CREATING HERMES INPUT-FILES WITH THE GNU m4 MACRO PACKAGE

S. Ramberger
CERN, 1211 Geneva 23, Switzerland

Abstract
A m4-macro package was written to facilitate the development of complex grids for hermes, the meshing preprocessor for ROXIE. A number of macros is provided to create grids of any regular shape easily. These macros are described in detail. The attachment of different grids is shown.

1 Introduction

Complex grids of a big number of elements of a common structure can be created by loops. To keep the hermes preprocessor as simple as possible it was decided not to include loop processing there. Instead m4 was chosen as a pre-pre-processor. The m4-macro package consists of routines necessary to create grids, to connect different grids, to set boundary conditions, and to direct the meshing. Because of the low level on which these routines operate, a concept for naming the keypoints, lines and areas was developed. Based on this naming convention, the naming is unified and helps to simplify the development of new macros. The m4-macro package is based on the tools package which provides basic routines to enhance the language elements provided by m4. We will introduce some of the macros here as well and show their power in small applications.

2 Getting Started

Before you can use the macros in your `*.iron` file you have to load the elements package by a line with the m4 built-in macro `include`

```
include(‘elements.m4’)
```

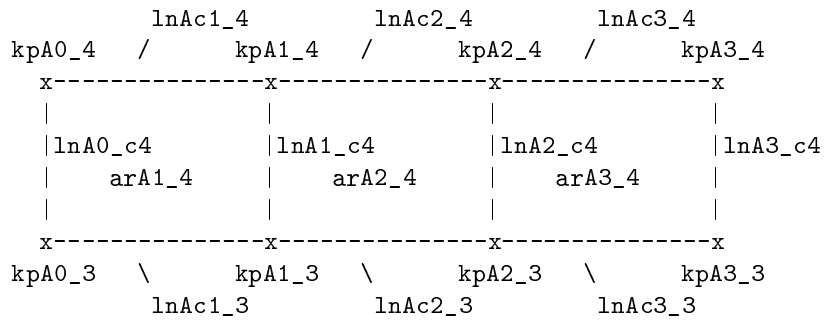
Note the special syntax of m4 starting parameters with a left apostrophe or acute accent ”‘” and ending them with a right apostrophe or grave accent ”’”. Depending on the print they may look different. Syntactical errors in m4 will produce error-messages which, however, are difficult to track down. Most times it is better to check which commands were correctly parsed and to check the line producing the errors.

Type in your macros on the following lines. You may mix m4-macros with normal hermes commands. All hermes commands end with the special character ”;”, whereas m4-macro commands don’t. Since m4 is just a simple text-replacement language it does not change anything in the file it does not recognize. Commands that are spelled incorrectly won’t produce any error message after parsing with m4 since they are not expanded. On the other hand hermes will complain when it finds a command name it doesn’t know.

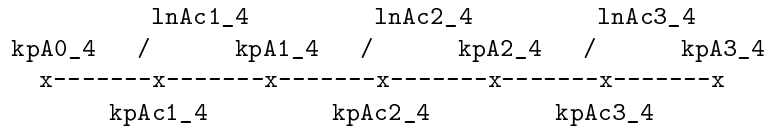
In case you are developing a complex `*.iron` file it may help to preprocess this by calling `m4` from the command line instead of calling `runhermes` for running m4 and hermes together. This will allow you to easily distinguish between m4 and hermes error messages. You should load the macro package by the simple `include` command as shown above not supplying the absolute path. The package is found by setting the environment variable `$M4PATH` to the correct directory. A command `runm4` doing this is included in ROXIE version 6.0.

3 The Naming Convention

To keep the macros as simple as possible it was necessary to introduce a naming convention so that the line macros know about the keypoints they should connect and that the area macros know which lines the area surround. Each keypoint name is encoded by a name which stays the same for all elements, and two indices showing the number of the keypoint in x- and y-direction. For example `kpA0_3` might be the first keypoint of a grid.



Since there are two lines which are connected to each keypoint, we use an additional ”c” at the co-ordinate of the corresponding direction, to specify which line is meant. A line `lnAc1_3` should therefore connect point `kpA0_3` to `kpA1_3`. Intermediate keypoints as they are used for curves will show the ”c” at the same position as the curve connecting it. For a curve with name `lnAc1_4`, the intermediate keypoint in x-direction would have the name `kpAc1_4`.



For all macros point indices and not line indices have to be specified.

4 Creating a Grid

The macro package provides 3 basic macros to create a grid. As in hermes you have to create keypoints, lines, and areas by seperate commands. You will first set the keypoints.

4.1 Creating a Grid of Keypoints

You would issue a command like

```
points(‘A’,‘‘i’,0,10’,‘‘j’,0,5’,‘0.01*[i,j]’,‘xy’)
```

which creates a grid of 10x5 macro-elements with 11x6 keypoints. The syntax is rather special and is further explained in the command reference. Just note that `points` has 5 arguments, each argument seperated by a comma, where the second argument consists again of 3 arguments etc.. You always have to specify the name as first argument. The second argument specifies a counter variable to be counted from 0 to 10 for the x-direction. The third variable specifies the counter for the y-direction going from 0 to 5 and finally you specify the function by which the co-ordinates of the keypoint is calculated. As you can see, you can directly specify the co-ordinates by referring to the counter variables. An additional parameter at the end here specifies that all lines in x- and y-direction should be lines rather than curves with tscreen points. If either x or y is omitted, a curved line is assumed in that direction. So no center points are produced by this macro-command. Note that the apostrophes as shown above have to appear in the same way regarding their type. The counter variable needs additional apostrophes. Failing to supply them in the right way sometimes makes the file not processible for m4. Wrong commands might lead to errors issued by hermes.

4.2 Creating the Lines Connecting the Keypoints

To keep the syntax simple all the commands are kept similar to each other. Creating lines is therefore as simple as creating keypoints:

```
lines(‘A’,‘‘i’,0,10’,‘‘j’,0,5’,‘xy’)
```

Again you have to specify the name of the lines which must be the same as the name of the keypoints to be connected. For the indices in fact you have to specify those for the keypoints and not for the lines. The arguments are basically the same and in most cases show the same indices as used for `points`, but the number of lines created for this 10x5 grid is 11x5+10x6. Since the positions of the keypoints are already defined, there is no need for a formula. Nevertheless the type of the line has to be specified by the additional last argument. The meaning is the same as above. Doing it this way the `lines` macro does not need any knowledge about the lines. But failing to supply the last argument in a way consistent with the point macros will let hermes complain about missing points. Optionally a bias for meshing can be added at the end of the command. See the command reference about how to do that.

4.3 Setting Areas

We are continuing in the same way as above. Another macro defines the areas which have to fit into the line-grid created so far.

```
areas(‘A’,‘‘i’,0,10’,‘‘j’,0,5’,‘BHiron1’)
```

Again the indices comply to the indices given for the keypoints. The macros derive the correct numbering from the point names conforming to the naming convention. Here all the 10x5 areas are filled with iron by the material BHiron1 as defined by the ROXIE B-H curve data-base.

4.4 Defining Mesh-lines

If all the areas should be meshed in the same way the command

```
meshes(‘A’,‘‘i’,0,10’,‘x’,5)
```

will do that for you by subdividing each macro-element by 5 finite elements as specified by the last argument. The third argument specifies the direction in which the counter should be increased. The same should be done in y direction:

```
meshes(‘A’,‘‘j’,0,5’,‘y’,3)
```

Note that the counter-variable is used internally for counting the lines in y direction. The name as defined as first argument is again the name of the (maybe one-dimensional) grid consisting of keypoints and lines. A `c` which might occur directly after the line name is due to the nameing convention and indicates only the index which is changing. It is therefore not a part of the name. Note as well that j is not the y-index itself but the index of the points which should connect the lines according to the naming convention (even though they might not exist). This was done in order to keep the same syntax in all the commands. We can therefore rely on getting the right results when we always specify the correct index of the points which should be the same everywhere.

4.5 Defining Boundaries

The setting of boundaries basically works in the same way as the setting of mesh-lines. Imagine that we want to specify a boundary on the right hand side of the grid. We then have to count the y-index from 0 to 5 but at the same time we want to fix the x-index at 10. This is achieved by setting an offset in the first argument:

```
Dbounds(‘A,10,0’,‘‘j’,0,5’,‘y’)
```

Here we introduce a Dirichlet-boundary for the lines `lnA10_c1` to `lnA10_c5`. In the same way Neumann boundaries `Nbounds` can be used. For further hints see 4.4 (Defining Mesh-lines)

5 Specially Shaped Grids

Creating a curved grid is as simple as creating the rectangular grid above, since the function in the point macros can be as complicated as we like. We are able to use any expression that can be processed by hermes. Since hermes knows how to handle kp-variables, and a lot of expressions this means becomes quite powerful.

5.1 A Circular Grid

To create a circular grid, we simply specify the grid points on a radius and count the angle. Instead of defining the coordinates as `[i,j]` we may use the hermes syntax [i `@` j].

```
points(‘A’,‘‘i’,4,8’,‘‘j’,0,10’,‘0.01*[i@j*Pi/20]’,‘x’)
```

Note that you have to specify `Pi` at least in hermes. It is not a built-in variable.

A possible structure would look like this:

```
include(‘elements.m4’)
Pi=3.1415926535;
points(‘A’,‘‘i’,4,8’,‘‘j’,0,10’,‘0.01*[i@j*Pi/20]’,‘x’)
lines(‘A’,‘‘i’,4,8’,‘‘j’,0,10’,‘x’)
areas(‘A’,‘‘i’,4,8’,‘‘j’,0,10’,BHiron1)
meshes(‘A’,‘‘i’,4,8’,‘x’,2)
meshes(‘A,4,0’,‘‘j’,0,10’,‘y’,3)
```

Note that macro-grids do not need that many subdivisions if there are no structural restrictions. The mesh above could be produced by mesh lines as well:

```
include(‘elements.m4’)
Pi=3.1415926535;
points(‘A’,‘‘i’,4,8,4’,‘‘j’,0,10,10’,‘0.01*[i@j*Pi/20]’,‘x’)
lines(‘A’,‘‘i’,4,8,4’,‘‘j’,0,10,10’,‘x’)
areas(‘A’,‘‘i’,4,8,4’,‘‘j’,0,10,10’,BHiron1)
meshes(‘A’,‘‘i’,4,8,4’,‘x’,8)
meshes(‘A,4,0’,‘‘j’,0,10,10’,‘y’,30)
```

5.2 Orientation

Note that all the grids you are going to define have to obey the orientation rule. For any area to be defined the orientation in which all the lines are specified is in mathematically positive sense which is counter clock-wise. This in turn means for macros that the coordinate system has to be a so called right-hand system where the y-axis is found by turning 90 degrees counter clock-wise from the x-axis.

6 Offsets and Step-Sizes

Offsets and variable step-sizes allow for adaption of the grid numbering. This is the adaption necessary to connect them directly. If direct connection is not possible the `attach` macro helps. It makes special grid structures possible as show in the following.

6.1 Offsets

As seen in the example, offsets are valuable for setting boundaries but they may also be used as in `points` or `lines` macros. The property of the offsets is to influence the numbering of the grid only. The values used in the formulas are corresponding to the real counter values. A macro like

```
points(‘A,3,5’,‘‘i’,3,5’,‘‘j’,6,7’,‘[i,j]’,‘xy’)
```

would create 6 points starting at `kpA6_11=[3,6]` and ending at `kpA8_12=[5,7]`.

6.2 Step-Sizes

All the loops defined by a counter variable and indices allow different formats:

- `‘1’` would mean a counter which cannot be referenced. The name-index is set to 1 plus the offset.
- `‘‘i’3’` is a variable which is fixed to 3. It may be referenced. This means as well that the dimension of the grid is reduced. If you specify both indices in this way, you end up with a single point and no lines or areas.
- `‘‘i’7,9’` is a counter variable whose index becomes 7, 8, and 9. The increment is therefore 1.
- `‘‘i’3,17,7’` is a counter which starts with 3 and is incremented by 7 to become 10 and 17.

The third and fourth version were already used in the examples above. The first and the second are useful for grid lines (one dimensional grids, so to say). If the index is to be used in a formula in the `points` macro, the first one is not applicable.

7 Connecting Grids

Grids can be directly connected if both grids have the same name and fit together obeying the naming convention. The following example creates a three part grid structure

```
include(‘elements.m4’)
points(‘A’,‘‘i’,0,10’,‘‘j’,0,4’,‘[i,0.5*j]’,‘xy’)
points(‘A,0,4’,‘‘i’,0,6’,‘‘j’,0,3’,‘[i,2*j+2]’,‘bxy’)
points(‘A,0,4’,‘‘i’,6,10’,‘‘j’,0,3’,‘[i,j*(3.5-0.25*i)+2]’,‘lbxy’)
lines(‘A’,‘‘i’,0,10’,‘‘j’,0,7’,‘xy’)
areas(‘A’,‘‘i’,0,10’,‘‘j’,0,7’,BHiron1) meshes(‘A’,‘‘i’,0,10’,‘x’,3)
meshes(‘A’,‘‘j’,0,4’,‘y’,2) meshes(‘A’,‘‘j’,4,7’,‘y’,4)
```

The last parameter of the first `points` macro specifies the use of lines in x- and y-direction. In the second `points` macro is set atop of the first macro grid. The last parameter therefore includes a ”b” to get rid of its bottom line since this line is provided by the top line of the first grid. The third `points` macro fits into the space left by the other two and does neither need a bottom nor its left line. The order of the specifier is arbitrary.

8 Attaching Grids

Contrary to a direct connection an attachment can be done for two grids with different names each of them obeying the naming convention seperately. Grids are attached by using the `attach` command. This is the trickiest macro available. It sets two lines of two grids equivalent to each other. Two grids may thereby join a line simply by referencing it as their own. Since we want to reference the old points and lines immediately when creating the points of the new lines, the `attach` macro has to be used before any definition of the new points. Implementation specific details are discussed in 10 (The inner workings of m4) Using

```
attach(‘B’,‘‘ii’,5,15’,‘A’,‘‘j’,0,10’,‘bt’)
```

you may now define a grid B which will use the top-line of grid A as its bottom-line. Both counter variables here specify the x-index so that in the example above, grid B will start at x=5. Every point and line joint with grid A is automatically dereferenced. Note that the order of the specifier in the last parameter fits to the order of the grids to be connected and is not arbitrary. The syntax is different to the other macros as there are two grids with one index each, counted in parallel. Eventhough one index might be counted up while the other is counted down, both counters have to be specified as if they were counted up. The correct counting is defined by the last parameter. An additional parameter may be used to switch off right or left edge connections. This will be shown in a later example. Since the macro for the attachment does not produce any visible code, a comment line was introduced to help with debugging. This can be switched off by defining the symbol nodebug.

```
define(‘nodebug’,‘whatever’)
```

In the next example we show how to connect rectangular grids to a semi-circular one. All the indices here are defined by variables as to make changes of the discretization easier. The variable definition is not included here.

```
points(‘K’,‘‘i’,0,xdisc’,‘‘j’,0,ydisc’,
‘[cent+(eval(2*i-xdisc))/2*grsz,j*grsz]’,‘xy’)
lines(‘K’,‘‘i’,0,xdisc’,‘‘j’,0,ydisc’,‘xy’)
areas(‘K’,‘‘i’,0,xdisc’,‘‘j’,0,ydisc’,0)

attach(‘C’,‘‘ii’,0,ydisc’,‘K’,‘‘j’,0,ydisc’,‘bl’)
attach(‘C’,‘‘ii’,ydisc,mdisc’,‘K,0,ydisc’,‘‘i’,0,xdisc’,‘bt’,‘l’)
attach(‘C’,‘‘ii’,mdisc,fdisc’,‘K,xdisc,0’,‘‘j’,0,ydisc’,‘br’,‘l’)

points(‘C’,‘‘i’,0,fdisc’,‘‘j’,0,2’,
      ‘[cent,0]+[rcoll+j*dcoll@(1-i/fdisc)*Pi]’,‘by’)
lines(‘C’,‘‘i’,0,fdisc’,‘‘j’,0,2’,‘by’)
areas(‘C’,‘‘i’,0,fdisc’,‘‘j’,0,1’,0)
areas(‘C’,‘‘i’,0,fdisc’,‘‘j’,1,2’,BHiron2)
```

The central grid ”K” is rectangular using lines. The attach macros prepare the connection to the the ”C” grid. Since the ”C” grid is put around the ”K” grid, the bottom line of ”C” is connected first to the left line, then to the top line, and then to the right line of the ”K” grid. Since certain attachments would be done twice at the corner points two attach macros show an additional parameter which may either switch off the left or the right corner connection. Now the ”C” grid can be defined without its bottom line (note the index ”b” in the points and lines macros) since this is represented by the ”K” grid. The connection is automatically established. To be sure that the attachment works properly, check the output of m4! The examples should be found in the iron-database of the ROXIE iron testcases.

9 Basic Macros

For more complicated purposes it might be useful to step down to the basic layers and to use the macros on which the elements macro package relies. Some of the commands are defined in the tools macro package and the rest are built-in macros of m4. For further details check the m4 manualas as well.

9.1 Defining non-regular grids

There are two possibilities to design non-regularly spaced grids:

- Variables may be specified including an index, e.g. `Part1, Part2, Part3` etc. and may be referenced by `Part‘’i` where the apostrophe creates the seperation between the name and the index.
- The `arg` macro may be used which selects a parameter from a list, e.g. `arg(‘i’,‘first, second, third’)` according to the index `i`.

9.2 `for` Loops

Most of the macros in this package use loops. Since only recursive procedures are possible with m4, a new ”for” macro was defined to facilitate the programming. A first implementation was copied from the m4-manual but further progress in defining the grid-macros lead to changes: For loops may now have 1 to 4 arguments.

The line

```
for(i,0,10,2,
   ‘dv DB‘’i‘’=1’)
```

may be used to define a number of hermes design-variables which may be refereneced as design-variables from ROXIE.

10 The inner workings of m4

As said before, m4 is a rather simple but powerful text-replacement language. It parses the text given and replaces all occurences of user defined macros by its definition text. This replacement text is parsed again and expanded as before. This feature can be used efficiently to define recursive macros replacing their name with a command text including a call to itself. Built-in functions only provide basic capabilities as for the definition of commands, for conditional execution, integer arithmetic, and system calls. Check the GNU m4 manual pages for further information. In case you can’t find them ask your system administrator or look for them on the world-wide web.

10.1 `define`

Whenever you want to define grids with a readjustable number of elements, define a m4-variable which you may change lateron. This allows as well adjustments of complex structures where the size of one grid depends on the size of another. An example of this kind was shown above.

The line

```
define(‘xdisc’,‘5’)
```

might define discretizations in x direction. Note that this changes the topology of your grid. Since m4 normally does not read any design-variables of ROXIE and hermes is not designed for topology changes, such changes have to be coded in the `*.iron` file. Since m4 does not distinguish between variables and macros also formulas can be defined and used in this way:

```
define(‘d’,‘10’)
define(‘ellipse’,‘[cx,0]+[a*Cos(pha-phe*i/d),b*Sin(pha-phe*i/d)]’)
points(‘E’,‘‘i’,0,d’,‘‘j’,0’,‘ellipse’)
```

The missing variables certainly have to be defined to allow hermes to procede.

10.2 `syscmd` and `esyscmd`

The `syscmd` and `esyscmd` macros allow to start shell-commands. Using this method files may be copied etc. Small perl commands might be executed or even complex meshing programs could be run creating the necessary input for hermes. This method is used as well in a seperate package `design-variables.m4` to make ROXIE design variables accesible from m4. Check this file for further information.

10.3 Material Changes

It is possible to change material data (B-H curves) from ROXIE by using design variables for the material definition. Material 0 is equivalent to BH_air, any number bigger than 0 is interpolated into a string BHiron with the specified number attached. It is therefore possible to change this variable in a certain range to get the corresponing material definitions, e.g. BHiron3 to BHiron6.