thSolver v1.0

Document Revision 1.0
September 1, 2006

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

Heterogeneous Coupled Systems Team
Large-Scale Complex Systems
Institute of High Performance Computing
1 Science Park Road, #01-01 The Capricorn
Science Park II, Singapore 117528

Preface

`thSolver` is a Boundary Element Method solver for the 3D Poisson's equation in microdevice thermal problems. I originally wrote this program to study the thermal characteristics of different dielectrophoretic trap designs, and I used it together with other program called `depSolver` [1]. I calculated the solution to the electrostatic problem using `depSolver` and then used its output as input for `thSolver`. This is still the most straightforward way to use `thSolver`, and the one I recommend. An example of research work done with this code is reference [2].

Using `thSolver` properly requires a good understanding behind the approximatis that lead to a simple Poisson equation for the temperature in microdevice thermal problems. Please do not run this program blindly. `thSolver` is a numerical tool, not a universal solution. Keep this in mind and read the documentation carefully before deciding if this is the right tool for your problem.

This manual consists of three parts: Part I describes how to prepare the inputs necessary for the program and how to post-process the solution; Part II describes the indirect formulation of the BEM for those interested in the details, and Part III gives details on the implementation. In order to use the code you striclty only need to read Part I, but I recommend anyone using the program to calculate thermal properties in microdevices to go carefully through Chapter 9 in Part III of this document. Part II has been included because most people are not familiar with the indirect formulation of the BEM, but you can safely ignore it if this does not apply to you.

The Institute of High Performance Computing (IHPC) distributes `thSolver` under a dual licensing policy. We believe in open source software for pushing the frontiers of science and engineering, and we encourage everyone to publish open source software under the GPL License. If you are developing and distributing open source applications under the GPL License, then you are free to use any element of `thSolver` under the GPL License. For OEMs, ISVs, and VARs who distribute any element of `thSolver` with their products, and do not license and distribute their source code under the GPL, IHPC provides a flexible OEM Commercial License.

Finally, although the GPL License does not require you to contact us, if you find this program useful, I would appreciate if you could send an email telling me that you are using `thSolver` and what application you are using the code for.

The complete package can be downloaded from `software.ihpc.a-star.edu.sg`.

Carlos Rosales Fernández
Singapore, September 1, 2006

Contents

I User Guide **1**

1 Introduction and general remarks **3**

1.1 Quick Install Guide 3

1.1.1 Compilation 4

1.1.2 Running the solver 4

2 Pre-Processing **7**

2.1 Generating the surface mesh with Patran 7

2.2 Converting Patran output to the right format 8

2.3 Generating mesh for domain integral evaluation 9

2.4 Generating evaluation points file 9

3 Input File Format **11**

3.1 Nodes Section 11

3.2 Elements Section 12

3.3 Materials Section 12

3.4 Interfaces Section . . . 13
3.5 Problem Section . . . 13
3.5.1 DIRECT domain integral solver . . . 14
3.5.2 CUBATURE domain integral solver . . . 15
3.5.3 ADAPTIVE domain integral solver . . . 15
3.6 Analysis Section . . . 15
3.7 Internal Points Section . . . 16
3.8 Columns Section . . . 17

4 Post-Processing **19**
4.1 break . . . 19
4.2 gplotForm . . . 20
4.3 thPost . . . 21

5 Example Files **23**

II The Indirect Boundary Element Formulation **25**

6 Indirect Formulation of the Boundary Element Method **27**
6.1 Laplace Equation in the IBEM Formulation . . . 27
6.2 Poisson Equation in the IBEM Formulation . . . 31

III Implementation Details **33**

7 Element Library **35**

8 Numerical Integration **37**

8.1 Regular integrals 37

8.2 Weakly singular integrals 37

8.3 Strongly singular integrals 39

8.3.1 The domain integal 39

8.4 Changing the number of integration points 44

9 The thermal problem in DEP traps **47**

9.1 IBEM treatment of the thermal problem in DEP trap design 50

A Function Listing **53**

References **57**

Part I

User Guide

Chapter 1

Introduction and general remarks

`thSolver` is distributed as a series of C function files, a make file for compilation, a Patran [3] 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 [4] and gnuplot [5]. The complete list of functions is collected in appendix A.

1.1 Quick Install Guide

First of all, unzip the `thSolver.tar.gz` file in a suitable directory. This can be done in a linux system by typing on the command line:

```
$ tar -zxvf thSolver.tar.gz
```

or alternatively using two separate commands:

```
$ gunzip depSolver-1.0.tar.gz
$ tar -xvf depSolver-1.0.tar
```

The following directories will be created:

`thSolver`	: Main program directory
`thSolver/bin`	: Directory where the binary files are stored after compilation
`thSolver/docs`	: Directory containing the code documentation
`thSolver/examples`	: Directory with examples to test the compiled code
`thSolver/src`	: 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/gplotFormat`	: 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 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/src` directory. Then proceed with the following steps.

1.1.1 Compilation

A makefile is provided for the compilation of the code. Inside `thSolver/src` type:

```
$ make thSolver
```

This will compile using gcc with the flag -O3 and several other performance optimization flags. If for some reason you don't like this, or you would like to add an architecture-related flag simply change the makefile. The executable files are automatically saved to `thSolver/bin`.

1.1.2 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.

Chapter 2

Pre-Processing

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

2.1 Generating the surface mesh with 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 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/p2b` 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:

```
$ ./meshgen 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:

```
$ ./meshgen 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`.

Chapter 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` $\times$ `ny` $\times$ `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` $\times$ `nyi` $\times$ `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` $\times$ `NG` $\times$ `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` $\times$ `ny` $\times$ `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 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` 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 `preCond` and `nInit` parameters
`gmres`	: GMRES solver, requires the parameters `preCond` and `nInit`

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` takes value 0 if no pre-conditioning is required and value 1 for Jacobi pre-conditioning (recommended). `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.

Chapter 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 gplotForm

In the directory `thSolver/utils/gplotFormat` there is an executable called `gplotForm` 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:

```
$ ./gplotForm 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:

```
$ ./gplotForm 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 `./gplotForm -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.

Chapter 5

Example Files

There is one example set of input and output files in `/thSolver/examples`. These files are provided as a way to become familiar with the input file format as well as a handy test of your executable. After compiling you may want to run one of the examples and compare the outputs you obtain with the outputs provided. Although small differences in the last digits may be due to different compilers / architectures, a significant difference will indicate a problem. Please contact the developer with details of the problem if this happens.

The examples provided are:

dep-trap-adaptive: Example of temperature calculation in a dielectrophoretic trap with two materials: a buffer liquid and an electrically insulating substrate. The domain integral is calculated using the adaptive algorithm.

dep-trap-direct: Same example as above but using a direct evaluation of the domain integral.

Part II

The Indirect Boundary Element Formulation

Chapter 6

Indirect Formulation of the Boundary Element Method

In this section the indirect formulation of the boundary element method is derived from the well-known direct formulation. Both Laplace and Poisson equations are treated in detail.

6.1 Laplace Equation in the IBEM Formulation

There is plenty of literature on the direct boundary element method formulation (DBEM), and the interested reader can find good reviews in the classical texts by Brebbia [6] and Wrobel [7]. In this section the equations for the indirect boundary element method (IBEM) will be derived from the well stablished equations for the DBEM.

Let us study a potential ϕ that obeys Laplace equation $\nabla^2\phi = 0$ in all space. Consider a bounded domain Ω enclosed by a smooth boundary Γ, and surrounded by its complementary domain $\overline{\Omega} = \mathbb{R}^3 - \Omega$ as shown in Figure 6.1.

The integral equation describing the potential at any point in the domain Ω is given by:

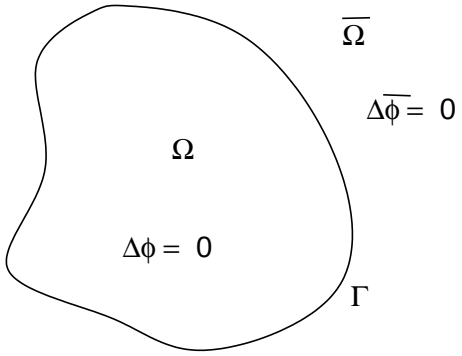

Figure 6.1: Domain used for the definition of the equations in the indirect method.

$$c(\vec{r})\phi(\vec{r}) + \int_\Gamma \phi(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma q(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' \tag{6.1}$$

where $G(\vec{r},\vec{r}')$ is the Green's function of the problem to solve, and $H(\vec{r},\vec{r}')$ its normal derivative in the direction pointing outwards of the domain. $q(\vec{r}')$ is defined as the normal derivative of the potential $\partial_{\mathrm{n}}\phi$ with the normal pointing outwards of the domain. $c(\vec{r})$ is given – for smooth surfaces – by:

$$c(\vec{r}) = \begin{cases} 1 & \text{if } \vec{r} \in \Omega \\ 1/2 & \text{if } \vec{r} \in \Gamma \\ 0 & \text{if } \vec{r} \in \overline{\Omega} \end{cases} \tag{6.2}$$

The Green's function for free space is defined as the solution to the problem:

$$\nabla^2 G(\vec{d}) = -\delta(\vec{d}) \tag{6.3}$$

which has the following forms in two and three dimensions:

$$G^{2D}(\vec{d}) = \frac{1}{2\pi}\ln\left(\frac{1}{d}\right) \tag{6.4}$$

$$G^{3D}(\vec{d}) = \frac{1}{4\pi d} \tag{6.5}$$

where we have defined the vector $\vec{d}$ as the relative distance between two vectors $\vec{r}$ and $\vec{r}'$:

$$\vec{d} = \vec{r} - \vec{r}' \tag{6.6}$$
$$d = |\vec{r} - \vec{r}'| \tag{6.7}$$

The normal derivative of the Green's function is obtained from equations (6.4) and (6.5) as:

$$H^{2D}(\vec{d}) = \frac{1}{2\pi}\frac{\vec{d}\cdot\hat{n}}{d^2} \tag{6.8}$$
$$H^{3D}(\vec{d}) = \frac{1}{4\pi}\frac{\vec{d}\cdot\hat{n}}{d^3} \tag{6.9}$$

Let $\overline{\phi}$ denote the solution to Laplace equation in the complementary domain $\overline{\Omega}$, and $\overline{q}$ be its normal derivative. The potential in the complementary domain must obey the following expression:

$$\overline{c}(\vec{r})\overline{\phi}(\vec{r}) + \int_\Gamma \overline{\phi}(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma \overline{q}(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' \tag{6.10}$$

For any point with position $\vec{r}$ interior to Ω we have $c(\vec{r}) = 1$ and $\overline{c}(\vec{r}) = 0$, so we can re-write equations (6.1) and (6.10) as:

$$\phi(\vec{r}) + \int_\Gamma \phi(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma q(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' \tag{6.11}$$
$$\int_\Gamma \overline{\phi}(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma \overline{q}(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' \tag{6.12}$$

Taking now the sum of these two equations, and using both in q and $\overline{q}$ the normal exterior to Ω, we obtain the following expression:

$$\phi(\vec{r}) + \int_\Gamma \left[\phi(\vec{r}') - \overline{\phi}(\vec{r}')\right] H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma \left[q(\vec{r}') + \overline{q}(\vec{r}')\right] G(\vec{r},\vec{r}')d\Gamma' \tag{6.13}$$

This is the general expression for the potential in the indirect boundary element method, where the integral on the right hand side can be considered as the contribution of a distribution of sources, and the integral on the left hand side as a the contribution from a distribution of dipoles. Let us define the following source and dipole densities:

$$s(\vec{r}') = \overline{q}(\vec{r}') + q(\vec{r}') \tag{6.14}$$
$$m(\vec{r}') = \overline{\phi}(\vec{r}') - \phi(\vec{r}') \tag{6.15}$$

Substituting now these definitions into the previous expression we obtain a more familiar result:

$$\phi(\vec{r}) = \int_\Gamma m(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' + \int_\Gamma s(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' \tag{6.16}$$

Comparing expressions (6.1) and (6.16) we see that they are formally identical, but that in the case of the IBEM we need to solve for twice as many unknowns. In principle, this would leave us with an underspecified system, but we can get around this problem by espcifying an extra boundary condition on each node.

This additional boundary condition will depend on what physical quantity we are trying to calculate. When the condition specified is the continuity of the potential we find that $m(\vec{r}') = 0$ and the problem is reduced to a *source formulation*:

$$\phi(\vec{r}) = \int_\Gamma s(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' \tag{6.17}$$

When the condition specified is the continuity of the normal derivative of the potential we find that $s(\vec{r}') = 0$ and the problem is reduced to a *dipole formulation*:

$$\phi(\vec{r}) = \int_\Gamma m(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' \tag{6.18}$$

6.2 Poisson Equation in the IBEM Formulation

Let us consider now the case where there is a certain distribution of sources in space, such that the potential obeys the Poisson equation, $\nabla^2\phi = b(\vec{r})$, as shown in Figure 6.2.

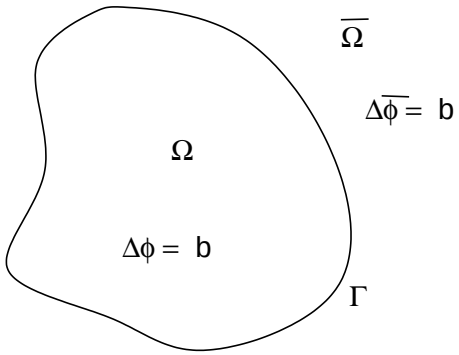

Figure 6.2: The Poisson problem domain for the definition of the IBEM equations.

In this case the equation that must be obeyed by the potential at any point in the domain Ω is given by:

$$c(\vec{r})\phi(\vec{r}) + \int_\Gamma \phi(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma q(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' - \int_\Omega b(\vec{r}')G(\vec{r},\vec{r}')d\Omega' \quad (6.19)$$

For the complementary domain we have a similar expression:

$$\overline{c}(\vec{r})\overline{\phi}(\vec{r}) + \int_\Gamma \overline{\phi}(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma \overline{q}(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' - \int_{\overline{\Omega}} b(\vec{r}')G(\vec{r},\vec{r}')d\Omega' \quad (6.20)$$

Just like inthe previous section, for any point at position $\vec{r}$ interior to Ω we have $c(\vec{r}) = 1$ and $\overline{c}(\vec{r}) = 0$, so we can re-write equations (6.19) and (6.20) as:

$$\phi(\vec{r}) \quad + \quad \int_\Gamma \phi(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma q(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' - \int_\Omega b(\vec{r}')G(\vec{r},\vec{r}')d\Omega' \quad (6.21)$$

$$\int_\Gamma \overline{\phi}(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma \overline{q}(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' - \int_{\overline{\Omega}} b(\vec{r}')G(\vec{r},\vec{r}')d\Omega' \quad (6.22)$$

Preceeding as in section 6.1 we add these two equations, and using both in q and $\overline{q}$ the normal exterior to Ω, we obtain the following expression:

$$\phi(\vec{r}) + \int_\Gamma \left[\phi(\vec{r}') - \overline{\phi}(\vec{r}')\right] H(\vec{r},\vec{r}')d\Gamma' = \int_\Gamma \left[q(\vec{r}') + \overline{q}(\vec{r}')\right] G(\vec{r},\vec{r}')d\Gamma' - \int_{\Omega+\overline{\Omega}} b(\vec{r}')G(\vec{r},\vec{r}')d\Omega' \quad (6.23)$$

Using the definitions (6.14) and (6.15) we can re-write this expression with a source term and a dipole term:

$$\phi(\vec{r}) = \int_\Gamma m(\vec{r}')H(\vec{r},\vec{r}')d\Gamma' + \int_\Gamma s(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' - \int_{\Omega+\overline{\Omega}} b(\vec{r}')G(\vec{r},\vec{r}')d\Omega' \quad (6.24)$$

As expected, this expression is very similar to the one obtained in the case of Laplace equation, with the exception of the domain integral due to the body force term.

Part III

Implementation Details

Chapter 7

Element Library

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

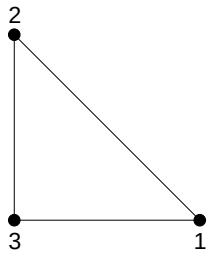

Figure 7.1: Linear interpolation.

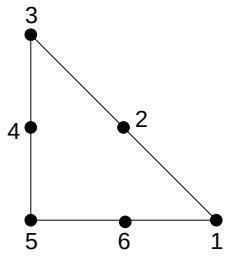

Figure 7.2: Quadratic interpolation.

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

$$N_1 = L_1 \tag{7.1}$$
$$N_2 = L_2 \tag{7.2}$$
$$N_3 = 1 - L_1 - L_2 \tag{7.3}$$

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

$$N_1 = L_1(2L_1 - 1) \tag{7.4}$$

$$
\begin{aligned}
N_2 &= 4L_1L_2 && (7.5)\\
N_3 &= L_2(2L_2-1) && (7.6)\\
N_4 &= 4L2(1-L_1-L_2) && (7.7)\\
N_5 &= L_3[1-2(L_1+L_2)] && (7.8)\\
N_6 &= 4L_1(1-L_1-L_2) && (7.9)
\end{aligned}
$$

Chapter 8

Numerical Integration

8.1 Regular integrals

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

$$\int_{-1}^{1} F(L_1, L_2) dL_1 dL_2 \approx \sum_{i=1}^{NG} F(L_1^i, L_2^i) w_i \tag{8.1}$$

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.

8.2 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 8.1 for the transformation.

$$\int_{-1}^{1} F(L_1, L_2)(1 + L_2) dL_1 dL_2 \approx \sum_{i=1}^{NG} \sum_{j=1}^{NG} F(L_1^i, L_2^j) w_i^{\text{Gauss}} w_j^{\text{Gauss-Jacobi}} \tag{8.2}$$

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.

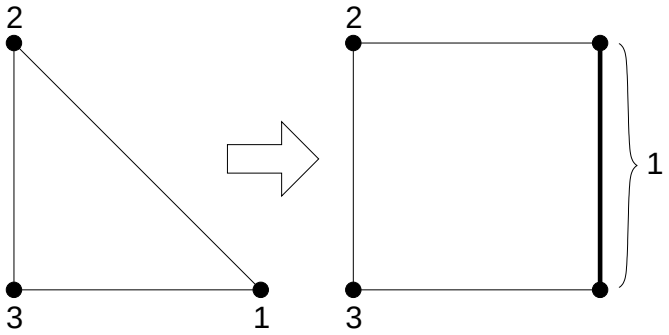

Figure 8.1: Regularization transformation for weakly singular integrals (linear case).

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 8.2.

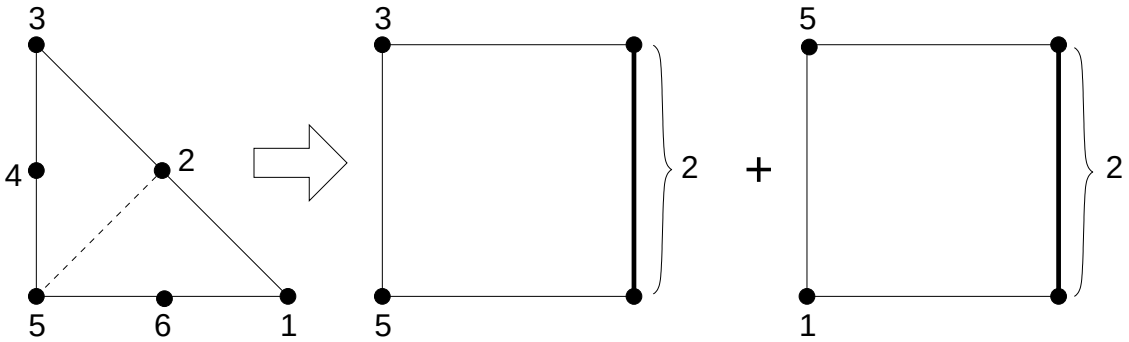

Figure 8.2: Regularization transformation for weakly singular integrals (quadratic case).

8.3 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 8.3 for a singularity at node 3 in a flat triangle.

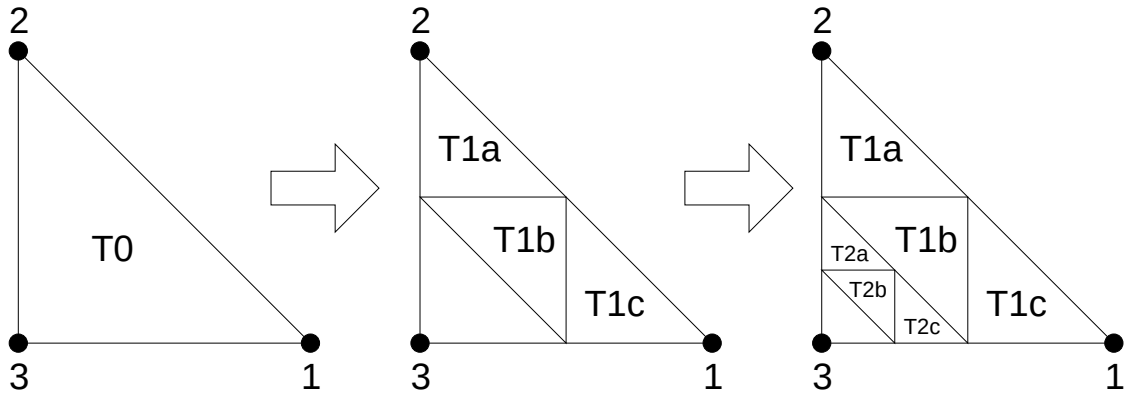

Figure 8.3: Subdivision process for strongly singular integrands with singularity at node 3 (only two consecutive subdivisions shown).

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.

8.3.1 The domain integal

There are several ways to deal with the domain integral that corresponds to the source term in Poisson's equation. `thSolver` supports direct evaluation, gaussian cubature, and adaptive direct evaluation as explained below.

Direct evaluation

Possibly the most straight forward way to solve this integral is to generate a certain number N_{in} of points homogeneously distributed inside the volume V of the trap and the integral is approximated by the following sum:

$$\int_{\Omega} b(\vec{r}')H(\vec{r},\vec{r}')d\Omega' \approx \frac{V}{N_{\text{in}}}\sum_{p=1}^{N_{\text{in}}} b(\vec{r}_p)G(\vec{r},\vec{r}_p) \tag{8.3}$$

As long as the number of points N_{in} is large enough to capture the spatial variation of $b(\vec{r})$ the results will be correct. This requires several steps:

1. Generate N_{in} positions in the volume of interest.
2. Calculate $E^2(\vec{r}_p)$ at all generated points and store in a vector.
3. For each node in a boundary between two materials calculate (8.3).
4. Add the result to the corresponding element of the right hand side vector in the system.

Once this has been done we can assemble the matrix for the system as usual and solve using Gauss elimination or GMRES iteration. This methos has teh disadvantage of requiring a very high number of evaluations of the source term when the source is highly non-homogeneous.

Gauss Cubature

By choosing cubature points that always lie within the unit cube we can integrate the source term numerically without encountering any singular integrand. `thSolver` using a product rule for the integration:

$$\int_{-1}^{1} F(L_1,L_2,L_3)dL_1dL_2dL_3 \approx \sum_{i=1}^{NG}\sum_{j=1}^{NG}\sum_{k=1}^{NG} F(x_i,y_j,z_k)w_iw_jw_k \tag{8.4}$$

The steps followed to perform the integral are very similar to those in the direct evaluation method:

1. Generate M_{in} cells in the volume of interest.
2. Use a linear transformation from each cube to the unit cube.
3. Calculate $E^2(\vec{r}_p)$ at all integration points and store in a vector.
4. For each node in a boundary between two materials calculate (8.3).
5. Add the result to the corresponding element of the right hand side vector in the system.

Notice that as the number of evaluation points in the domain increases as NG^3, so this method will become computationally expensive very quickly for problems with non-homogeneous source terms.

Assumes cells are given in the form shown in figure 8.4. The electric field is calculated at every corner of the cell and then interpolated using trilinear functions at the cubature points.

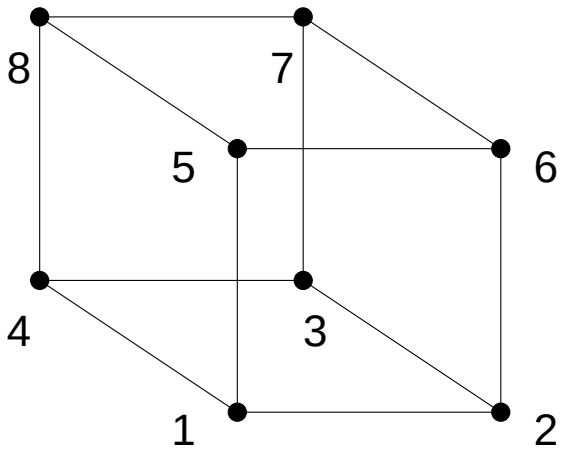

Figure 8.4: Basic cell used for gaussian cubature. The node ordering is matched to the output from the utility cellgen.

Adaptive Direct Evaluation

The method we use in this case is based on an adaptive evaluation of the volume integral. Finding an optimal mesh for each collection point in the surface, as strictly required by the source term integral, is not practical as it would result in an evaluation that is even more computationally expensive than a direct evaluation in a homogeneous mesh. But for cases where the source term changes more rapidly than $1/|\vec{r}-\vec{r}'|$ we will assume that the optimal mesh for the source term integral provides a nearly optimal mesh for the actual system of equations.

There are two reasons for the high efficiency of the adaptive method we present:

- It does not require derivative evaluations
- It does not require checks for duplicated nodes

Because we are evaluating an integral rather than a function directly, in order to obtain a highly accurate evaluation we would like to make the contribution to the integral from all the cells in which the volume is divided to be the same. Assuming that the variation of the integrand is large but smooth over the region of interest – that is, it does not fluctuate with a high frequency with respect to the domain size –, we can use as subdivision criteria for each cell the comparison of its local contribution with the average contribution from all cells in the domain. When the individual contribution from a cell to the integral is large compared to the average, the cell must be subdivided in order to refine the calculation in that region. If the contribution of the cell is small in comparison to the average then the cell is left unchanged. Once the subdivision check has been done in all cells a new evaluation of the integral is done using the refined mesh and it is compared with the previous one. If the convergence criteria set by the user is met the subdivision ends, otherwise a new subdivision check is done again, and so on until the convergence criteria is met. This process is shown in Figure 8.5.

It is important to notice that in cases where the source term oscillates rapidly within the domain this approximated method will not provide accurate results unless it is complemented by a consistency check. One can easily imagine a case where, due to the lack of a criteria involving local derivatives, the contribution from a cell where the integrand is highly oscillating is underestimated if the evaluation point falls on a local minimum of the oscillation. A simple test would consist on a second evaluation

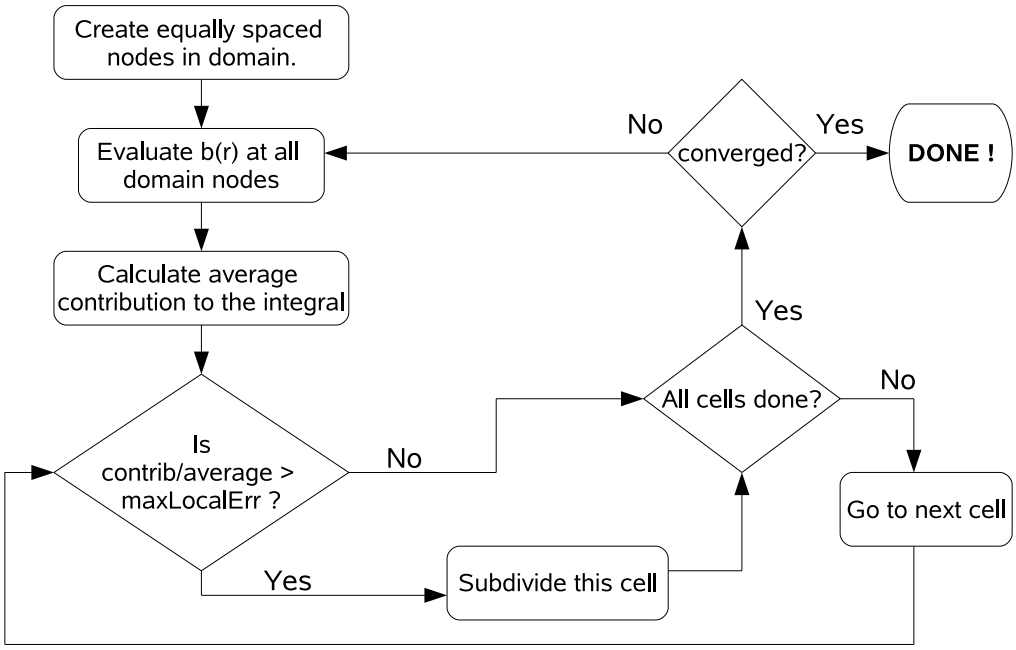

Figure 8.5: Adaptive meshing algorithm description. b(r) stands for the source term.

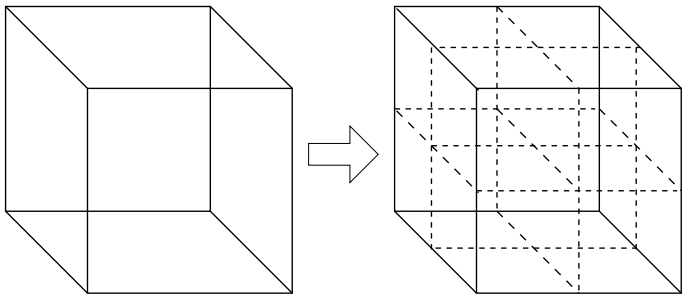

Figure 8.6: Cell division

of the source term with a slightly displaced mesh to verify that the original mesh is adequate and it is not missing regions of particularly fast spatial change. For many physical problems, as the one shown in this paper, this is not relevant, and the adaptive evaluation is both accurate and efficient.

The choice of insertion points at each subdivision level is such that there is no need to keep track of the nearest neighbors of each point or to double check in order to avoid repeated nodes. This advantage is gained because the subdivision is done in terms of evaluation cells. At each subdivision level if the contribution of a cell to the total value of the integral exceeds the average contribution from all cells by a relative amount set by the user, the cell is divided in eight identical cells. For subdivision purposes the original cell ceases to exist – it is flagged as an inactive cell in the code – and it is substituted by its eight children as shown in Figure 8.6. When the integral is calculated in the refined mesh at the next subdivision level the integrand is evaluated at the centers of each of the new eight cells besides the original point marking the center of the parent cell. In this way there is never the risk of introducing a new evaluation point on a position currently occupied by another, and the implementation of the algorithm becomes very efficient.

8.4 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.

All integration schemes were chosen so that the integration points are in the interior of the elements, avoiding issues with evaluation points falling exactly on top of mesh nodes. The numerical values of the gaussian integration are reproduced 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

Chapter 9

The thermal problem in DEP traps

It is of interest to know what is the steady state temperature in a microdevice, because biological material is very sensitive to high temperatures, and the currents in these devices can be very large, producing signifficant a signifficant amount of heat in a very small volume. In order to know whether a design is viable it is necessary to know the temperature that the liquid will reach when the device is in operation. Some work on the subject has been done for 2D designs by Green *et. al.* [8]. The basic problem is described in Figure 9.1.

The energy balance equation can be written as:

$$\underbrace{\rho_{\mathrm{m}} c_{\mathrm{p}} \frac{\partial T}{\partial t}}_{I} + \underbrace{\rho_{\mathrm{m}} c_{\mathrm{p}} \vec{u} \cdot \nabla T}_{II} = \underbrace{k \nabla^2 T}_{III} + \underbrace{\sigma E^2}_{IV} \tag{9.1}$$

where we have assumed a constant thermal conductivity, k, and a constant electrical conductivity, σ. In this expression c_{p} is the specific heat at constant pressure, and ρ_{m} is the mass density. The viscous dissipation term has been neglected because it is of order 10^{-10} times the joule heating (term IV). In order to simplify this equation we will consider the orders of magnitude of the different terms in the equation following the reasoning of Ramos *et. al.* [9]

First of all, let us justify the steady state approximation. The typical diffusion time

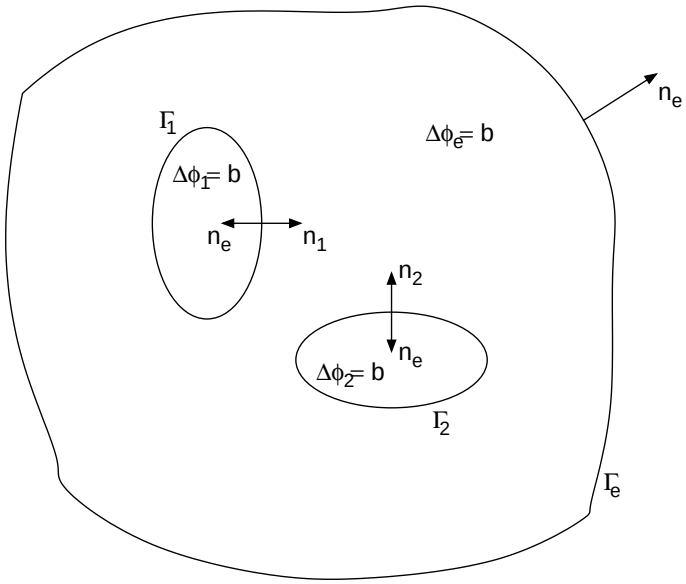

Figure 9.1: Poisson problem with several materials. Note the definition of the normal vectors.

for the temperature front can be obtained from the thermal conductivity or Fourier equation:

$$\frac{\partial T}{\partial t} = \frac{k}{\rho_{\mathrm{m}} c_{\mathrm{p}}} \nabla^2 T \tag{9.2}$$

An order of magnitude analysis of this expression gives us:

$$t_{\mathrm{diff}} \sim \frac{\rho_{\mathrm{m}} c_{\mathrm{p}} L^2}{k} \tag{9.3}$$

where L is the characteristic length of the system. For water we can use $k = 0.6$ $Jm^{-1}s^{-1}K^{-1}$, $c_{\mathrm{p}} = 4.18 \cdot 10^3$ $JKg^{-1}K^{-1}$, and $\rho_{\mathrm{m}} = 10^3$ Kgm^{-3}. For a trap height of 50 μm we find:

$$t_{\mathrm{diff}} \sim \frac{10^3 \cdot 4.18 \cdot 10^3 \cdot (5 \cdot 10^{-5})^2}{0.6} = 1.7 \cdot 10^{-2} \quad \mathrm{s} \tag{9.4}$$

Which implies that a thermal equilibrium is reached within 20 ms of the trap being switched on. In an AC field the temperature of the system can be expressed as a time-averaged temperature T_0 plus a time-dependent part $\Delta T(t)$ that depends on 2ω. Going back to equation (9.2) we find:

$$\frac{\Delta T}{\Delta t} \sim \frac{k}{\rho_{\mathrm{m}} c_{\mathrm{p}} L^2} T \tag{9.5}$$

So that we can write, using (9.3):

$$\frac{\Delta T}{T} \sim \frac{1}{2\omega t_{\mathrm{diff}}} \tag{9.6}$$

If we want the fluctuation in the temperature to be negligible we need to make sure that $\Delta T/T \ll 1$. Using the value of t_{diff} calculated before in (9.4) we find:

$$\frac{1}{f} \ll 4\pi t_{\mathrm{diff}} \qquad \Rightarrow \qquad f \gg \frac{1}{4\pi t_{\mathrm{diff}}} = 5 \quad \mathrm{Hz} \tag{9.7}$$

This indicates that as long as the frequency of the applied field is above 50 Hz we can consider the system in a steady state and neglect term I in (9.1).

We still have a significantly complicated equation, so we will considernow the effect of the convective flow in the energy equation. In order to neglect the effect of the field flow on the temperature profile we need:

$$|\rho_{\mathrm{m}} c_{\mathrm{p}} \vec{u} \cdot \nabla T| \ll |k \nabla^2 T| \tag{9.8}$$

Using dimensional analysis we find:

$$u \ll \frac{k}{\rho_{\mathrm{m}} c_{\mathrm{p}} L} \tag{9.9}$$

And using the same numerical values as before we obtain an upper limit for the flow speed:

$$u \ll \frac{0.6}{10^3 \cdot 4.18 \cdot 10^3 \cdot 5 \cdot 10^{-5}} = 2 \cdot 10^{-3} \quad \text{m/s} \tag{9.10}$$

Thus, as long as the flow speed remains well under 2 mm/s we ca safely ignore convection (term II) in equation (9.1).

This leaves us with a very simplified equation to solve, that takes the form of a Poisson's equation:

$$\nabla^2 T = \frac{\sigma E^2}{k} \tag{9.11}$$

9.1 IBEM treatment of the thermal problem in DEP trap design

As shown above the equation to solve in this case is the Poisson equation, with the source term given by the modulus square of the electric field, that must be calculated first.

The boundary conditions can be either a given temperature or a material interface. We will consider that the temperature is specified in the external boundaries of the system. In the surfaces separating different materials the boundary conditions conditions are continuity of the temperature and continuity of the normal flux:

$$T_{\mathrm{e}}|_{\Gamma_{\mathrm{i}}} = T_{\mathrm{i}}|_{\Gamma_{\mathrm{i}}} \tag{9.12}$$

$$k_{\mathrm{e}} \left.\frac{\partial T_{\mathrm{e}}}{\partial n}\right|_{\Gamma_{\mathrm{i}}} = k_{\mathrm{i}} \left.\frac{\partial T_{\mathrm{i}}}{\partial n}\right|_{\Gamma_{\mathrm{i}}} \tag{9.13}$$

Due to the continuity of the temperature in the whole domain we must set the dipole density to zero in all surfaces, and we are left with a simplified version of equation (6.24) with only sources:

$$T(\vec{r}) = \int_{\Gamma} t(\vec{r}')G(\vec{r},\vec{r}')d\Gamma' - \int_{\Omega} b(\vec{r}')G(\vec{r},\vec{r}')d\Omega' \tag{9.14}$$

where $b(\vec{r}')$ is given in this case by:

$$b(\vec{r}') = -\frac{\sigma}{k} E^2(\vec{r}') \tag{9.15}$$

If we redefine the Green's function and its derivative as in (6.5) and (6.9) we can rewrite this as:

$$T(\vec{r}) = \frac{1}{2\alpha\pi k_0} \left[\int_\Gamma t(\vec{r}') G(\vec{r}, \vec{r}') d\Gamma' - \int_\Omega b(\vec{r}') G(\vec{r}, \vec{r}') d\Omega' \right] \tag{9.16}$$

Taking the normal derivative of this equation at a point not in a boundary gives:

$$\frac{\partial T(\vec{r})}{\partial n} = \frac{1}{2\alpha\pi k_0} \left[\int_\Gamma t(\vec{r}') H(\vec{r}, \vec{r}') d\Gamma' - \int_\Omega b(\vec{r}') H(\vec{r}, \vec{r}') d\Omega' \right] \tag{9.17}$$

Taking this to the boundary and includig the jump in the normal derivative we obtain the following expressions:

$$\frac{\partial T_{\mathrm{e}}(\vec{r})}{\partial n} = -\frac{t(\vec{r})}{2k_0} + \frac{1}{2\alpha\pi k_0} \left[\int_\Gamma t(\vec{r}') H(\vec{r}, \vec{r}') d\Gamma' - \int_\Omega b(\vec{r}') H(\vec{r}, \vec{r}') d\Omega' \right] \qquad \text{if } \vec{r} \in \Gamma_{\mathrm{i}} \tag{9.18}$$

$$\frac{\partial T_{\mathrm{i}}(\vec{r})}{\partial n} = \frac{t(\vec{r})}{2k_0} + \frac{1}{2\alpha\pi k_0} \left[\int_\Gamma t(\vec{r}') H(\vec{r}, \vec{r}') d\Gamma' - \int_\Omega b(\vec{r}') H(\vec{r}, \vec{r}') d\Omega' \right] \qquad \text{if } \vec{r} \in \Gamma_{\mathrm{i}} \tag{9.19}$$

Substituting these expressions into the boundary condition (9.13) we find:

$$\begin{aligned} k_{\mathrm{e}} \left\{ -\frac{t(\vec{r})}{2k_0} + \frac{1}{2\alpha\pi k_0} \left[\int_\Gamma t(\vec{r}') H(\vec{r}, \vec{r}') d\Gamma' - \int_\Omega b(\vec{r}') H(\vec{r}, \vec{r}') d\Omega' \right] \right\} \; = \\ k_{\mathrm{i}} \left\{ \frac{t(\vec{r})}{2k_0} + \frac{1}{2\alpha\pi k_0} \left[\int_\Gamma t(\vec{r}') H(\vec{r}, \vec{r}') d\Gamma' - \int_\Omega b(\vec{r}') H(\vec{r}, \vec{r}') d\Omega' \right] \right\} \end{aligned} \tag{9.20}$$

Rearranging terms we find:

$$-\frac{(k_{\mathrm{e}}+k_{\mathrm{i}})t(\vec{r})}{2k_0}=\frac{(k_{\mathrm{i}}-k_{\mathrm{e}})}{2\alpha\pi k_0}\left[\int_\Gamma t(\vec{r}')H(\vec{r},\vec{r}')d\Gamma'-\int_\Omega b(\vec{r}')H(\vec{r},\vec{r}')d\Omega'\right] \tag{9.21}$$

And isolating $t(\vec{r})$:

$$t(\vec{r})=\frac{(k_{\mathrm{e}}-k_{\mathrm{i}})}{\alpha\pi(k_{\mathrm{e}}+k_{\mathrm{i}})}\left[\int_\Gamma t(\vec{r}')H(\vec{r},\vec{r}')d\Gamma'-\int_\Omega b(\vec{r}')H(\vec{r},\vec{r}')d\Omega'\right] \tag{9.22}$$

Appendix A

Function Listing

Directory	: thSolver/src
Source Files	: 101
Compilation Script	: make thSolver

bodyForceAdapt_tria3.c
bodyForceAdapt_tria3.h
bodyForceAdapt_tria6.c
bodyForceAdapt_tria6.h
bodyForceCubat_tria3.c
bodyForceCubat_tria3.h
bodyForceCubat_tria6.c
bodyForceCubat_tria6.h
bodyForce_tria3.c
bodyForce_tria3.h
bodyForce_tria6.c
bodyForce_tria6.h
comFilter.c
constants.h
depThermal.c
depThermal.h
dotProd.c
doubleMatrix.c
doublePointer.c
doubleTensor.c
getNormal_tria6.c
initRes.c
initRes.h
integral_tria3.h
integral_tria6.h
interp3d.c
intG_tria3.c
intG_tria6.c
intH_tria3.c
intH_tria6.c
intSingularG_tria3.c
intSingularG_tria6.c
intSingularH_tria3.c
intSingularH_tria6.c
iterGMRES.c
iterGMRES.h
L2Norm.c
makefile
matVectProd.c
shape_tria3.c
temperature_tria6.h
thermalFormA_tria3.c
thermalFormA_tria3.h
thermalFormA_tria6.c
thermalFormA_tria6.h
thermalPostProcessAdapt_tria3.c
thermalPostProcessAdapt_tria3.h
thermalPostProcessAdapt_tria6.c
thermalPostProcessAdapt_tria6.h
thermalPostProcessColAdapt_tria3.c
thermalPostProcessColAdapt_tria3.h
thermalPostProcessColAdapt_tria6.c
thermalPostProcessColAdapt_tria6.h
thermalPostProcessColCubat_tria3.c
thermalPostProcessColCubat_tria3.h
thermalPostProcessColCubat_tria6.c
thermalPostProcessColCubat_tria6.h
thermalPostProcessCol_tria3.c
thermalPostProcessCol_tria3.h
thermalPostProcessCol_tria6.c

```
doubleVector.c          shape_tria6.c             thermalPostProcessCol_tria6.h
elemType.c              solverGMRES.c             thermalPostProcessCubat_tria3.c
errorHandler.c          solverGMRES.h             thermalPostProcessCubat_tria3.h
field_tria3.c           temperatureAdapt_tria3.c  thermalPostProcessCubat_tria6.c
field_tria3.h           temperatureAdapt_tria3.h  thermalPostProcessCubat_tria6.h
field_tria6.c           temperatureAdapt_tria6.c  thermalPostProcess_tria3.c
field_tria6.h           temperatureAdapt_tria6.h  thermalPostProcess_tria3.h
freeDoubleMatrix.c      temperatureCubat_tria3.c  thermalPostProcess_tria6.c
freeDoublePointer.c     temperatureCubat_tria3.h  thermalPostProcess_tria6.h
freeDoubleTensor.c      temperatureCubat_tria6.c  uintMatrix.c
freeUintMatrix.c        temperatureCubat_tria6.h  uintVector.c
gaussBksb.c             temperature_tria3.c       X2L_tria3.c
gaussData.h             temperature_tria3.h       X2L_tria6.c
getNormal_tria3.c       temperature_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/gplotForm
Source Files	: 2
Compilation Script	: gplotComp

```
gplotForm.c             errorHandler.c
```

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

meshgen.c errorHandler.c

Directory	: thSolver/utils/p2b
Source Files	: 5
Compilation Script	: p2bComp

bem_save.pcl doubleMatrix.c errorHandler.c
freeDoubleMatrix.c p2b.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

Bibliography

[1] depsolver is an open source boundary element method solver for electrostatic problems freely distributed by the Institute of High Performance Computing in Singapore. for more information visit their web site `http://software.ihpc.a-star.edu.sg`.

[2] C. Rosales. Electrode surface ratio optimization for thermal performance in 3d dielectrophoretic single-cell traps. *Electrophoresis*, 27:1984–1995, 2006.

[3] Patran is an open-architecture, 3-d computer aided-engineering software package from msc.software corporation. for more information visit their web site at `http://www.mscsoftware.com/products/patran.cfm`.

[4] Matlab (*matrix laboratory*) is an interactive programming environment from The MathWorks that provides tools for data visualization, data analysis, and numeric computation. for more information visit their web site at `http://www.mathworks.com/products/matlab/description1.html`.

[5] Gnuplot is a command-line driven interactive data and function plotting utility distributed as copyrighted but free software. for more information visit the official gnuplot web site `http://www.gnuplot.info/`.

[6] C. A. Brebbia. *The boundary element method for engineers*. Pentech Press, London (UK), 1978.

[7] L. C. Wrobel. *The Boundary Element Method*. John Wiley & Sons, Chichester (UK), 2002.

[8] N. G. Green, A. Ramos, A. Gonz'alez, A. Castellanos, and H. Morgan. Electrothermally induced fluid flow on microelectrodes. *Journal of Electrostatics*, 53:71–87, 2001.

[9] A. Ramos, H. Morgan, N. G. Green, and A. Castellanos. AC electrokinetics: a review of forces in microelectrode structures. *Journal of Physics D: Applied Physics*, 31:2338–2353, 1998.