# `thSolver` User Guide

v1.2, May 1, 2006

**by Carlos Rosales Fernández**
Heterogeneous Coupled Systems
Institute of High Performance Computing
A*STAR, Singapore

# Preface

`thSolver` is a boundary element method solver for Poisson's equation in microdevice thermal problems. It handles multiple materials and Dirichlet boundary conditions. The discretization is done using isoparametric triangles, and both linear and quadratic elements are available. The domain integral can be calculated by regular direct evaluation, gaussian cubature, or adaptive direct evaluation. The user can choose between an iterative GMRES solver with Jacobi preconditioner, a simple Gauss elimination solver, a Gauss-Jordan solver with full pivoting, or a LU decomposition solver. This manual describes mainly the correct structure of the input files. For any questions or comments contact the author at:

Carlos Rosales Fernández `<carlos@ihpc.a-star.edu.sg>`

Computational Fluid Dynamics Division
Institute of High Performance Computing
1 Science Park Road, #01-01 The Capricorn
Science Park II, Singapore 117528

Singapore, May 1, 2006

# Contents

# 1 Introduction and general remarks

`thSolver` is distributed as a series of C function files, a simple script for compilation called `thSolverComp`, a MCS.PATRAN script for mesh generation called `bem_save.pcl`, and tools to modify the output file to set them in formats which are easily plotted in matlab. The complete list of functions is collected in appendix A.

## 1.1 Quick Install Guide

First of all, unzip the `thSolver.zip` file in a suitable directory. If you chose to keep the directory structure there will be three directories created inside your destination folder:

| | |
|---|---|
| `thSolver` | : Main program directory |
| `thSolver/docs` | : Directory containing this document in pdf form |
| `thSolver/source` | : Directory containing the C source code |
| `thSolver/utils` | : Directory with utilities for pre- and post-processing |
| `utils/break` | : Utility to break data files into several units |
| `utils/cellgen` | : Utility to generate volume meshes for the domain integral |
| `utils/gnuplot` | : Utility to format any file for 3D plotting in gnuplot with pm3d |
| `utils/meshgen` | : Utility to generate planes of points for post-processing |
| `utils/patran2bem` | : Utility to transform mesh from MSC.PATRAN format |
| `utils/thPost` | : Utility to interpolate the temperature values |

In order to install and run `thSolver` all the `.c` and `.h` listed in appendix A are necessary. Make sure all the mentioned files are inside the `thSolver` directory. Then proceed with the following steps.

**Compilation**

The compilation is done using a makefile. Type `make thSolver` to produce the main executable `thSolver`.

The compilation uses gcc with the flag -O3 and several other performance optimization flags. This produces the fastest code for any architecture. If for some reason you don't like this, or you would like to add an architecture-related flag (highly recommended) simply change the flags in the makefile.

## Running the solver

In order to run `thSolver` certain input files are needed. These files have the extension `.bem` and are:

| | |
|---|---|
| `input.bem` | : Main input file |
| `nodes.bem` | : File containing the coordinates of the nodes in the mesh |
| `elems.bem` | : File containing the element connectivity of the mesh |
| `bcs.bem` | : File containing the boundary conditions at every node |
| `elems.domain` | : File containing the element conectivity in the electric problem |
| `solution.domain` | : File containing the solution from the electrical problem |

There are also several optional input files, necessary only when certain options are set in `input.bem`:

| | |
|---|---|
| `internal.bem` | : [opt] File containing the internal points where the temperature is required |
| `cellnodes.bem` | : [opt] File containing the coordinates of the nodes in the domain integral mesh. Used only when the domain integral is done by cubature. |
| `cells.bem` | : [opt] File containing the element connectivity of the domain integral mesh. Used only when the domain integral is done by cubature. |

Notice that the names of all these files are read from `input.bem` and can be changed at will. Only the main input file `input.bem` must keep this name as it is hard coded. Once these files have been properly set – see the following section for details on the format and contents –, one can simply run `thSolver` in the background, since all output is directed to files and there is no interaction with the program while it runs. The output files generated by the program are also divided into those that are produced on every run:

| | |
|---|---|
| `bem.log` | : Main log file, logs program advance and execution time |
| `solution.dat` | : Solution in the format `x y z s` |

And those that are produced only for certain input options:

|  |  |  |
|---|---|---|
| `gmres.log` | : | [opt] GMRES log file, registers the error per iteration |
| `temperature.dat` | : | [opt] Contains the temperature at the required points |
| `tempStats.dat` | : | [opt] Contain the temperature statistics (minimum, maximum, average and standard deviation) for a set of `NT` $\times$ `NT` $\times$ `NT` points. |

## 2 Pre-Processing

This section describes how to set up the necessary input files.

### 2.1 Generating the surface mesh with MSC.PATRAN

To generate the mesh and the boundary conditions using Patran, simply generate the geometry using a structural model and ensuring that the normals are outward-facing (otherwise go to *Elements* and use *Modify→Element→Reverse*).

Then go to *Loads* and use the *Displacement* type to set the boundary conditions in the conductors in the system as:

```
< 1 T 0 >
< 1 0 0 >
```

The first number (1) indicates the potential is given, and the last number can be anything, because it is not used.

Next, use the *Force* type to set the boundary conditions in the interfaces as:

```
< 0 0 interfaceID >
< 0 0 interfaceID >
```

In this case it is important that the first two values are zero, since they are the type of boundary condition (the value of `vBCType[]` in the code) and the right hand side of the boundary equation (the value of `vB[]` in the code).

Once this is done run the command `!!input bem_save.pcl` in Patran's command line, and make sure that you receive a message saying that the compilation has been successful (this should be instantaneous). Then run `bem_save()` in the same command line of Patran. This saves the nodes, elements and boundary conditions in the files `nodes.out`, `elems.out` and `bcs.out`, and also stores information about the mesh (number of nodes, elements and nodes per element) in the file `mesh_info.out`.

## 2.2 Converting MSC.Patran output to the right format

In order to get the input files in the exact format needed for the `thSolver` executable a further step is necessary. Go to the directory called `thSolver/utils/patran2bem` and run:

```
 ./p2b nodeNumber elemNumber elemType scaling bcsNumber reOrder
```

This needs the output files from Patran, `nodes.out`, `elems.out` and `bcs.out`, and yields the correctly formatted `nodes.bem`, `elems.bem` and `bcs.bem`.

This last step seems a bit unnecessary, but Patran uses a different order for the quadratic elements than `thSolver`, and setting the parameter `reOrder` to one transforms the element connectivity to the appropriate format for the program. Also it is handy when one needs to test the same system but scaled at different sizes, because no re-meshing is necessary. Additionally this filter takes care of repeated nodes in the mesh left by Patran, and updates the corresponding references in the element connectivity file, resulting in a more stable system of equations (otherwise there would be repeated equations in the coefficient matrix, making the system not solvable!).

Full help can be obtained on screen by calling the program as: `./p2b -h`.

## 2.3 Generating mesh for domain integral evaluation

When cubature is chosen as the method to solve the domain integral two extra fileas are necessary: `cells.bem` and `cellnodes.bem`. These files can be easily produced by using the utility `cellgen`, found under the directory `thSolver/utils/cellgen`. The utility is called as:

```
 ./cellgen nx ny nz xmin xmax ymin ymax zmin zmax
```

This command generates a regular 3d mesh with limits [(`xmin`,`xmax`) : (`ymin`,`ymax`)

: (`zmin`,`zmax`)]. The nodes are stored into file `cellnodes.dat` and the element connectivity into file `cells.dat`. Example of use:

```
./cellgen 100 20 10 -50 50 -10 10 -25 25
```

This command produces a regular mesh with 8-noded hex elements with 100 points in the x direction, 20 points in the y direction, and 10 points in the z direction. The limits of the mesh are [(-50,50) : (-10:10) : (-25,25)].

The two output files have the required format to be used directly with thSolver. Full ull help can be obtained on screen by calling the program as: `./cellgen -h`.

## 2.4 Generating evaluation points file

Inside `thSolver/utils/meshgen` there is an executable file that generates sets of points in constant planes. This is useful to generate the `internal.bem` files. It is called as:

```
./mesh a constantPlane nx ny xmin xmax ymin ymax r lineNumber
```

This generates a regular 2D mesh with limits [(`xmin`,`xmax`):(`ymin`,`ymax`)] in the plane `constantPlane` = `r`, where `constantPlane` takes the values `x`, `y` or `z`. The parameter `lineNumber` can take values 0 or 1:

`lineNumber` = 0 indicates the line number will not be included in the file
`lineNumber` = 1 indicates the line number will be recorded in the file

Example of use:

```
./mesh 1 z 100 20 -50 50 -10 10 0.0 1
```

Produces a mesh in the plane z = 0.0 in the range x = (-50,50), y = (-10,10), with 100 points in the x direction and 20 in the y direction. The line number is saved to the ouput file starting with 1.

The output file is called 'mesh.dat' for convenience. It can be called sucesively in order to produce a single file with all the necessary points, as long as 'a' contains the correct number of the first element of the plane for each call. If a 21x21 set of points is being generated the first call will be done with `a` = 1, the second with `a` = 442, etc ...

Full help can be obtained on screen by calling the program as: `./mesh -h`.

# 3 Input File Format

The main input file, `input.bem`, is divided in several sections, each of them with a title to increase readability. C++ style comments are allowed in the file, either occupying a line of their own or situated after the data in a line. The same kind of comments may be included in any of the other `.bem` files. Blank lines can be used to make the file easier to read.

## 3.1 Nodes Section

The title `NODES` is typically used for this section of the file, as it contains the total umber of nodes in the mesh and the name of the file with the nodes positions and indexes. This section should look like:

```
NODES
nodeNumber
nodeFilename
```

The data file containing the nodes can have any name (up to 32 characters long), such as the mentioned `nodes.bem`, and must be written in the form:

```
nodeID x y z
```

And the nodes must be sequentially numbered from 1 to `nodeNumber`.

## 3.2 Elements Section

The title `ELEMENTS` is usually given to this section, that must contain the total number of elements in the mesh, the type of elements used, and the name of the file containing the element connectivity information. This section should look like:

```
ELEMENTS
elemNumber
elemType
elemFilename
```

Where `elemType` can be one of the following:

| | |
|---|---|
| `tria3` | : Linear interpolation in triangles (3-noded triangles) |
| `tria6` | : Quadratic interpolation in triangles (6-noded triangles) |

The actual name of the element type can be in uppercase, lowercase, or a mixture of the two, as long as the spelling is correct!

The file containing the element connectivity information must have the following structure:

```
elemID nodeID1 nodeID2 ...  nodeIDM
```

Where `M` is 3 for linear interpolation in triangles and 6 for quadratic interpolation. The elemID must range from 1 to `elemNumber`. Note that all numbers in this file should be integers.

## 3.3 Materials Section

This section contains the number of materials used in the simulation and the values of the electric conductivity and relative permittivity of each of them, usually under the title `MATERIALS`. This section should look like:

```
MATERIALS
matID sigma kappa
```

Where `sigma` is the electrical conductivity of the material, and `kappa` its thermal conductivity.

## 3.4 Interfaces Section

This sections carries the title `INTERFACES`, and contains the information about the interfaces between different materials in the following format:

```
INTERFACES
interfaceNumber
interfaceID mat1 mat2
```

Where `mat1` and `mat2` are the two materials that interface. No support for three material interfaces is provided in the code or the input files.

## 3.5 Problem Section

This section is named `PROBLEM` and contains data related to the domain integral produced by the right hand side term in Poisson's equation. In this case we need to have a previous solution for the electric field in the domain of interest.The section has the following format:

```
PROBLEM
nDomain nDomainElems
xmin ymin zmin
xmax ymax zmax
domainIntegralSolver
{nx ny nz}
{nxi nyi nzi}
{NG nCellNodes nCells}
{cellNodesFilename}
{cellsFilename}
{maxError localError}
domainSolutionFilename
domainElemsFilename
bcsFilename
```

Where all lines within curly brackets are optional and depend on the domain integral solver chosen. `nDomain` and and `nDomainElems` are the number of nodes and elements in the surface mesh for the calculation of the electric field. `(xmin,ymin,zmin)` and `(xmax,ymax,zmax)` are the minimum and maximum values of (x,y,z) in the trap [only rectangular domains are considered]. There are three options for the type of solver for the domain integral `domainIntegralSolver`: `direct`, `cubature`, `adaptive`. Each of them is described in detail below together with the optional lines required by each method. The other three compulsory inputs are the names of the solution file corresponding to the electric field problem, `domainSolutionFilename`; the element file corresponding to the electric problem, `domainElemsFilename`; and the file containing the boundary conditions for the thermal problem, `bcsFilename`.

The file that contains the boundary conditions must be in the following format:

```
nodeID bcType value1 value2
```

A material interface is indicated by a `bcType` of 0, a `value1` of 0, and a `value2` equal to the `interfaceID` where the node belongs.

A dirichlet boundary condition – in this case, temperature given on a surface – is indicated by a `bcType` of 1, and a `value1` equal to the temperature. The `value2` is not used in the code and must be omited from the file.

Let's now see the description of the three different options available for the evaluation of the domain integral and their corresponding parameters.

### 3.5.1 DIRECT domain integral solver

Activated by setting `domainIntegralSolver` to `direct`. It evaluates the integral directly by creating a regular distribution of `nx` × `ny` × `nz` points within the trap defined by `(xmin,ymin,zmin)` and `(xmax,ymax,zmax)`. The electric field values needed for the domain integral are then calculated at `nxi` × `nyi` × `nzi` by tri-linear interpolation, and the integral is calculated as the sum of the contributions from each point multiplied by the volume associated with each point – whihc is always the same, as the point distribution is regular.

### 3.5.2 CUBATURE domain integral solver

Activated by setting `domainIntegralSolver` to `cubature`. It evaluates the integral using gaussian cubature with `NG` × `NG` × `NG` integration points in each of the `nCells` defined by the `nCellNodes` in the file `cellNodesFilename` and the connectivity file `cellsFilename`. The cells are 8-noded hex elements.

### 3.5.3 ADAPTIVE domain integral solver

Activated by setting `domainIntegralSolver` to `adaptive`. Uses direct evaluation in an adaptive mesh to calculate the domain integral. The initial mesh is homogeneous and has `nx` × `ny` × `nz` points. Each point is associated with a surrounding cell that is subdivided progessively in order to find an optimal mesh for the evaluation of the integral. Subsequent refinement stages take place until the global error between succesive domain integral evaluations is less than `maxError`. For each refinement pass only cells where the ratio of the cell's contribution to the average contribution is more than `localError` are subdivided. The maximum number of points allowed for the integral evaluation is defined in the source file `constants.h` and is called `MAXNODES`. It is not recommended to set `localError` too close to one in order to avoid excessive subdivision in each refinement pass. values between 1.2 and 2.0 give good results in general, but this is of course problem-dependent.

## 3.6 Analysis Section

This is the simplest section in the input file. It is usually titled `ANALYSIS`, and contains al the information relative to which solver to use, and what postprocessing to do with the solution. This section must follow the format below:

```
ANALYSIS
solver preCond nInit
analysisType
```

It is not always necessary to include all these parameters in the section. The particular set of them necessary for a calculation depends on the `solver` and `analysisType` requested. Let us examine the section line by line in more detail.

`solver` can be specified as:

| | |
|---|---|
| `gaussBksb` | : Gauss elimination solver with partial pivoting (columns only), does not require the `nInit` parameter |
| `gaussJordan` | : Gauss-Jordan elimination solver with full pivoting, does not require the `nInit` parameter |
| `ludcmp` | : LU decomposition solver, does not require the `nInit` parameter |
| `gmres` | : GMRES solver, requires that the parameters `preCond` and `nInit` are set |

The name of the solver can be in lowercase or uppercase. The parameters `preCond` and `nInit` must be used only with the GMRES solver. `preCond` indicates if a Jacobi pre-conditioner should be used (1) or not (0). It is highly recommended to set this value to 1. `nInit` takes the value 0 if no initial guess is given for the solution, and the number of nodes where the solution is provided otherwise. The file containing the solution must be named `solution.init`, and its format must be:

```
x y z s
```

Where `s` is the solution at the point `(x,y,z)`, and the file has `nInit` rows.

The `analysisType` is an integer that can take the following values:

|   |   |
|---|---|
| 0 | : Calculate only the temperature statistics in the trap |
| 1 | : Calculate both temperature statistics and the temperature at given points |

## 3.7 Internal Points Section

This section is optional, and only necessary when the potential or the electric field are going to be calculated. It is usually title `INTERNALPOINTS` and has the following format:

```
INTERNALPOINTS
internalPointsNumber
internalPointsFilename
```

Where `internalPointsNumber` is the total number of rows in the file `internalPointsFilename`, which has the following structure:

```
pointID x y z
```

And the prefered structure is to keep z fixed, y fixed, x fixed, because in this way slices at different constant z are kept separated and are easy to pos-tprocess later on.

## 3.8 Columns Section

This section is provided in order to be able to calculate properly the electric field and temperature in a channel with cylindrical or square electrode columns. The format is simply:

```
COLUMNS
colType
nColumns
X1 Y1 R1
...
XnCol YnCol RnCol
```

where `nColumns` is the total number of columns in the domain, and `colType` is the type of column (1 for cylindrical columns, 2 for square columns).

In the case of cylindrical columns `Xi` and `Yi` are the positions of the ith column centre and `Ri` is the radius of the ith column.

In the case of square cross-section columns `Xi` and `Yi` are the positions of the ith column centre and `Ri` is half the side length of the ith column.

If `nColumns` is zero the rest of the section will be ignored by the program.

# 4 Post-Processing

This section describes how to use the tools in the `utils` folder in order to manipulate the program's output. There are three main utilities called `break`, `separate` and `thPost`.

## 4.1 break

This program breaks a single output data file into several independent files. This is useful when the output includes calculations of the potential and the field in several planes and it is necessary to have the results from each plane in an independent file in order to plot them. It resides in `thSolver/utils/break` and must be called as:

```
./break fileName fileNumber colNumber rowNumber
```

where `fileName` is the name of the file to break up, `fileNumber` is the number of output files, `colNumber` is the number of columns in the input file, and `rowNumber` is the number of rows in each of the output files. The output files will be named `data1`, `data2`, ..., `datafileNumber`. The input file is not modified.

If called as `./break -h` it will print a short help to the screen.

## 4.2 gnuplot

In the directory `thSolver/utils/gnuplot` there is an executable called `separate` that allows to separate any given data file with an arbitratry number of rows into blocks of a fixed size. This can be used for plotting a set of data corresponding to a plane, for example $z = 0$, in gnuplot with the pm3d option. The utility is called as:

```
./separate filename nRows nBlock nCols
```

Where `filename` is the name of the file to separate into blocks, `nRows` is the total number of rows in the file, `nBlock` is the size of the blocks to make, and nCols the number of columns in the file. This produces as output the file `temp.gnu` with the block-separated data that has a blank line every nBlock lines of the original file.

Let's assume that we generated the internal points file using the utility `meshgen` described in section [2.4], and that we asked meshgen to generate 100 points in the x direction and 20 in the y direction as in the example of use given. Because the total number of points is $100 \times 20 = 2000$ and we only have 4 columns (x,y,z,T) in the temperature data file we call separate as:

```
 ./separate temperature.dat 2000 20 4
```

The output file can be now used in gnuplot to produce a density plot. First run gnuplot by calling it from the command line:

```
 #gnuplot
```

Once inside gnuplot type:

```
 #gnuplot> set pm3d
 #gnuplot> splot 'temp.gnu' u 1:2:3:4 w pm3d
```

if you only want a xy surface plot then use instead:

```
 #gnuplot> set pm3d map
 #gnuplot> splot 'temp.gnu' u 1:2:4 w pm3d
```

If called as `./separate -h` it will print a short help to the screen.

## 4.3 thPost

In order to process the file `temperature.dat` you must go into directory `thSolver/utils/thPost` and run:

```
./thpost n col
```

Where `n` is the number of points in each spatial direction and `col` is the index of the constant column. The program uses bilinear interpolation to produce a new file of size $(2n-1) \times (2n-1) \times (2n-1)$ in order to get nicer looking surface plots with gnuplot when using the pm3d option. The output files are:

| | | |
|---|---|---|
| `gnu-temperature.dat` | : | Original temperature values in gnuplot format |
| `gnu-temperature-big.dat` | : | Interpolated temperature values in gnuplot format |

This is useful for later post-processing in gnuplot, since the temperature values can now be plotted using the pm3d option as detailed in the previous section but will loo much smoother.

## 5 Example Files

This section contains an example of a problem containing two materials. This is the listing for the file `input.bem` :

```
NODES
6000           //Total number of nodes
nodes.bem      //Nodes data file

ELEMENTS
2400           //Total number of elements
tria6          //Quadratic interpolation in triangles
elems.bem      //Elements data file

MATERIALS
2              //Total number of different materials
1 1.4E-4 6.0E-1//Electrical and thermal Conductivities for material 1 (fluid)
2 0.0E+0 1.0E+1//Electrical and thermal Conductivities for material 2 (wall)

INTERFACES
1              //Number of interfaces
1 1 2          //Only one interface (between materials 1 and 2)

PROBLEM
8336 3904                //nDomain nDomainElems
-5.0e-5 -5.0e-5 -5.0e-5 //xmin ymin zmin
5.0e-5 5.0e-5 5.0e-5    //xmax ymax zmax
cubature                //domainIntegralSolver
1  8000  6589           //NG  nCellNodes  nCells
cellnodes.bem           //Nodes for domain mesh
cells.bem               //Element connectivity for domain mesh
solution.domain         //Solution of the electric problem
elems.domain            //Element conectivity for electric problem
bcs.bem                 //Boundary conditions data file

ANALYSIS
gaussBksb         //Gauss solver with backsubstitution

INTERNALPOINTS
1323           //Total number of internal points for post-processing
```

```
internal.bem   //Internal points data file

COLUMNS
0              //Using flat electrodes, no columns
```

The first data file referenced in `input.bem` is the file containing the nodes. The file `nodes.bem` could look like:

```
//nodeID   x              y              z
1         -2.500000e-05  -2.500000e-05  -2.500000e-05
2         -5.000000e-06  -2.500000e-05  -2.500000e-05
...
5999      -3.899404e-05   2.786016e-05   2.500000e-05
6000      -1.476191e-05   1.842516e-05   2.500000e-05
```

The file `elems.bem` would be:

```
//elemID node1 node2 node3 node4 node5 node6
1        2     335   58    344   59    336
2        59    468   125   345   60    337
...
2399     4325  2145  2125  4324  5987  5988
2400     2145  4321  5647  5780  5999  6000
```

The boundary conditions file `bcs.bem` has the following format:

```
//nodeID bcType value1 value2
1        1      300           //Given temperature
2        1      300
...
5999     0      0      1      //Material interface
6000     0      0      1
```

Finally, the `internal.bem` file is:

```
//pointID  x          y          z
1         -5.00E-05  -5.00E-05   1.00E-06
2         -4.50E-05  -5.00E-05   1.00E-06
...
1322       1.25E-05   4.50E-05   5.00E-05
1323       1.25E-05   5.00E-05   5.00E-05
```

## A Function Listing

| | |
|---|---|
| Directory | : `thSolver/source` |
| Source Files | : 105 |
| Compilation Script | : `make thSolver` |

```
bodyForce_tria3.c        getNormal_tria3.c        temperatureCubat_tria3.h
bodyForce_tria3.h        getNormal_tria6.c        temperatureCubat_tria6.c
bodyForce_tria6.c        initRes.c                temperatureCubat_tria6.h
bodyForce_tria6.h        initRes.h                thermalFormA_tria3.c
bodyForceAdapt_tria3.c   integral_tria3.h         thermalFormA_tria3.h
bodyForceAdapt_tria3.h   integral_tria6.h         thermalFormA_tria6.c
bodyForceAdapt_tria6.c   interp3d.c               thermalFormA_tria6.h
bodyForceAdapt_tria6.h   intG_tria3.c             thermalPostProcess_tria3.c
bodyForceCubat_tria3.c   intG_tria6.c             thermalPostProcess_tria3.h
bodyForceCubat_tria3.h   intH_tria3.c             thermalPostProcess_tria6.c
bodyForceCubat_tria6.c   intH_tria6.c             thermalPostProcess_tria6.h
bodyForceCubat_tria6.h   intSingularG_tria3.c     thermalPostProcessAdapt_tria3.c
comFilter.c              intSingularG_tria6.c     thermalPostProcessAdapt_tria3.h
constants.h              intSingularH_tria3.c     thermalPostProcessAdapt_tria6.c
depThermal.c             intSingularH_tria6.c     thermalPostProcessAdapt_tria6.h
depThermal.h             iterGMRES.c              thermalPostProcessCol_tria3.c
dotProd.c                iterGMRES.h              thermalPostProcessCol_tria3.h
doubleMatrix.c           L2Norm.c                 thermalPostProcessCol_tria6.c
doublePointer.c          lubksb.c                 thermalPostProcessCol_tria6.h
doubleTensor.c           ludcmp.c                 thermalPostProcessColAdapt_tria3.c
doubleVector.c           makefile                 thermalPostProcessColAdapt_tria3.h
elemType.c               matVectProd.c            thermalPostProcessColAdapt_tria6.c
errorHandler.c           shape_tria3.c            thermalPostProcessColAdapt_tria6.h
field_tria3.c            shape_tria6.c            thermalPostProcessColCubat_tria3.c
field_tria3.h            solverGMRES.c            thermalPostProcessColCubat_tria3.h
field_tria6.c            solverGMRES.h            thermalPostProcessColCubat_tria6.c
field_tria6.h            temperature_tria3.c      thermalPostProcessColCubat_tria6.h
freeDoubleMatrix.c       temperature_tria3.h      thermalPostProcessCubat_tria3.c
freeDoublePointer.c      temperature_tria6.c      thermalPostProcessCubat_tria3.h
freeDoubleTensor.c       temperature_tria6.h      thermalPostProcessCubat_tria6.c
freeUintMatrix.c         temperatureAdapt_tria3.c thermalPostProcessCubat_tria6.h
gaussBksb.c              temperatureAdapt_tria3.h uintMatrix.c
gaussData.h              temperatureAdapt_tria6.c uintVector.c
gaussJordan.c            temperatureAdapt_tria6.h X2L_tria3.c
```

| | | |
|---|---|---|
| `gaussJordan.h` | `temperatureCubat_tria3.c` | `X2L_tria6.c` |

---

| | |
|---|---|
| Directory | : `thSolver/utils/break` |
| Source Files | : 2 |
| Compilation Script | : `breakComp` |

| | |
|---|---|
| `break.c` | `errorHandler.c` |

---

| | |
|---|---|
| Directory | : `thSolver/utils/cellgen` |
| Source Files | : 2 |
| Compilation Script | : `cellComp` |

| | |
|---|---|
| `cellgen.c` | `errorHandler.c` |

---

| | |
|---|---|
| Directory | : `thSolver/utils/gnuplot` |
| Source Files | : 2 |
| Compilation Script | : `sepComp` |

| | |
|---|---|
| `separate.c` | `errorHandler.c` |

---

| | |
|---|---|
| Directory | : `thSolver/utils/meshgen` |
| Source Files | : 2 |
| Compilation Script | : `meshComp` |

| | |
|---|---|
| `meshgen.c` | `errorHandler.c` |

---

| | |
|---|---|
| Directory | : `thSolver/utils/patran2bem` |
| Source Files | : 5 |
| Compilation Script | : `p2bComp` |

| | | |
|---|---|---|
| `bem_save.pcl` | `doubleMatrix.c` | `errorHandler.c` |
| `freeDoubleMatrix.c` | `patran2bem.c` | |

Notice that the file `bem_save.pcl` must be in the same directory from which Patran is executed, so copy the file over as necessary.

---

| | |
|---|---|
| Directory | : `thSolver/utils/thPost` |
| Source Files | : 5 |
| Compilation Script | : `thPostComp` |

```
doubleMatrix.c         doubleVector.c          errorHandler.c
freeDoubleMatrix.c     thPost.c
```

---

## B Element Library

We have used the following convention for the nodes in the elements:

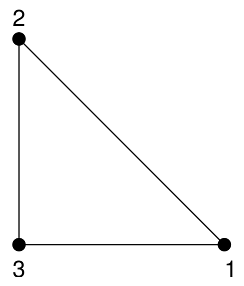

Figure 1: Linear interpolation.

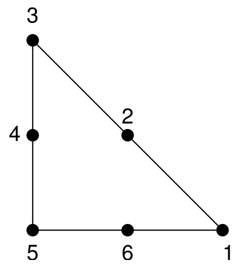

Figure 2: Quadratic interpolation.

Using this convention the shape functions for the linear case are simply given by:

$$N_1 = L_1 \tag{1}$$
$$N_2 = L_2 \tag{2}$$
$$N_3 = 1 - L_1 - L_2 \tag{3}$$

Under this convention the quadratic interpolation shape functions are given by:

$$N_1 = L_1(2L_1 - 1) \tag{4}$$
$$N_2 = 4L_1L_2 \tag{5}$$
$$N_3 = L_2(2L_2 - 1) \tag{6}$$
$$N_4 = 4L2(1 - L_1 - L_2) \tag{7}$$
$$N_5 = L_3[1 - 2(L_1 + L_2)] \tag{8}$$
$$N_6 = 4L_1(1 - L_1 - L_2) \tag{9}$$

For the domain integration cells we use 8-noded hex elements as shown in Figure 3. Using this convention the tri-linear interpolation shape functions are given by:

$$N_1 = \frac{1}{8}(1 - L_1)(1 - L_2)(1 - L_3) \tag{10}$$

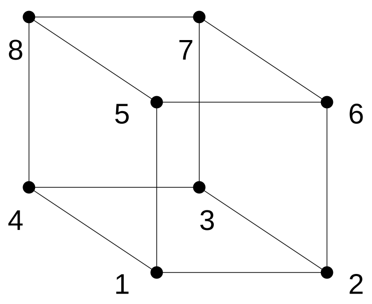

Figure 3: Hex elements used for tri-linear interpolation.

$$
\begin{aligned}
N_2 &= \frac{1}{8}(1+L_1)(1-L_2)(1-L_3) && (11)\\
N_3 &= \frac{1}{8}(1+L_1)(1+L_2)(1-L_3) && (12)\\
N_4 &= \frac{1}{8}(1-L_1)(1+L_2)(1-L_3) && (13)\\
N_5 &= \frac{1}{8}(1-L_1)(1-L_2)(1+L_3) && (14)\\
N_6 &= \frac{1}{8}(1+L_1)(1-L_2)(1+L_3) && (15)\\
N_7 &= \frac{1}{8}(1+L_1)(1+L_2)(1+L_3) && (16)\\
N_8 &= \frac{1}{8}(1-L_1)(1+L_2)(1+L_3) && (17)\\
& && (18)
\end{aligned}
$$

# C Numerical Integration

## Regular integrals

For non-singular integrals the integration is done using gaussian quadrature with NG integration points per element as in:

$$\int_{-1}^{1} F(L_1, L_2) dL_1 dL_2 \approx \sum_{i=1}^{NG} F(L_1^i, L_2^i) w_i \tag{19}$$

By default the program uses 7 points per triangular element. Tests where done with 16 and 64 points per element and the accuracy of the results was not affected, so 7 points were kept for speed.

## Weakly singular integrals

For weakly singular integrals the integration is done through a regularization transformation that eliminates the singularity. The element is transformed into a triangle with a singularity in node 1 and then into a degenerate square. In the degenerate square we can use Gauss-Jacobi integration to integrate getting rid of the singularity. See Figure 3 for the transformation.

$$\int_{-1}^{1} F(L_1, L_2)(1 + L_2) dL_1 dL_2 \approx \sum_{i=1}^{NG} \sum_{j=1}^{NG} F(L_1^i, L_2^j) w_i^{\text{Gauss}} w_j^{\text{Gauss}-\text{Jacobi}} \tag{20}$$

Notice that the Gauss-Jacobi integration is necessary in only one of the directions, so in the other the standard Gauss quadrature on a line is used and the product of the two provides the correct integration as indicated by the expression above.

In the case of quadratic triangles when the singular point is on an edge of the triangle rather than on a vertex the triangle is divided in two and then the same regularization procedure is applied to both subtriangles as shown in Figure 4.

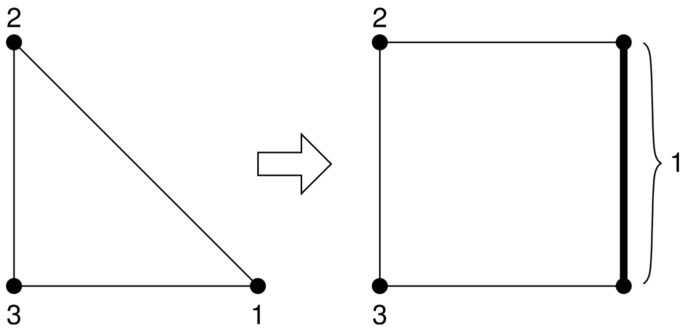

Figure 4: Regularization transformation for weakly singular integrals (linear case).

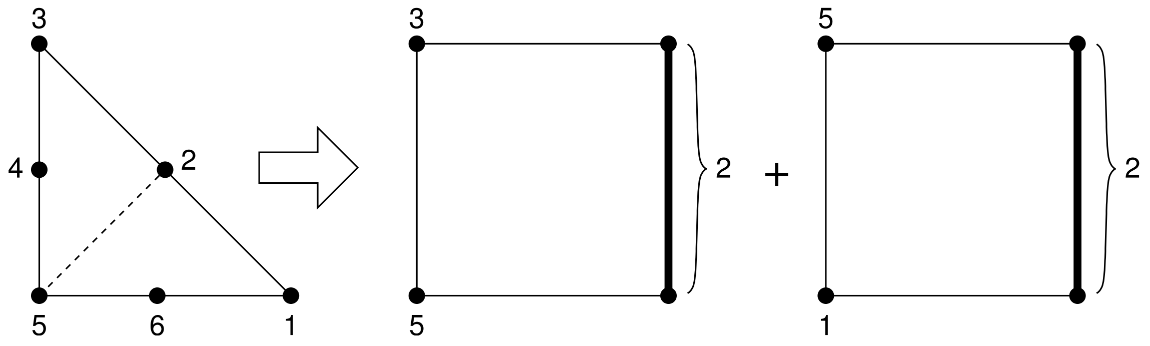

Figure 5: Regularization transformation for weakly singular integrals (quadratic case).

## Strongly singular integrals

For strongly singular integrals we subdivide the triangle progressively in up to `NSUBDIVISIONS` –found in file `constants.h`– subsequent divisions, and integrate using standard gausian quadrature in each subtriangle except the closest to the singular point, which is neglected (it can be shown that the integrand goes to zero very close to the singularity point). The subdivision process is ilustrated in Figure 5 for a singularity at node 3 in a flat triangle.

In the cases where the singular point is on the edge of a quadratic element rather than on a vertex the triangle is divided in two and the same subdivision process applied to the resulting subtriangles.

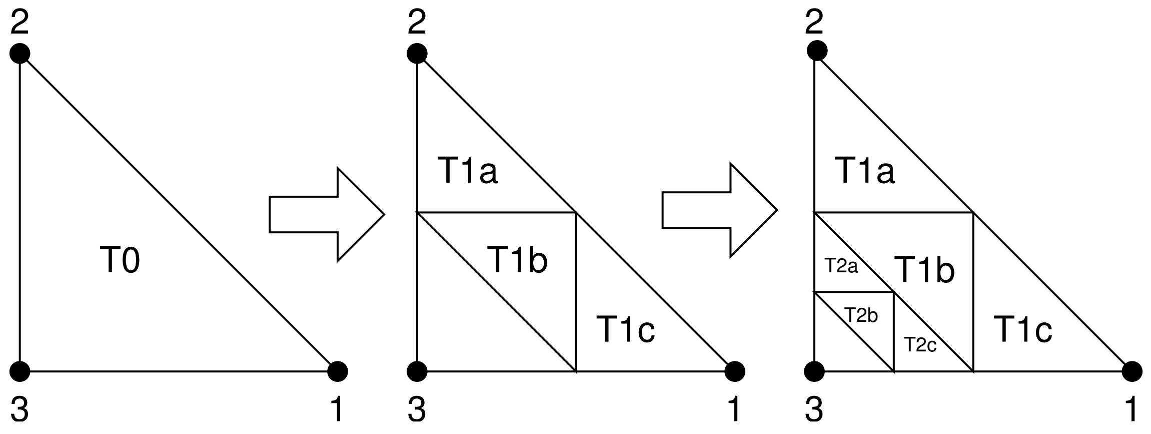

Figure 6: Subdivision process for strongly singular integrands with singularity at node 3 (only two consecutive subdivisions shown).

## Changing the number of integration points

The gaussian quadrature data is stored in the file `gaussData.h`, and has two options, one with 7 integration points and another one with 64 integrations points. By default the code uses 7 integration points because increasing this number does not seem to improve accuracy, but this can be changed by following these stpdf:

1. Change the value of `TNGAUSS` and `TSNGAUSS` to 64 in file `constants.h`
2. Comment out the 7 point `TGauss` and `TSGauss` matrices in file `gaussData.h`
3. Uncomment the 64 point `TGauss` and `TSGauss` matrices in file `gaussData.h`
4. Recompile the source code

The constant values `TNGAUSS` and `TSNGAUSS` are the number of integration points in regular and strongly singular integrals, so it is also possible to use different values for them. Keep in mind that `NSUBDIVISIONS` are made in the case of strongly singular integrals, and therefore many more points are used for the integration even if `TNGAUSS` and `TSNGAUSS` are given the same value.

The numerical values of the gaussian integration are shown in the following tables.

Table 1: Abscissas and weights for Gauss quadrature on a triangle

| NG | $x_i$ | $y_i$ | $w_i$ |
|---|---|---|---|
| 7 | 0.333333333333333 | 0.333333333333333 | 0.112500000000000 |
| | 0.470142064105115 | 0.470142064105115 | 0.066197076394253 |
| | 0.059715871789770 | 0.470142064105115 | 0.066197076394253 |
| | 0.470142064105115 | 0.059715871789770 | 0.066197076394253 |
| | 0.101286507323456 | 0.101286507323456 | 0.062969590272414 |
| | 0.797426985353088 | 0.101286507323456 | 0.062969590272414 |
| | 0.101286507323456 | 0.797426985353088 | 0.062969590272414 |

Table 2: Abscissas and weights for Gauss-Jacobi quadrature on a line

| NG | $x_i$ | $w_i$ |
|---|---|---|
| 8 | -0.910732089420060 | 0.013180765768995 |
| | -0.711267485915709 | 0.713716106239446 |
| | -0.426350485711139 | 0.181757278018796 |
| | -0.090373369606853 | 0.316798397969277 |
| | 0.256135670833455 | 0.424189437743720 |
| | 0.571383041208738 | 0.450023197883551 |
| | 0.817352784200412 | 0.364476094545495 |
| | 0.964440169705273 | 0.178203217446225 |

Table 3: Abscissas and weights for Gauss quadrature on a line

| NG | $x_i$ | $w_i$ |
|---|---|---|
| 8 | -0.960289856497536 | 0.101228536290370 |
| | -0.796666477413627 | 0.222381034453376 |
| | -0.525532409916329 | 0.313706645877887 |
| | -0.183434642495650 | 0.362683783378362 |
| | 0.183434642495650 | 0.362683783378362 |
| | 0.525532409916329 | 0.313706645877887 |
| | 0.796666477413627 | 0.222381034453376 |
| | 0.960289856497536 | 0.101228536290370 |