# `depSolver` User Guide

v1.1, August 1, 2006

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

# Preface

`depSolver` is a boundary element method solver for Laplace's equation in AC electrostatic 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 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>`

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

Singapore, July 31, 2005

# Contents

# 1 Introduction and general remarks

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

## 1.1 Quick Install Guide

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

```
tar -zxvf depSolver.tgz
```

The following directories will be created:

| | |
|---|---|
| `depSolver` | : Main program directory |
| `depSolver/bin` | : Directory where the binary files are stored after compilation |
| `depSolver/docs` | : Directory containing this document in pdf form |
| `depSolver/examples` | : Directory with examples to test the compiled code |
| `depSolver/src` | : Directory containing the C source code |
| `depSolver/utils` | : Directory with utilities for pre- and post-processing |
| `utils/depBreak` | : Utility to break data files into several units |
| `utils/fieldPost` | : Utility to format the field values for plotting in matlab |
| `utils/forcePost` | : Utility to format the force values for plotting in matlab |
| `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/potPost` | : Utility to format the potential values for plotting in matlab |

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

### Compilation

Two versions of the code can be compiled. The basic version is called `depSolver`, and an additional version called `depSolver-sft` is also provided. Both versions of the code are identical except that in `depSolver-sft` it is assumed that a particle - which can be of any shape - is created in the geometry and the program will shift it as a rigid body following a vector given in the input file. Notice that this shift assumes that the particle is created at the center of coordinates. This version is useful to produme Maxwell Stress Tensor calculations of the dielectrophoretic force without having to create a new mesh for each particle position. It can be abused to work without a particle or using a zero displacement, but the basic `depSolver` is less confusing to use in those cases.

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

```
make depSolver
```

or

```
make depSolver-sft
```

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 `depSolver/bin`.

### Running the solver

In order to run `depSolver` certain input files are needed. These files have the extension `.bem` and the ones needed for every run 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

There are also two optional files files which are only used for some calculations:

`forcepoints.bem` : [opt] File containing the points where the force is calculated when using the multipolar approximation
`internal.bem` : [opt] File containing the internal points where the potential or field (or both) are required

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 `depSolver` 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 in 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 Re[s] Im[s]`

And those that are produced only for certain input options:

`gmres.log` : [opt] GMRES log file, registers the error per iteration
`field.dat` : [opt] Contains the electric field at the required points
`force-mp.dat` : [opt] Contains the DEP force (multipolar approximation)
`formce-mst.dat` : [opt] Contains the DEP force (Maxwell's stress tensor method)
`potential.dat` : [opt] Contains the potential at the required points

# 2 Pre-Processing

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

## 2.1 Mesh Generation Using 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 Re[V] 0 >
< 1 Im[V] 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 `depSolver` executable a further step is necessary. Go to the directory called `depSolver/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 `depSolver`, and setting the parameter `reOrder` to one transforms the element connectivity to the appropriate format for the program. It is also 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 evaluation points file

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

```
./meshgen a nx ny nz n rmin rmax 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 eps
```

Where `sigma` is the conductivity of the material, and `eps` its relative permittivity.

## 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 the frequency of the external field and the name of the file that has the boundary conditions. It looks like this:

```
PROBLEM
frequency
bcsFilename
```

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

```
nodeID bcType value1Re value2Re
...
nodeID bcType value1Im value2Im
...
```

A dielectric interface is indicated by a `bcType` of 0, a `value1` of 0, and a `value2` equal to the `interfaceID` where the node belongs. All three values must be the same for the real and imaginary parts.

If the dielectric interface belongs to a particle and the force is calculated using the Maxwell Stress tensor method the `bcType` must be set to 6 in order to differentiate this surface from any other dielectric interfaces in the system. `value1` and `value2` must be set to 0 and the `interfaceID` where the node belongs as for any other dielectric interface.

A dirichlet boundary condition - in this case, potential given on a conductor - is indicated by a `bcType` of 1, a `value1Re` equal to the real part of the potential, and a `value1Im` equal to the imaginary part of the potential. The `value2` is not used in the code and must be omited from the file.

Notice that in the boundary conditions file the real part of the boundary conditions is written first, so that for N nodes there are 2N rows, the first N with the real components, the last N with the imaginary ones.

The values 2 to 5 for the boundary condition type are not used by the program and may be assigned to other boundary condition types at a later stage.

## 3.6 Re-positioning Section

This section must only be used with `depSolver-sft`. It contains the first node corresponding to the particle and the displacement vector for the shift:

```
REPOSITION
pNode
dx dy dz
```

where `pNode` is the last node of the mesh before the particle nodes (if the nodes in the electrodes go from 1 to 536 and the nodes in the particle from 537 to 752 we would use `pNode` = 536. In MSC.Patran the particle must be meshed last to use this option correctly. (`dx`,`dy`,`dz`) are teh displacements in each of the spatial directions.

## 3.7 Analysis Section

This is the most complicated 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
pointNumber a b c
forceFilename
```

where `forceFilename` is the file containing the points where the force is calculated (when using the multipolar approximation), and has the following format:

```
X1 Y1 Z1
...
Xpn Ypn Zpn
```

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 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 Re[s] Im[s]
```

Where `Re[s]` and `Im[s]` are the real and imaginary parts of the solution at the point `(x,y,z)`, and the file has `nInit` rows.

A good practice is to solve the problem of an empty trap first specifying `nInit` as zero, and then use the solution as initial guess for a trap containing the particle. Note that due to the way the program works one should use exactly the same mesh (apart from the addition of the particle) in both cases. Normally one would generate the mesh for the empty trap, run it, then generate the mesh for the trap with te particle but taking care of meshing in the same order as before, leaving the mesh on the particle surface to the end. This guarantees the program will work correctly. If in doubt set `nInit` to zero.

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

0 : Calculate only the potential $\phi$
1 : Calculate only the electric field $\vec{E}$
2 : Calculate $\phi$ and $\vec{E}$
3 : Calculate $\phi$, $\vec{E}$, and $\vec{F}_{\mathrm{DEP}}$ using Maxwell's stress tensor method
4 : Calculate only $\vec{F}_{\mathrm{DEP}}$ using Maxwell's stress tensor method
5 : Calculate $\phi$, $\vec{E}$, and $\vec{F}_{\mathrm{DEP}}$ using a dipolar approximation on a sphere
6 : Calculate $\phi$, $\vec{E}$, and $\vec{F}_{\mathrm{DEP}}$ using a quadrupolar approx. on a sphere
7 : Calculate $\phi$, $\vec{E}$, and $\vec{F}_{\mathrm{DEP}}$ using an octupolar approx. on a sphere
8 : Not available (reserved for multipolar approx. of order 4 on a sphere)
9 : Not available (reserved for multipolar approx. of order 5 on a sphere)
10 : Calculate $\phi$, $\vec{E}$, and $\vec{F}_{\mathrm{DEP}}$ using a dipolar approx. for a homogeneous ellipsoid
11 : Calculate $\phi$, $\vec{E}$, and $\vec{F}_{\mathrm{DEP}}$ using a dipolar approx. for a single-shelled ellipsoid

A quadrupolar approximation in this context indicates that dipole + quadrupole terms are included in the calculation, while an octupolar approximation indicates the inclusion of dipolar + quadrupolar + octupolar. When `analysisType` takes values [0–4] the parameters `(a,b,c)` do not need to be included in the section. When the value is between 5 and 9 only `a` has to be specified, and it represents the radius of the spherical particle under consideration. If the calculation of the force is done for an ellipsoid and options 10 or 11 are chosen, all `(a,b,c)` parameters must be specified, and they represent the three semiaxes of the ellipsoid.

The parameter `pointNumber` is only used with options [5–11] and indicates on how many positions of the centre of the sphere/ellipsoid should the force be calculated. The particular positions to use are indicater in the following lines by `(x,y,z)`. If options [1–4] are chosen for the type of analysis neither `pointNumber` nor these last lines should not be included in the file.

## 3.8 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 postprocess later on.

## 3.9 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`, `pot-post` and `field-post`.

## 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 `depSolver/utils/breaker` 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 `depSolver/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 [2.3], 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 potential.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 FieldPost

To process the file `field.dat` you must go into directory `depSolver/utils/fieldPost` and run:

```
./fieldPost nx ny nxi nyi
```

Where `nx` and `ny` are the number of points in the plane in the x and y directions –or xz, yz– and `nxi` and `nyi` are the number of interpolation points desired in each direction. The input file must be called `field.dat` and can be copied directly from the main program's output without changes. If several planes of data were calculated it is necessary to first call `break` –see section [4.1]– in order to get the independent files for each plane, and then name one of this files as `field.dat` before running `fieldpr`. The outputs from this will be:

`E.dat` : File containing the modulus of the electric field as a `nx` by `ny` matrix
`X.dat` : File containing the x position vector of size `nx` (monotonically increasing)
`Y.dat` : File containing the y position vector of size `ny` (monotonically increasing)
`XI.dat` : File containing the interpolated x position vector of size `nxi`
`YI.dat` : File containing the interpolated y position vector of size `nyi`

This is useful for later post-processing in matlab, since the vectors can be imported and then an interpolation done with the commands:

```
>> [X,Y] = meshgrid(X,Y);
>> [XI,YI] = meshgrid(XI,YI);
>> EI = interp2(X,Y,E,XI,YI,'cubic');
```

And then plotted using:

```
>> surf(XI,YI,VI)
>> contourf(XI,YI,EI,25)
```

Notice that the parser will get rid of the constant valued coordinate but will not store it anywhere, so the user must be aware of what constant plane he is using and rename the output files correspondingly.

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

## 4.4 ForcePost

To process the file `force.dat` you must go into directory `depSolver/utils/forcePost` and run:

```
./forcePost nx ny nxi nyi step
```

Where `nx` and `ny` are the number of points in the x and y directions respectively, and `nxi` and `nyi` are the number of required interpolated values. The input `step` is used to control how many of the points are included in the files for the arrow plot data. Using `step` = 1 saves all data, using `step` = 2 saves 1 point of every 4 (divides by two in both x and y) and so on. This is useful because too many points produce very messy arrow plots.

The program produces the following outputs:

`X.dat` : File containing the x coordinate vector of size `nxny/step`$^2$
`Y.dat` : File containing the y coordinate vector of size `nxny/step`$^2$
`FX.dat` : File containing $F_x$ as a vector of size `nxny/step`$^2$
`FY.dat` : File containing $F_y$ as a vector of size `nxny/step`$^2$
`FZ.dat` : File containing $F_z$ as a vector of size `nxny/step`$^2$
`XX.dat` : File containing the x coordinate vector of size `nx`
`YY.dat` : File containing the y coordinate vector of size `ny`
`XXI.dat` : File containing the interpolated x coordinate vector of size `nxi`
`YYI.dat` : File containing the interpolated y coordinate vector of size `nyi`
`FXX.dat` : File containing $F_x$ as a vector of size `nxny`
`FYY.dat` : File containing $F_y$ as a vector of size `nxny`

The files `X.dat`, `Y.dat`, `FX.dat`, `FY.dat`, and `Fz.dat` can be used directly in order to obtain arrow plots in matlab by using the following commands:

```
>> quiver(X,Y,FX,FY,3,'k')
```

where 3 is the arrow size and `'k'` indicates that the arrows are black.

A nice figure is made by superimposing an arrow plot of the force over a density plot of the electric field intensity. This is achieved by typing (if `X.dat` and `Y.dat` correspond to the electric field values):

```
>> [X,Y] = meshgrid(X,Y);
>> [XI,YI] = meshgrid(XI,YI);
>> EI = interp2(X,Y,E,XI,YI,'cubic);
>> contourf(XI,YI,EI,35)
>> hold on
>> quiver(X,Y,FX,FY,3,'k')
```

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

## 4.5 PotPost

In order to divide the file `potential.dat` into files containing only a plane you must go into directory `depSolver/utils/potPost` and run:

```
./potPost nx ny nxi nyi
```

Where `nx` and `ny` are the number of points in the plane in the x and y directions –or xz, yz– and `nxi` and `nyi` are the number of interpolation points desired in each direction. The outputs from this will be:

| | |
|---|---|
| `V.dat` | : File containing the real part of the potential as a `nx` by `ny` matrix |
| `X.dat` | : File containing the x position vector of size `nx` (monotonically increasing) |
| `Y.dat` | : File containing the y position vector of size `ny` (monotonically increasing) |
| `XI.dat` | : File containing the interpolated x position vector of size `nxi` |
| `YI.dat` | : File containing the interpolated y position vector of size `nyi` |

After `potPost` has run we can use matlab to produce figures of the field intensity in the same manner as we did with the electric field. The input file is not modified by the program.

Notice that the parser will get rid of the constant valued coordinate but will not store it anywhere, so the user must be aware of what constant plane he is using and rename the output files correspondingly. The input file is not modified.

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

# 5 Example Files

This section contains examples of a dielectrophoretic trap consistent of a set of electrodes (8 circles forming an octupole), a fluid surrounding them, and a spherical particle. Example 1 indicates the input files necessary for a multipolar approximation of the force of order 3, and example 2 shows the input files necessary for a Maxwell's stress tensor calculation of the force.

## 5.1 Empty trap

In this example we only need to mesh the eight electrodes and impose a given potential on them. This is the listing for the file `input.bem` :

```
NODES
8336            //Total number of nodes
nodes.bem       //Nodes data file

ELEMENTS
3904            //Total number of elements
tria6           //Quadratic interpolation in triangles
elems.bem       //Elements data file

MATERIALS
2               //Total number of different materials (needed for force calculation)
1 1.4e-4 80.0   //Conductivity and eps for material 1 (fluid)
2 2.4e-3 2.50   //Conductivity and eps for material 2 (spherical particle)

INTERFACES
0               //We simulate an empty trap

PROBLEM
1.0e6           //External field frequency
bcs.bem         //Boundary conditions data file

ANALYSIS
gmres 0         //GMRS solver without initial guess
5               //Calculate potential, electric field and force in dipolar approx.
10 5.0e-6       //Number of points where the force is obtained and sphere radius
```

```
forcepoints.bem//Force calculation points data file

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
...
8335      -3.899404e-05   2.786016e-05   2.500000e-05
8336      -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
...
3903     7489  8336  7488  8031  7425  8033
3904     7488  8336  7489  8032  7487  8024
```

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

```
//nodeID bcType value1Re
1        1      1              //Here start the real components of the
2        1      1              //boundary conditions for the electrodes
...
8335     1      1
8336     1      1
```

```
//nodeID bcType value1Im
1        1      0              //Here start the imaginary components of the
2        1      0              //boundary conditions for the electrodes
...
8335     1      0
8336     1      0
```

The file containing the positions where the force calculation is required, `forcepoints.bem` is:

```
//pointID  x          y         z
1        0.0       0.0       0.0
2        0.0       0.0       5.0E-6
...
9        1.25E-5   1.25E-5   1.5E-5
10       1.25E-5   1.25E-5   1.75E-5
```

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
```

## 5.2 Trap with spherical particle

In this case we use the result from the calculation in example 1 as initial guess for the GMRES solver. The `input.bem` file in this case is:

```
NODES
9479           //Total number of nodes
nodes.bem      //Nodes data file

ELEMENTS
```

```
4462           //Total number of elements
tria6          //Quadratic interpolation in triangles
elems.bem      //Elements data file

MATERIALS
2              //Only 2 different materials
1 1.4e-4 80.0  //Conductivity and eps for material 1 (fluid)
2 2.4e-3 2.50  //Conductivity and eps for material 2 (spherical particle)

INTERFACES
1              //Interface between liquid and particle
1 1 2          //interfaceID mat1 mat2

PROBLEMS
1.0e6          //External field frequency
bcs.bem        //Boundary conditions data file

ANALYSIS
gmres 8336     //GMRES with inital guess given at 8336 points
3              //Potential, field, and force using Maxwell's stress tensor method

INTERNALPOINTS
1323           //Total number of points for post-processing
internal.bem   //Internal points data file

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

The `nodes.bem` file in this case is exactly the same as the one in example 1 up to row 8336, and then has the new nodes corresponding to the surface of the particle:

```
//nodeID   x              y              z
1         -2.500000e-05  -2.500000e-05  -2.500000e-05
2         -5.000000e-06  -2.500000e-05  -2.500000e-05
...
8336      -1.476191e-05   1.842516e-05   2.500000e-05     //Last node from electrodes
8337       5.960000e-14   3.750000e-15  -5.000000e-06     //First node from particle
8338      -1.680000e-12  -1.080000e-27   5.000000e-06
...
9478       2.075937e-06 -3.695387e-06   -2.652281e-06
9479       3.301234e-06 -3.678036e-06   -7.575639e-07
```

Likewise for the file `elems.bem`, which is the same as in the previous example up to row 3904:

```
//elemID node1 node2 node3 node4 node5 node6
1        2     335   58    344   59    336
2        59    468   125   345   60    337
...
3904     7488  8336  7489  8032  7487  8024     //Last element from electrodes
3905     8339  8656  8363  8657  8340  8631     //First element from particle
3906     8340  8658  8364  8659  8341  8632
...
4461     8533  9479  8563  9218  8511  9476
4462     8563  9479  8533  9133  8562  9217
```

The boundary conditions file has a similar format, but in this case we need to consider the aditionl parameter `value2` when the boundary condition is given for a node in the dielectric interface:

```
//nodeID bcType value1Re value2Re
1        1      1              //Here start the real components of the
2        1      1              //boundary conditions for the electrodes
...
8336     1      1
8337     0      0        1     //Here start the real components of the
8338     0      0        1     //boundary conditions for the interface
...
9478     0      0        1
9479     0      0        1

//nodeID bcType value1Im value2Im
1        1      0              //Here start the imaginarycomponents of the
2        1      0              //boundary conditions for the electrodes
...
8336     1      0
8337     0      0        1     //Here start the imaginary components of the
8338     0      0        1     //boundary conditions for the interface
...
9478     0      0        1
9479     0      0        1
```

The file with the internal points is the same than in the previous section.

# A Function Listing

| | |
|---|---|
| Directory | : depSolver/source |
| Source Files | : 61 |
| Compilation Script | : depSolverComp |

comFilter.c
constants.h
depElectric.c
depolarization.c
doubleMatrix.c
doublePointer.c
doubleVector.c
electricFormA_tria3.c
electricFormA_tria6.c
electricFormFullA_tria3.c
electricFormFullA_tria6.c
elemType.c
errorHandler.c
field_tria3.c
field_tria6.c
forceEllipsoid_tria3.c
forceEllipsoid_tria6.c
forceMST_tria3.c
forceMST_tria6.c
forceMultipole_tria3.c
forceMultipole_tria6.c
freeDoubleMatrix.c
freeDoublePointer.c
freeUintMatrix.c
gaussBksb.c
gaussData.h
gaussJordan.c
getLocalNormal_tria3.c
getLocalNormal_tria6.c
getNormal_tria3.c
getNormal_tria6.cc
gmres2.hc
intDE_tria3.c
intDE_tria6.c
intF_tria3.c
intF_tria6.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
lubksb.c
ludcmp.c
postProcessCol_tria3.c
postProcessCol_tria6.c
postProcess_tria3.c
postProcess_tria6.c
potential_tria3.c
potential_tria6.c
project.h
shape_line2.c
shape_tria3.c
shape_tria6.c
uintMatrix.c
uintVector.c
X2L_line2.c
X2L_tria3.c
X2L_tria6.c

---

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

break.c
errorHandler.c

---

| | |
|---|---|
| Directory | : depSolver/utils/fieldPost |
| Source Files | : 4 |
| Compilation Script | : fieldPostComp |

doubleMatrix.c errorHandler.c freeDoubleMatrix.c
fieldPost.c

---

| | |
|---|---|
| Directory | : depSolver/utils/forcePost |
| Source Files | : 4 |
| Compilation Script | : forcePostComp |

doubleMatrix.c errorHandler.c freeDoubleMatrix.c
forcePost.c

---

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

separate.c errorHandler.c

---

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

meshgen.c errorHandler.c

---

| | |
|---|---|
| Directory | : depSolver/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 | : `depSolver/utils/potPost` |
| Source Files | : 4 |
| Compilation Script | : `thPostComp` |

`doubleMatrix.c` `errorHandler.c` `freeDoubleMatrix.c`
`potPost.c`

---

# B Element Library

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

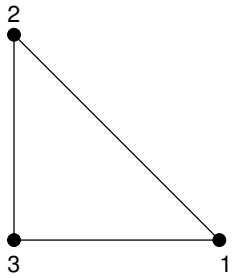


Figure 1: Linear interpolation.

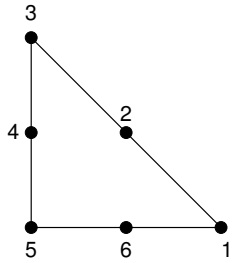


Figure 2: Quadratic interpolation.

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

$$N_1 = L_1 \quad (1)$$
$$N_2 = L_2 \quad (2)$$
$$N_3 = 1 - L_1 - L_2 \quad (3)$$

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

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

# C Numerical Integration

## Regular integrals

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

$$\int_{-1}^{1} F(L_1, L_2) dL_1 dL_2 \approx \sum_{i=1}^{NG} F(L_1^i, L_2^i) w_i \tag{10}$$

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.

$$\int_{-1}^{1} F(L_1, L_2)(1 + L_2) dL_1 dL_2 \approx \sum_{i=1}^{NG} \sum_{j=1}^{NG} F(L_1^i, L_2^j) w_i^{\text{Gauss}} w_j^{\text{Gauss}-\text{Jacobi}} \tag{11}$$

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.

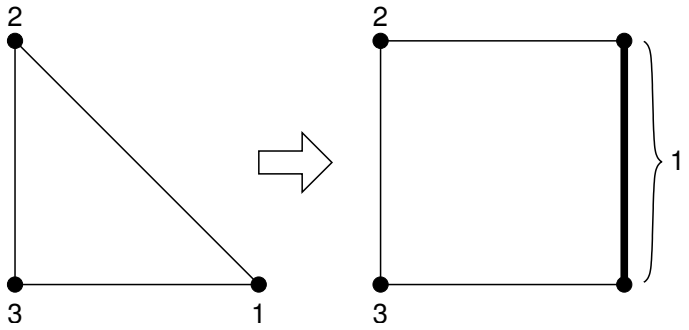


Figure 3: Regularization transformation for weakly singular integrals (linear case).

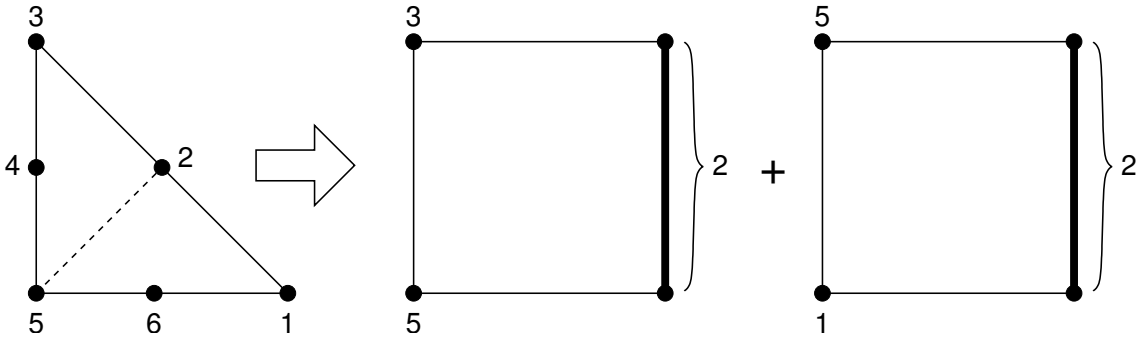


Figure 4: Regularization transformation for weakly singular integrals (quadratic case).

## 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.

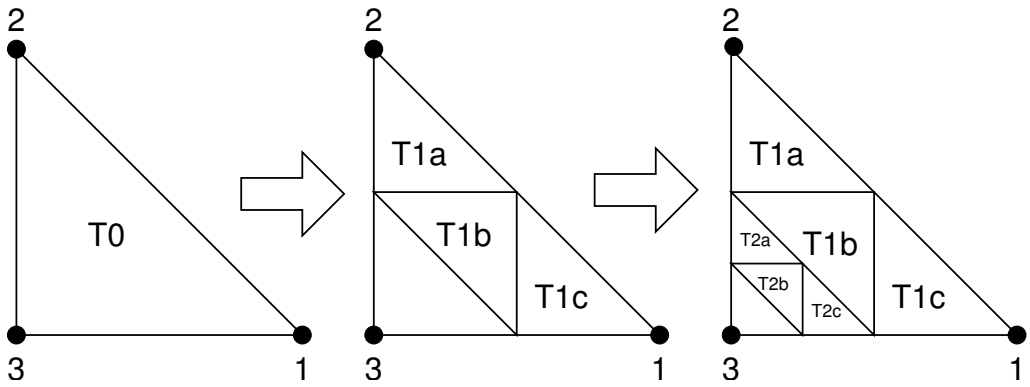


Figure 5: Subdivision process for strongly singular integrands with singularity at node 3 (only two consecutive subdivisions shown).

## 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 steps:

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 |