stkSolver 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

`stkSolver` is a Boundary Element Method solver for 3D Stokes flow. I originally developed this code to calculate accurately the drag experienced by arbitrarily shaped bodies. The current implementation is limited to using given velocities as the boundary conditins of the problem. The source code is written in C and makes use of separate functions for most taks in order to make modifications easier.

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 direct velocity 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 go through Part II in order to understand the capabilities of the code.

The Institute of High Performance Computing (IHPC) distributes `depSolver` 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 `depSolver` under the GPL License. For OEMs, ISVs, and VARs who distribute any element of `depSolver` 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, I would appreciate if you could send an email telling me that you are using `depSolver` 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

Part I

User Guide

Chapter 1

Introduction and general remarks

`stkSolver` is distributed as a series of C function files, a make file for compilation, a Patran [1] 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 [2] and gnuplot [3]. The complete list of functions is collected in appendix A.

1.1 Quick Install Guide

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

```
$ tar -zxvf stkSolver.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:

`stkSolver`	: Main program directory
`stkSolver/bin`	: Directory where the binary files are stored after compilation
`stkSolver/docs`	: Directory containing this document in pdf form
`stkSolver/examples`	: Directory with examples to test the compiled code
`stkSolver/src`	: Directory containing the C source code
`stkSolver/utils`	: Directory with utilities for pre- and post-processing
`utils/break`	: Utility to break data files into several units
`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

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

1.1.1 Compilation

Two versions of the code can be compiled. The basic version is called `stkSolver`, and an additional version called `stkSolver-sft` is also provided. Both versions of the code are identical except that in `stkSolver-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 calculations of the drag without having to create a new mesh for each particle position.

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

```
$ make stkSolver
```

or

```
$ make stkSolver-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 `stkSolver/bin`.

1.1.2 Running the solver

In order to run `stkSolver` 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

There is also one optional input file, necessary only when certain options are set in `input.bem`:

`internal.bem`	: [opt] File containing the internal points where the velocity or pressure 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 `stkSolver` 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
`presure.dat`	: [opt] Contains the presure at the required points
`velocity.dat`	: [opt] Contains the velocity at the required 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:

```
< vx vy vz >
< 0 0 0 >
```

The first row contains the given velocity at each node, and the second row can be anything, because it is not used.

Once this is done run the command `!!input bem_save_stk.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_stk()` 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 stkSolver executable a further step is necessary. Go to the directory called stkSolver/utils/p2b-stk and run:

```
$ ./p2b-stk 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 stkSolver, 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-stk -h.

2.3 Generating evaluation points file

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

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

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

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

Example of use:

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

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

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

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

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
nodeNumberWall
nodeNumberTotal
nodeFilename
```

Where `nodeNumberWall` is the number of nodes in the geometry excluding the particle, and `nodeNumberTotal` is the total number of nodes in the geometry.

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
elemNumberWall
elemNumberTotal
elemType
elemFilename
```

Where `elemNumberWall` is the number of elements in the geometry excluding the particle, and `elemNumberTotal` is the total number of elements in the geometry. The element type `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 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
dynViscosity
maxU Ly Lz
bcsFilename
```

Where `dynViscosity` is the dynamic viscosity of the liquid. `Ly` and `Lz` are the channel width and height, and `maxU` is the maximum velocity for the parabolic profile, These tree values are only used when a parabolic profile is used as boundary condition, but they must always be present in the input file. `bcsFilename` is the file containing the boundary conditions for the Stokes problem.

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

```
nodeID bcType value1 value2 value3
```

A dirichlet boundary condition – in this case, given velocity – is indicated by a `bcType` of 1, with `value1` equal to v_x, `value2` equal to v_y, and `value3` equal to v_z.

If the user wants to specify a parabolic profile in a square channel section the value of `value1` must be negative, and the other two values can be set to anything. In this case it is important to set correctly the values of `Ly` and `Lz` for the channel and the `maxU` in the input file so that the correct profile is applied.

3.4 Analysis Section

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

```
ANALYSIS
solver preCond nInit
analysisType
```

It is not always necessary to include all these parameters in the section. The particular set of them necessary for a calculation depends on the `solver` 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
`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 flow velocity at the given points
1 : Calculate only the presure at the given points
2 : Calculate both velocity and presure at the given points

3.5 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.

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 two main utilities called `break` and `temp-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 `stkSolver/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 `stkSolver/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 [2.3], 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. For the post-processing of the velocity data the total number of points is $100 \times 20 = 2000$ and we have 6 columns (x,y,z,vx,vy,vz), so we call separate as:

```
$ ./gplotForm velocity.dat 2000 20 6
```

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.

Chapter 5

Example Files

There are four example sets of input and output files in `/depSolver/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:

channel: Simple example of flow in an empty rectangular channel.

sphere: Example of drag on a moving sphere.

sphere-channel: Example with a sphere floating inside a rectangular channel.

Part II

Direct Velocity Formulation of the Stokes Problem in BEM

Chapter 6

The DVBEM equations

Stokes flow is a linearized approximation of the full Navier-Stokes equations for very low Reynolds numbers, where inertial effects are negligible in comparison to viscous effects. For and incompressible liquid it is given by:

$$\mu\nabla^2\vec{u} - \vec{\nabla}p = 0 \tag{6.1}$$
$$\nabla\cdot\vec{u} = 0 \tag{6.2}$$

The direct velocity formulation of the stokes problem in the boundary element method was originally developed by Youngren and Acrivos [4] to describe the flow around an arbitrarily shaped body, and is described by the following equation:

$$c(\vec{r})u_i(\vec{r}) = -\frac{1}{8\pi\mu}\int_\Gamma G_{ij}(\vec{r},\vec{r}')f_i(\vec{r}')d\Gamma' + \frac{1}{8\pi}\int_\Gamma u_i(\vec{r}')T_{ijk}(\vec{r},\vec{r}')n_k(\vec{r}')d\Gamma' \tag{6.3}$$

where G_{ij} and T_{ijk} are given by:

$$G_{ij}(\vec{r},\vec{r}') = \frac{\delta_{ij}}{|\vec{r}-\vec{r}'|} + \frac{(\vec{r}-\vec{r}')_i(\vec{r}-\vec{r}')_j}{|\vec{r}-\vec{r}'|^3} \tag{6.4}$$
$$T_{ijk}(\vec{r},\vec{r}') = -6\frac{(\vec{r}-\vec{r}')_i(\vec{r}-\vec{r}')_j(\vec{r}-\vec{r}')_k n_k(\vec{r}')}{|\vec{r}-\vec{r}'|^5} \tag{6.5}$$

while f_i is the surface force and c is a constant that for smooth surfaces takes the values:

$$c(\vec{r}) = \begin{cases} 1 & \text{if } \vec{r} \in \Omega \\ 1/2 & \text{if } \vec{r} \in \Gamma \\ 0 & \text{if } \vec{r} \in \overline{\Omega} \end{cases} \tag{6.6}$$

Where Ω represent the volume enclosed by the closed boundary surface Γ, and $\overline{\Omega}$ represents the volume external to Γ.

If we limit our study to flows surrounding a rigid body the velocity in the second integral is constant over the rigid body surface and the equation can be rewritten as:

$$c(\vec{r})u_i(\vec{r}) = -\frac{1}{8\pi\mu}\int_\Gamma G_{ij}(\vec{r},\vec{r}')f_i(\vec{r}')d\Gamma' + \frac{u_i^{\text{rb}}}{8\pi}\int_\Gamma T_{ijk}(\vec{r},\vec{r}')n_k(\vec{r}')d\Gamma' \tag{6.7}$$

The last integral takes the following values:

$$\frac{1}{8\pi}\int_\Gamma T_{ijk}(\vec{r},\vec{r}')n_k(\vec{r}')d\Gamma' = \delta_{ij} \times \begin{cases} 1 & \text{if } \vec{r} \in \Omega \\ 1/2 & \text{if } \vec{r} \in \Gamma \\ 0 & \text{if } \vec{r} \in \overline{\Omega} \end{cases} \tag{6.8}$$

This indicates that when considering closed non-deformable surfaces the integral corresponding to T_{ijk} is zero in the fluid and, in order to keep the solution continuous, it is also zero when the evaluation point is in the boundary Γ. In this case the equation to solve for any point in the boundary Γ is:

$$u_i(\vec{r}) = -\frac{1}{8\pi\mu}\int_\Gamma G_{ij}(\vec{r},\vec{r}')f_i(\vec{r}')d\Gamma' \tag{6.9}$$

The corresponding equation for the pressure is:

$$p(\vec{r}) = -\frac{1}{8\pi}\int_\Gamma P_i(\vec{r},\vec{r}')f_i(\vec{r}')d\Gamma' + P_{\text{inf}} \tag{6.10}$$

with P_i given by:

$$P_i(\vec{r}, \vec{r}') = 2\frac{(\vec{r} - \vec{r}')}{|\vec{r} - \vec{r}'|^3} \tag{6.11}$$

Thus our implementation is valid for problem involving rigid bodies, but not deformable surfaces of any kind.

Part III

Implementation Details

Chapter 7

Element Library

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

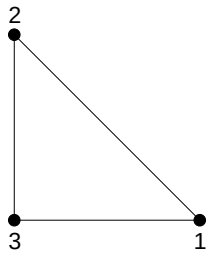

Figure 7.1: Linear interpolation.

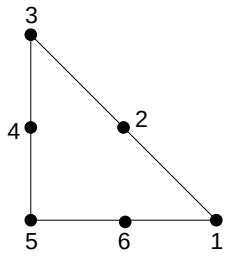

Figure 7.2: Quadratic interpolation.

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.

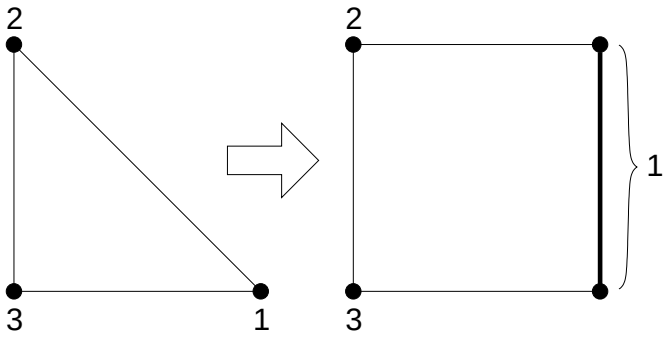

Figure 8.1: Regularization transformation for weakly singular integrals (linear case).

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.

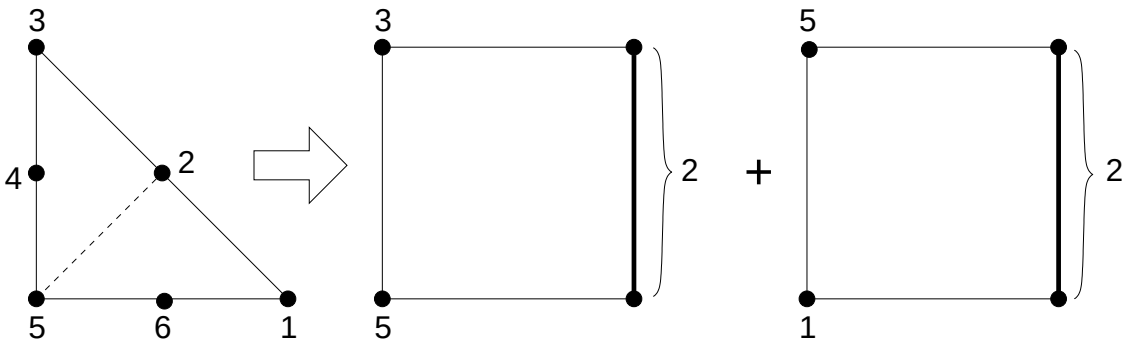

Figure 8.2: Regularization transformation for weakly singular integrals (quadratic case).

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.

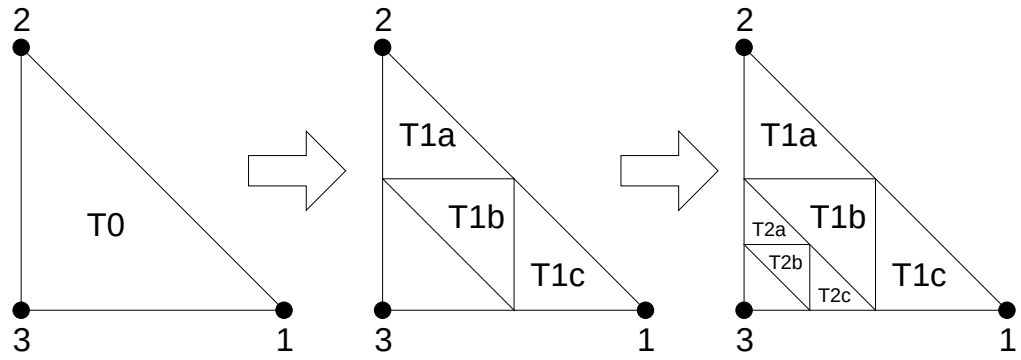

Figure 8.3: Subdivision process for strongly singular integrands with singularity at node 3 (only two consecutive subdivisions shown).

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.

8.4 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 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.

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

Appendix A

Function Listing

Directory	: stkSolver/source
Source Files	: 71
Compilation Script	: stkSolverComp

```
comFilter.c              integral_tria6.h         stokes3d.h
constants.h              intGStk_tria3.c          stokes3d-shift.c
dotProd.c                intGStk_tria6.c          stokesFormA_tria3.c
doubleMatrix.c           intHStk_tria3.c          stokesFormA_tria3.h
doublePointer.c          intHStk_tria6.c          stokesFormA_tria6.c
doubleVector.c           intSingularGStk_tria3.c  stokesFormA_tria6.h
elemType.c               intSingularGStk_tria6.c  stokesPostProcess_tria3.c
errorHandler.c           intSingularHStk_tria3.c  stokesPostProcess_tria3.h
force_tria3.c            intSingularHStk_tria6.c  stokesPostProcess_tria6.c
force_tria3.h            iterGMRES.c              stokesPostProcess_tria6.h
force_tria6.c            iterGMRES.h              torque_tria3.c
force_tria6.h            L2Norm.c                 torque_tria3.h
freeDoubleMatrix.c       makefile                 torque_tria6.c
freeDoublePointer.c      matVectProd.c            torque_tria6.h
freeUintMatrix.c         pressure_tria3.c         uintMatrix.c
gaussBksb.c              pressure_tria3.h         uintVector.c
gaussData.h              pressure_tria6.c         update.c
getLocalNormal_tria3.c   pressure_tria6.h         velocity_tria3.c
getLocalNormal_tria6.c   profileSetup3D.c         velocity_tria3.h
getNormal_tria3.c        shape_tria3.c            velocity_tria6.c
```

```
getNormal_tria6.c      shape_tria6.c          velocity_tria6.h
initRes.c              solverGMRES.c          X2L_tria3.c
initRes.h              solverGMRES.h          X2L_tria6.c
integral_tria3.h       stokes3d.c
```

Directory	: stkSolver/utils/break
Source Files	: 2
Compilation Script	: breakComp

```
break.c                errorHandler.c
```

Directory	: stkSolver/utils/gplotFormat
Source Files	: 2
Compilation Script	: gplotComp

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

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

```
meshgen.c              errorHandler.c
```

Directory	: stkSolver/utils/p2b-stk
Source Files	: 5
Compilation Script	: p2b-stkComp

```
bem_save_stk.pcl       doubleMatrix.c         errorHandler.c
freeDoubleMatrix.c     p2b-stk.c
```

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

Bibliography

[1] 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`.

[2] 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`.

[3] 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/`.

[4] G. K. Youngren and A. Acrivos. Stokes flow past a particle of arbitrary shape: a numerical method of solution. *Journal of Fluid Mechanics*, 69(2):377–403, 1975.