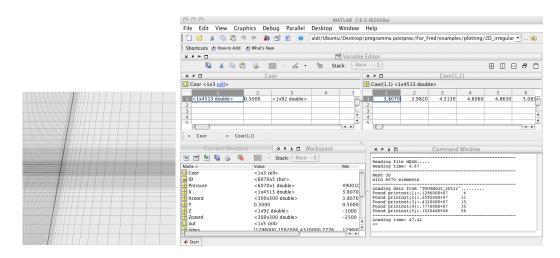


Figure 8: Left: 2D irregular mesh. **Right**: MATLAB prompt and variables after running the "main.m" file in the example. Note the elevate number of point in x-direction  $(Coor\{1,1\})$ , resulting because the mesh is not regular.

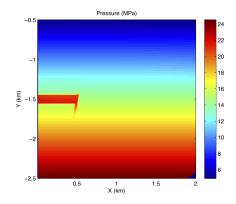


Figure 9: Plotting of a 2D irregular domain simulation results.